A Model-Driven Framework for
Context-Dependent Component Testing

Abu Zafer Javed
Master of Computer Science
Bachelor of Computer Science

A thesis submitted for the degree of Doctor of Philosophy at
The University of Queensland in 2014
School of information Technology and Electrical Engineering
Abstract

A model-driven framework for context-dependent testing of components (MD-CDCT) is the outcome of this research. Component-based Software Development (CSD) is a process in which software applications are developed by reusing existing components. Primarily, this research devises a technique for evaluating and extending the test adequacy of a component for its reuse in a new context (system). Secondly, it uses the Model-Driven Architecture (MDA) in the context of component testing and CSD. MDA is an emerging approach for developing software applications from high-level models. In MDA, models at various levels of abstraction are used to automate software development and testing activities.

When a general-purpose component is used in a new context, it needs to be thoroughly tested for that context. MD-CDCT models the usage of a component, in a context of interest, using usage scenarios and interaction diagrams. From these scenarios, test cases for performing CDCT are automatically generated using a model-driven tool. We then evaluate and extend the adequacy of the generated test cases by comparing them with the test cases that were executed during component testing by the component developer which are provided with the component as metadata. Finally, we execute the extended set of test cases to test the component for the new context.

This approach is novel in that it applies the emerging MDA technology to context-dependent testing of components. The proposed framework benefits from the advantages of an MDA-based approach, such as portability, interoperability and maintainability. Another novelty is the use of test cases, which are executed during component testing, to evaluate and extend the adequacy of component testing. We have developed two prototype tools (a tool for test case execution, and a tool for test suite comparison) to provide tool support for this approach.

Five case studies are used to illustrate and evaluate the MD-CDCT. The first is a simulation of a Vending Machine system. This case study illustrates the application of MD-CDCT. The second is a simulation of an ATM system. This case study shows how to generate test cases from software models by making use of the model transformation technology. The third and fourth case studies show how the comparison of test suites can be used to evaluate and extend their test adequacy. The final case study evaluates the framework and the prototype tool support on a non-trivial system, which is Lucene (a search engine).
**Declaration by author**

This thesis is composed of my original work, and contains no material previously published or written by another person except where due reference has been made in the text. I have clearly stated the contribution by others to jointly-authored works that I have included in my thesis.

I have clearly stated the contribution of others to my thesis as a whole, including statistical assistance, survey design, data analysis, significant technical procedures, professional editorial advice, and any other original research work used or reported in my thesis. The content of my thesis is the result of work I have carried out since the commencement of my research higher degree candidature and does not include a substantial part of work that has been submitted to qualify for the award of any other degree or diploma in any university or other tertiary institution. I have clearly stated which parts of my thesis, if any, have been submitted to qualify for another award.

I acknowledge that an electronic copy of my thesis must be lodged with the University Library and, subject to the General Award Rules of The University of Queensland, immediately made available for research and study in accordance with the *Copyright Act 1968*.

I acknowledge that copyright of all material contained in my thesis resides with the copyright holder(s) of that material. Where appropriate I have obtained copyright permission from the copyright holder to reproduce material in this thesis.
Publications during candidature


Publications included in this thesis

[1] Incorporated as part of Chapter 3.

Strooper provided guidance on the presentation of the paper's contributions. He also provided proof reading. Watson provided proof reading and feedback on the paper's contributions. Duddy was involved with the Eclipse modelling technologies. The breakdown of contributions in terms of the work done during research and writing the paper is summarised below:

<table>
<thead>
<tr>
<th>Contributor</th>
<th>Statement of contribution</th>
</tr>
</thead>
</table>
| Abu Zafer Javed (Candidate)  | Developed approach (70%)  
Designed and developed tool support (70%)  
Designed and carried out case study (80%)  
Wrote and edited the paper (80%) |
| Paul Strooper                | Developed approach (10%)  
Designed and developed tool support (10%)  
Designed and carried out case study (10%)  
Wrote and edited the paper (10%) |
| Geoffrey Watson              | Developed approach (10%)  
Designed and developed tool support (10%)  
Designed and carried out case study (10%)  
Wrote and edited the paper (10%) |
| Keith Duddy                  | Developed approach (10%)  
Designed and developed tool support (10%) |
Contributions by others to the thesis

Prof. Paul Strooper was the primary supervisor during the candidature. Paul's contribution steered the academic arguments of the thesis topics and encouraged the exploration of ideas. He provided a sounding board for ideas and topics of conversation.

Dr Geoffrey Norman Watson was a secondary supervisor during the candidature. He provided advice about the thesis topics relationship to the state of the art in industry. He proof read and provided feedback on papers and this thesis.

Keith Duddy was not directly connected with the university. His contribution was as an industry advisor whose experience guided the application of model-transformation technology to the testing domain.

Statement of parts of the thesis submitted to qualify for the award of another degree

None
Acknowledgements

As my thesis journey was ending, my daughter's life journey was beginning. This thesis is dedicated to my daughter, Anaya, with a hope that she would one day realise that education is most powerful weapon you can use to change the world.

There are many people whom I would like to thank for their help during the course of my research. In particular, I would express my heartfelt gratitude to supervisors Professor Paul Strooper for his discussions to guide my research. He has been very encouraging and motivating advisor. Whenever, the vastness of research got me lost, he provided me the right direction to finding the solution to research-related problems. I also thank Dr Geoffrey Watson for his advice throughout my candidature, and Keith Duddy for his consultation at the initial stage of this research project. Finally, I would also like to thanks my parents Javed Alam and Safia Khanam for their love, and support which is of immeasurable help throughout my doctoral studies.
Keywords

Component-based testing
Component-based software development
Model-drive testing
Model-based testing
Model-driven software development

Australian and New Zealand Standard Research Classifications (ANZSRC)

ANZSRC code: 080309, Software Engineering, 100%

Fields of Research (FoR) Classification

FoR code: 0803, Computer Software, 100%
# TABLE OF CONTENTS

**List of Figures**  .......................................................................................................................... 7

**List of Tables**  .......................................................................................................................... 9

**List of Abbreviations**  .............................................................................................................. 11

**Chapter 1 - Introduction**  ......................................................................................................... 13

1.1. Background  ............................................................................................................................. 13
   1.1.1. Component-based software development  ........................................................................ 13
   1.1.2. Context-dependent component testing  .......................................................................... 14
   1.1.3. Model-driven architecture  ............................................................................................. 15

1.2. The Problems in Context-Dependent Testing of Components  .............................................. 16

1.3. Overview of Approach  .......................................................................................................... 17

1.4. Contributions  ......................................................................................................................... 20

1.5. Thesis Structure  ..................................................................................................................... 20

**Chapter 2 - The Framework**  .................................................................................................... 23

2.1. Introduction  ............................................................................................................................ 23


2.3. An Example Application  ........................................................................................................ 28
   2.3.1. Vending machine system  ............................................................................................... 28
      2.3.1.1. Implementation of the Vending Machine System  ..................................................... 29
      2.3.1.2. Test Suite for component testing of Dispenser  ....................................................... 33

   2.3.2. Applying the framework  ................................................................................................ 34
      2.3.2.1. Modelling usage scenarios of CB software  ................................................................ 34
      2.3.2.2. Deriving component test cases for CDCT  ............................................................... 37
      2.3.2.3. Comparing test suites  ............................................................................................. 37
      2.3.2.4. Enriching CT  ......................................................................................................... 39
      2.3.2.5. Execute enriched test suite  ...................................................................................... 41

2.4. Related Work  ......................................................................................................................... 41
   2.4.1. Built-in testing  ............................................................................................................... 41
   2.4.2. Component metadata  ...................................................................................................... 42
   2.4.3. Component certification  .................................................................................................. 43
   2.4.4. Component contracts  ..................................................................................................... 45
   2.4.5. Testable architecture  ..................................................................................................... 46
   2.4.6. Other approaches  .......................................................................................................... 48
2.4.7. IEC Standards

2.4.7.1. IEC-62814
2.4.7.2. IEC-61508
2.4.7.3. IEC-62628

2.4.8. IEEE Standards

2.4.8.2. IEEE Standard 1517-2010

2.4.9. Summary of the related work

2.5. Discussion

Chapter 3 - Model-Driven Test Case Generation

3.1. Introduction

3.2. Model-Driven Testing

3.3. MTCG: A Method for Automating Test Case Generation

3.3.1. Generating test cases using model-driven architecture

3.3.1.1. Step 1: Transforming SMC into xUnit
3.3.1.2. Step 2: Generating JUnit from xUnit
3.3.1.3. SMC Meta-model
3.3.1.4. xUnit Meta-model

3.3.2. Tracing

3.4. Case Study: Determining Applicability of MTCG

3.4.1. Introduction

3.4.2. Object description

3.4.3. Interface of ATM

3.4.4. Experimental planning

3.4.5. Applying MTCG to the case study

3.4.5.1. An example sequence diagram
3.4.5.2. SMC model
3.4.5.3. xUnit model
3.4.5.4. Test data file
3.4.5.5. JUnit test case
3.4.5.6. Traces

3.4.6. Reuse of vertical transformation rules

3.4.7. Discussion

3.4.8. Threats to validity

3.5. Related Work

3.5.1. Model-based testing

3.5.1.1. Model validation using interaction diagrams
3.5.1.2. Implementation verification using interaction diagrams
### Chapter 4 - Comparison of Test Suites

- **4.1. Introduction**
- **4.2. TestSuiteComparator: A Method for Comparing Test Suites**
- **4.3. Case Studies**
  - **4.3.1. Case Study: Determining applicability of TestSuiteComparator**
  - **4.3.2. Case Study: Evaluating effectiveness of TestSuiteComparator**
- **4.4. Related Work**
- **4.5. Discussion**

### Chapter 5 - A Case Study

- **5.1. Introduction**
- **5.2. Object Description**
  - **5.2.1. Lucene queries**
  - **5.2.2. Components (modules) of Lucene**
  - **5.2.3. Tasks performed by Searcher**
  - **5.2.4. Overloaded implementations of the search method**
- **5.3. Experimental Planning**
- **5.4. Applying the Framework to the Case Study**
  - **5.4.1. Modelling usage scenarios of CB software**
  - **5.4.2. Deriving component test cases for CDCT**
  - **5.4.3. Comparing test suites**
    - **5.4.3.1. Instrumenting CT**

---

3.5.2. Model-driven testing

3.6. Discussion

3.7. Limitations of MTCG Prototype

---

Chapter 4 - Comparison of Test Suites

- **4.1. Introduction**
- **4.2. TestSuiteComparator: A Method for Comparing Test Suites**
- **4.3. Case Studies**
  - **4.3.1. Case Study: Determining applicability of TestSuiteComparator**
  - **4.3.2. Case Study: Evaluating effectiveness of TestSuiteComparator**
- **4.4. Related Work**
- **4.5. Discussion**

Chapter 5 - A Case Study

- **5.1. Introduction**
- **5.2. Object Description**
  - **5.2.1. Lucene queries**
  - **5.2.2. Components (modules) of Lucene**
  - **5.2.3. Tasks performed by Searcher**
  - **5.2.4. Overloaded implementations of the search method**
- **5.3. Experimental Planning**
- **5.4. Applying the Framework to the Case Study**
  - **5.4.1. Modelling usage scenarios of CB software**
  - **5.4.2. Deriving component test cases for CDCT**
  - **5.4.3. Comparing test suites**
    - **5.4.3.1. Instrumenting CT**
5.4.3.2. Capturing execution traces of CT ...................................................... 138
5.4.3.4. Comparing test suites ................................................................. 139
5.4.4. Enriching CT ................................................................................ 141
5.4.5. Executing enriched test suite ......................................................... 141
  5.4.5.1. Transforming the extending test suite into an executable test suite using MTCG ................................................................. 142
  5.4.5.2. Executing the enriched test suite ............................................... 142
5.5. Data Interpretation ............................................................................. 143
5.6. Discussion .......................................................................................... 144
  5.6.1. Challenges in comparing test suites ............................................ 145
  5.6.2. Limitations of MTCG ................................................................. 147
  5.6.3. An improvement in TSC ............................................................... 150
  5.6.4. Limitations of the case study ..................................................... 150
5.7. Threats to Validity ............................................................................. 151

CHAPTER 6 - CONCLUSION ....................................................................... 153
  6.1. Summary ......................................................................................... 153
  6.2. Discussion ....................................................................................... 154
    6.2.1. MD-CDCCT .............................................................................. 154
    6.2.2. Evaluation of MD-CDCCT ....................................................... 155
  6.3. Future Work ................................................................................... 156

REFERENCES ........................................................................................... 159

APPENDIX A: VENDING MACHINE EXAMPLE ................................................. 173
  A.1: Test Suite for Dispenser (CT) ............................................................... 173
  A.2: SMC-Model for Vending Machine (CDCT) ....................................... 174
  A.3: Test data for Vending Machine (CDCT) .......................................... 175
  A.4: xUnit-Model for Vending Machine (CDCT) ...................................... 178
  A.5: Test suite for Vending Machine (CDCT) ........................................... 179
  A.6: Execution Traces of test suite for Vending Machine (CDCT) ............ 185
  A.7: Test Suite for ∆CT ........................................................................ 186

APPENDIX B: MODEL-DRIVEN TEST CASE GENERATION ................................... 187
  B.1: Tefkat rules (horizontal transformation) ............................................ 187
  B.2: MOFScript rules (vertical transformation) ...................................... 191
    B.2.1: Rules for xUnit to JUnit ............................................................ 191
    B.2.2: Rules for xUnit to SUnit .......................................................... 202
B.3: JUnit test cases generated by MTCG

B.3.1: JUnit test case for withdrawal operation

B.3.2: JUnit test case for deposit operation

B.3.3: JUnit test case for transfer operation

B.3.4: JUnit test case for balance inquiry operation

B.4: SUnit test cases generated by MTCG

B.4.1: SUnit test case for withdrawal operation

B.4.2: SUnit test case for deposit operation

B.4.3: SUnit test case for transfer operation

B.4.4: SUnit test case for balance inquiry operation

APPENDIX C: COMPARISON OF TEST SUITES

C.1: Implementation of Dijkstra's shortest-path algorithm

C.2: Test suites for the shortest-path case study

C.2.1: Test suite 1 for the shortest-path case study

C.2.2: Test suite 2 for the shortest-path case study

C.2.3: Test suite 3 the shortest-path case study

C.3: Execution traces of test suites for the shortest-path case study

C.4: Equivalence classes for the shortest-path case study

C.5: Implementation of Boyer Moore's pattern-matching algorithm

C.6: Test suites for the pattern-matching case study

C.6.1: Test suite 1 for the pattern-matching case study

C.6.2: Test suite 2 for the pattern-matching case study

C.6.3: Test suite 3 for the pattern-matching case study

C.7: Execution traces of test suites for the pattern-matching case study

C.8: Equivalence classes for the pattern-matching case study

C.9: Defects seeded in the pattern-matching case study

APPENDIX D: LUCENE CASE STUDY

D.1: Unique traces of CDCT (CDCT \text{UNIQUE-TRACES})

D.2: Unique traces of CT (CT \text{UNIQUE-TRACES})

D.3: Pairs in CDCT \text{UNIQUE-TRACES} (CDCT \text{PAIRS})

D.4: Pairs in CT \text{UNIQUE-TRACES} (CT \text{PAIRS})

D.5: Abstract test suite for enriching CDCT (\Delta CDCT)

D.6: Sequence diagrams for the use cases of Searcher in \Delta CDCT

D.7: SMC Model for smc1
LIST OF FIGURES

Figure 1: Component testing vs. context-dependent component testing ........................................ 15
Figure 2: Overview of approach ........................................................................................................ 19
Figure 3: Adequacy of testing and component acceptability ............................................................ 25
Figure 4: Component diagram of the vending machine system ....................................................... 28
Figure 5: The application: Vending Machine System ...................................................................... 31
Figure 6: The component: Dispenser .................................................................................................. 32
Figure 7: Vending Machine GUI (front-end) ..................................................................................... 32
Figure 8: Interaction diagram for the Vending Machine System ..................................................... 33
Figure 9: Overall process for generating test cases .......................................................................... 61
Figure 10: Architectural Diagram .................................................................................................... 62
Figure 11: Overview of methodology ............................................................................................... 64
Figure 12: An example Tefkat rule ................................................................................................... 65
Figure 13: Example MOFScript rules ............................................................................................... 66
Figure 14: SMC meta-model ............................................................................................................ 67
Figure 15: xUnit meta-model .......................................................................................................... 68
Figure 16: An example sequence diagram for the ATM case study ............................................... 72
Figure 17: SMC model for the example sequence diagram ............................................................. 74
Figure 18: xUnit model for the example sequence diagram ........................................................... 74
Figure 19: Test data file for the example sequence diagram ........................................................... 75
Figure 20: JUnit test case for the example sequence diagram ........................................................ 77
Figure 21: Execution trace for the example sequence diagram ...................................................... 78
Figure 22: M2T rules for mapAssertion, mapMethod and mapExpectedValue .............................. 79
Figure 23: Overview of TestSuiteComparator ................................................................................ 92
Figure 24: Component diagram of Lucene ....................................................................................... 129
Figure 25: Example trace and its description for the Lucene case study ......................................... 138
Figure 26: Code snippets of test suite for Searcher ......................................................................... 146
Figure 27: Changing M2M and M2T rules for access modifiers .................................................... 149
Figure 28: Changing M2M and M2T rules for declaring arrays ....................................................... 150
LIST OF TABLES

Table 1: Adequacy of CDCT and CT ................................................................. 26
Table 2: Test suite for component testing of Dispenser (CT) ........................................ 35
Table 3: Usage scenarios of the Vending Machine System which use Dispenser .................. 36
Table 4: Test Cases of CDCT of Dispenser .................................................................. 36
Table 5: Test cases extracted from CDCT of Dispenser ................................................. 38
Table 6: Gaps in CT and CDCT ................................................................................. 40
Table 7: Test cases for the gaps in the component testing of Dispenser (∆CT) .................. 40
Table 8: Comparison of CBT approaches .................................................................... 55
Table 9: Code reusability matrix of M2T rules ............................................................... 80
Table 10: Test cases extracted from test suites for shortest-path case study ....................... 96
Table 11: Test cases and their equivalence classes for shortest-path case study ................ 99
Table 12: Equivalence-class representation of test suites for shortest-path case study ........ 99
Table 13: Comparison of test suites for shortest-path case study .................................. 100
Table 14: Test cases extracted from test suites for pattern-matching case study .......... 106
Table 15: Test cases and equivalence classes for pattern-matching case study .............. 108
Table 16: Equivalence-class representation of test suites for pattern-matching case study ... 109
Table 17: Comparison of test suites for pattern-matching case study ......................... 109
Table 18: The enriched test suites for pattern-matching case study ................................ 111
Table 19: Coverage of the original and extended test suites for pattern-matching case study 112
Table 20: Results of coverage analysis for pattern-matching case study ....................... 113
Table 21: Mutants killed by the original and extended test suites for pattern-matching case study 114
Table 22: Results of the mutation analysis for pattern-matching case study .................. 114
Table 23: Executing conditions of a test case for the Lucene case study ....................... 114
Table 24: Data-partitioning of query text (content of query object) for the Lucene case study 135
Table 25: Pairs missing in CT .................................................................................... 140
Table 26: Execution results of ∆CT/ ............................................................................. 143
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBT</td>
<td>Component-Based Testing</td>
</tr>
<tr>
<td>CB Software</td>
<td>Component-Based Software</td>
</tr>
<tr>
<td>CDCT</td>
<td>Context-Dependent Component Testing</td>
</tr>
<tr>
<td>CDCT_UNIQUE_TRACES</td>
<td>Unique execution traces of CDCT</td>
</tr>
<tr>
<td>CIM</td>
<td>Computation Independent Model</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial-Off-The-Shelf</td>
</tr>
<tr>
<td>CSD</td>
<td>Component-based Software Development</td>
</tr>
<tr>
<td>CT</td>
<td>Component Testing</td>
</tr>
<tr>
<td>CT_UNIQUE_TRACES</td>
<td>Unique execution traces of CT</td>
</tr>
<tr>
<td>MDA</td>
<td>Model-Driven Architecture</td>
</tr>
<tr>
<td>MD-CDCT</td>
<td>A Model-Driven Framework for Context-Dependent Component Testing</td>
</tr>
<tr>
<td>MD-CDCT_Refined</td>
<td>An Improved version of MD-CDCT</td>
</tr>
<tr>
<td>MTCG</td>
<td>Model-Driven Test Case Generation</td>
</tr>
<tr>
<td>MTCG_Prototype</td>
<td>A Prototype implementation of MTCG</td>
</tr>
<tr>
<td>MTT</td>
<td>Model Transformation Tool</td>
</tr>
<tr>
<td>M2M</td>
<td>Model to Model Transformation</td>
</tr>
<tr>
<td>M2T</td>
<td>Model to Text Transformation</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>PIM</td>
<td>Platform Independent Model</td>
</tr>
<tr>
<td>PSM</td>
<td>Platform Specific Model</td>
</tr>
<tr>
<td>SMC</td>
<td>Sequence of Method Calls</td>
</tr>
<tr>
<td>TSC</td>
<td>Test Suite Comparator</td>
</tr>
<tr>
<td>V&amp;V</td>
<td>Verification and Validation</td>
</tr>
</tbody>
</table>
CHAPTER 1 - INTRODUCTION

In this thesis, we propose a novel model-driven framework for context-dependent testing of components (CDCT). This research was primarily aimed at evaluating and extending the adequacy of component testing in the context of Component-based Software Development (CSD). A secondary objective of this work was to apply Model-Driven Architecture (MDA) to component testing and CSD.

1.1. Background

Before discussing the problems and contributions, I briefly introduce three areas of software engineering that provide the foundation for this thesis.

1.1.1. Component-based software development

The idea of components and their reuse was widely advocated by McIlroy [1] when he wrote a paper on software components in the first ever software engineering conference held in 1968. He wrote:

"My thesis is that the software industry is weakly founded, in part because of the absence of a software components sub-industry. A components industry could be immensely successful."

Component-based Software Development (CSD) is an approach in which software applications are developed by reusing readily available components [2-5]. Two important advantages of CSD are shorter development time and lower development cost [6-9].

The term “component” has many definitions and is used in different ways by practitioners [10, 11]. We adopt Szyperski’s definition, who defines a component as “a unit of composition with contractually specified interfaces and explicit context dependencies” [12]. The interface of a component is a collection of service access points along with their semantic specifications [13]. These interfaces are the methods, implemented by the component developer, through which the reusing system interacts with the component.

In CSD, the component provider develops component(s) and a set of interfaces to the component(s). The component user reuses the component(s), using their interfaces, to develop software applications [14]. Software developed using this process is called Component-Based (CB) software.
1.1.2. Context-dependent component testing

Although CSD has advantages over traditional software development, it requires specialised or extended testing techniques. Component testing (CT) refers to all activities that are related to testing of a component in isolation, independent of a particular system in which it is used. The component developers perform component testing to confirm that components exhibit the desired behaviour. This is often referred to as unit or module testing [15], and it increases the reliability of the component in CB software.

From a component user’s perspective, the component developer’s testing may fail to test the component sufficiently because:

1. Components are generally developed to provide a wide range of functionally to achieve greater applicability [16]. The component developer may not be able to extensively test a component for all contexts of reuse because of time and resource constraints. Instead, they perform a uniform testing across the whole range of functionality. Thus, there is a possibility that the developer’s testing may extensively test functionality that is not used in the user’s context, while less thoroughly testing functionality that is intensively used in the user’s context especially when the use of the component is novel.

2. The component may be reused in a context or manner that was not imagined when the component was developed, and hence the component may not be tested well or at all for that particular use. It is hard for a component developer to envisage all possible ways in which a component can be reused. If the component developer provides a precise specification of the behaviour of the component, for example in the form of pre/post-conditions, then there should be no unimagined context. However, it could well be that the original preconditions are subsequently found to be too strong. In such a case the preconditions could be relaxed and the testing should be expanded to check this enhanced use. More importantly, most real software is not specified using such pre/post-conditions, in which case it is possible that the developer has not considered and adequately tested all contexts of reuse.

The component user would benefit from more intensive testing targeting the specific context of reuse of the component. This context-dependent component testing (CDCT), which is also known as CB testing, refers to all activities that are related to testing a component in the scope of CSD [17]. CDCT testing aims to increase the reliability of the component for the
particular context in which it is reused. CDCT allows the component user to focus on finding errors related to the use of the component in a specific context. Figure 1 shows the relationship between CT, which is uniform across different possible uses, and CDCT, which is more thorough for a particular use.

The significance of context in component testing is demonstrated by the failure of the ARIANE 5 rocket which was launched in 1996 [18]. It crashed 40 seconds after take-off. The subsequent investigation revealed that a software component of the ARIANE 5 that was reused from the earlier ARIANE 4 failed to function properly. The reason for the failure was the higher initial velocity of the ARIANE 5 compared to the ARIANE 4. The reused component worked fine in the ARIANE 4 but it failed with the ARIANE 5. More thorough testing of the component in the context of the ARIANE 5 (CDCT) might have detected the problem.

1.1.3. Model-driven architecture

MDA is an initiative by the Object Management Group (OMG) to support the development of interoperable, portable and reusable software systems [19]. In MDA, models at various levels of abstraction are the central software design artifacts. They are used to facilitate both abstraction and automated development. MDA can contribute throughout the software development life cycle. Business analysts can develop a business model which is a computation independent model (CIM). Architects and designers develop Platform Independent Models (PIM) using the CIM. Developers and testers derive Platform Specific Models (PSM) from the PIM, for generating application code and test code respectively.
Software maintenance teams can apply model slicing techniques to identify parts of the model that relate to a functionality that is being changed.

Some of the advantages of an MDA-based approach are:

1. MDA tools can partially automate the development process by generating most of the code from models, resulting in less code to hand-craft [20].
2. MDA tools for reverse engineering can automate synthesis of software models [21].
3. MDA tools can help in generating test cases from software models to verify the software implementation [22]. In this thesis, we shall use Eclipse-based [23] MDA tools for generating test cases from software models.

A simple use of MDA is to model a system in a platform-independent modelling language (e.g. UML). The PIM can then be transformed into PSM by executing transformation specifications that are mappings between the PIM and some implementation language (e.g. Java) [24]. The same process of transforming a PIM to a PSM can be used for automating the generation of test cases (e.g. JUnit [25] test cases for Java).

Though MDA technology has been around for over a decade, the software industry has not fully benefited from it, especially in the area of software testing. Researchers are still exploring different ways to maximise the use of MDA to support software testing [26]. In this thesis, we leverage the automation of component-based testing by making use of the MDA technology.

1.2. The Problems in Context-Dependent Testing of Components

The most significant challenge in CSD is the testing of a component in a new context [17]. There is a possibility that a component is developed and tested in one context (by the component provider) and used in another context (by the component user) for which it was not adequately tested. As noted above, the techniques of Context Dependent Testing have been developed to address this. However, CDCT itself has challenges and limitations. The most important problems are [27]:

1. Unexpected behaviour of component(s) when reused in a new context.
2. Lack of access to internal working of a component.
3. Test adequacy criteria for component reuse.

The reusing system can use the component in such a way that states are activated and paths are exercised that were not tested by the component provider. Therefore, the use of a
component in a new context can expose previously undiscovered errors. Testing of a component in the context in which it is being reused is necessary as it can discover the defects that are specific to this context. Moreover, more thorough testing can be performed by focusing on the particular usage of the component.

The second problem is the lack of access to the internal working of the component. For some components, such as Commercial-Off-The-Shelf (COTS) components, source code and design artifacts may not be available to the component user. This lack of information limits controllability and observability of a component. Controllability is the ease with which the inputs and internal states of a component can be controlled [27]. Observability is the extent to which inputs, outputs and behaviour of a component can be observed [28]. Controllability and observability are two key aspects of testability. Often, a component that is part of CB software may be difficult to exercise because its internal states and outputs may be obscured. Further, it may be difficult to observe outputs of the component [29]. The correct output is not enough to decide on the correct working of a component because sometimes a component behaves incorrectly and still returns the correct output value. This lack of controllability and observability reduces component testability [30, 31].

A criterion for adequate testing of a component at the time of its reuse is another concern [32, 33]. Component interfaces can provide some information about the component model, but they do not provide enough information for devising test adequacy criteria [16]. The component user often picks test adequacy criteria like executing all method calls that are part of the component’s interface, but this may not be sufficient [34].

1.3. Overview of Approach

In this thesis we address the first and third problem: i) the testing of a component in the context of a new system, and ii) determining the adequacy of component testing in the context of its reuse. We provide a model-driven solution to these problems which includes evaluating and extending test adequacy of component testing and generating concrete and executable test cases by making use of the MDA technology. The solution requires that the test suite used for component testing is provided with the component as component metadata.

An overview of our approach is shown in Figure 2. Bold arrows represent tasks performed by humans. Dashed lines represent associations between artefacts. Rectangles represent inputs, intermediate or final outputs of a task (manual or automated).
In this approach, first we model the usage scenarios of CB software (artefact D). For this purpose, we identify the usage scenarios of the CB software which include the component’s functionality. Artefacts A and B represent the CB software and the component (which is being reused) respectively.

Second, we generate a concrete and executable test suite for CDCT (artefact E) from the usage model of the CB software (artefact D). We generate the test suite using a model-driven tool which is described in Chapter 3.

Third, we compare the test suite for CDCT (artefact E) with the test suite which was used for component testing (artefact C) by the component developer. As mentioned before, this step requires that the test suite used to perform component testing is available. The objective of this comparison is to discover gaps (artefact F) in the component testing. These gaps show weaknesses in the component testing by highlighting any areas of the component which were not tested well or at all during component testing. We compare these test suites using a tool which is described in Chapter 4. This tool is semi-automated, i.e. some tasks are automated and some are manual, as they require human judgment.

Fourth, we devise test cases to target the gaps (artefact F) that are identified by comparing the test suites. As the test suite for CDCT is devised from system-level usage scenarios, it is at a higher level of abstraction than the component testing. Therefore, a gap in component testing not only indicates a missing test case but it points to an area of the component which is not tested well or at all. This is the area where we should extend the test adequacy of the component testing or CDCT. Therefore, if we find a usage scenario of CDCT which is not tested during component testing, we expand this usage scenario by devising multiple test cases for this scenario. Depending upon the nature of the gap, testers can decide to what level the usage scenario should be expanded and how many test cases should be created. These test cases extensively test the area (functionality) of the component which is used in the new context but not tested adequately during the component testing. We then add these test cases to the test suite for component testing (artefact C) or the test suite for CDCT (artefact E) to come up with an enriched test suite (artefact G).
Task 1. Model usage scenarios of CB software

Task 2. Derive component test cases for CDCT (with tool support)

Task 3. Compare test suites (with tool support)

Task 4. Enrich component testing or CDCT

Task 5. Execute enriched test suite (with tool support)

Figure 2: Overview of approach
Finally, we execute the enriched test suite (artefact G), using the model-driven tool, to test the component for the new CB software. The test results (artefact H) of the enriched test suite show the component’s acceptability for the CB software.

1.4. Contributions

In this thesis, we propose a novel model-driven framework for context-dependent testing of components for a new context. It analyses the testing performed at the time of component development, identifies weaknesses and extends the adequacy of the testing in the context in which the component is reused. The key aspect of this framework is the use of the emerging MDA technology for component testing.

A second contribution is the use of MDA’s model-transformation technology to support the automation of software testing. We design a method for automated generation of test cases from software models in general and interaction diagrams in particular. Our MDA-based technique transforms source models into target models using transformation specifications as mentioned in Section 1.1.3. These transformation specifications are applied using model transformation tools. We provide prototype tool support for this method.

A third contribution is a method for comparing two test suites to identify gaps. For this purpose, we establish a criterion for comparing test suites (and test cases) and provide prototype tool support. We use this tool to evaluate component testing. It identifies weaknesses of the component testing and indicates areas which require further testing.

The last contribution is the evaluation of the framework on a real and substantial system. We apply the framework to a search engine library to demonstrate its applicability and effectiveness.

1.5. Thesis Structure

This thesis is organised into six chapters. As the framework proposed in this thesis makes contributions in different areas which include: i) component-based testing, ii) model-driven testing, and iii) test suite comparison, we chose to discuss the related work in each chapter instead of having a separate chapter for this.

Chapter 1 provides an introduction to this thesis. It describes background, discusses the problems addressed by this work and the contributions of this thesis. Chapter 2 describes the framework for context-dependent component testing. The framework is illustrated using a
vending machine system. It presents the related work for component testing. Chapter 3 describes the tool for automated generation of test cases (MTCG). The application of this tool is illustrated using an ATM simulation. It discusses the related work in the area of model-based testing. Chapter 4 describes the method for comparing test suites (TSC). The applicability and effectiveness of TSC is demonstrated by applying it to two case studies. Finally, the work related to comparison of test suites is discussed. Chapter 5 presents a case study. The framework is evaluated by applying it to an existing component, a search-engine module consisting of 112 Java source files and 18,389 lines of code. This demonstrates the viability and effectiveness of the framework. The testing performed by the developers was evaluated using TSC, and the test suite for CDCT was enriched to extend its test adequacy. Concrete and executable test cases were generated for the enriched test suite using MTCG. The enriched test suite was executed and this detected some defects which were not discovered previously. Chapter 6 concludes, providing a summary of the work, identifying contributions, and discussing future work.
CHAPTER 2 - THE FRAMEWORK

2.1. Introduction

The reuse of a component in a new context and the adequacy of a component’s testing at the time of its reuse are the major problems which emerged with the introduction of CSD as discussed in Section 1.2. We propose a model-driven framework to address these problems.

In this chapter we describe this framework. We illustrate the applications of the framework using a small example. We present the related work in the area of component-based testing. Finally, we discuss advantages and limitations of the framework.

2.2. MD-CDC: A Model-Driven Framework for Context-Dependent Component Testing

We propose a Model-Driven framework for Context-Dependent Component Testing (MD-CDC) to test a component for a new context, and to evaluate and extend the adequacy of the testing done at the time of component development (i.e. Component Testing). The proposed framework consists of the following tasks (shown in Figure 2):

1. Modelling usage scenarios of CB software
2. Deriving component test cases for CDCT (with tool support)
3. Comparing test suites (with tool support)
4. Enriching component testing or CDCT
5. Executing the enriched test suite (with tool support)

1. Modelling usage scenarios of CB software

First, we model the usage scenarios of CB software which use the component’s functionality (task 1 in Figure 2). These usage scenarios can be derived from use cases. We realise these scenarios using interaction diagrams [35]. The framework is general in that it is not specific to a particular modelling element. For this thesis, we chose to apply this framework using interaction diagrams. However, it can be applied using other modelling elements, such as statecharts. We use interaction diagrams for the following reasons:

i) They are behavioural elements of a UML design that describe dynamic interactions among the components of a system.
ii) They play an important role in the software development processes that are use-case driven [36], such as the Rational Unified Process [37].

iii) As they are constructed at an early stage of software development, testing based on them can start early in the software life cycle.

2. Deriving component test cases for CDCT

Second, we derive test cases from the interaction diagrams, associated with each usage scenario, for performing CDCT (task 2 in Figure 2). We generate these test cases using a model-driven tool which is described in Chapter 3.

A test case consists of input(s), an expected result and executing conditions [38]. For a component, a test case is a sequence of method calls (SMC), for a certain scenario, along with their parameter values, expected return values, and in some cases executing conditions. An executing condition of a test case is the environment in which it runs, e.g. the state of the data store.

3. Comparing test suites

Third, we evaluate the test adequacy of the component testing by comparing it with CDCT (task 3 in Figure 2). CDCT and the component testing are compared using our tool for comparing test suites which is described in Chapter 4. The objective of this comparison is to identify gaps in the component testing. These gaps indicate weaknesses of component testing (i.e. the fewer the gaps are, the stronger the component testing is). If there are gaps in the component testing, the component must be tested better for the new context.

We determine the adequacy of component testing which is provided with the component as component certification metadata either as developer certification metadata [39] or as third-party certification metadata [40]. The component certification metadata consists of the test cases used to certify (test) the component. This step requires that the test cases of the component (which the developer used to test the component) are available. We compare the test cases generated for CDCT with the test cases that were executed during component testing, to determine the adequacy of the component testing.

This comparison can identify weaknesses in both the component testing and CDCT, which is illustrated in Figure 3. CDCT denotes the test cases that are devised for performing CDCT, by the component user. CT denotes the test cases that were executed to certify the component, by the component developer. CDCT ∩ CT denotes the test cases that are
common to CDCT and CT. CT – CDCT denotes the test cases that are present in CT but not in CDCT. CDCT – CT denotes the test cases that are present in CDCT but not in CT.

The CT is considered to be adequate if all the test cases in CDCT were already covered during component testing, i.e. CDCT – CT = ø. However, there is a weakness in CT if some of the test cases in CDCT were not executed during component testing, i.e., CDCT – CT ≠ ø. The adequacy of CT shows that the component is acceptable for the new context. Conversely, the component may not be acceptable if CT is inadequate. The discussion on the test adequacy of CDCT and CT is summarised in Table 1.

The CDCT is considered to be adequate if it contains all the test cases that were executed during component testing, i.e., CT – CDCT = ø. However, CDCT is potentially inadequate if some of the test cases in CT are not contained in CDCT, i.e., CT – CDCT ≠ ø. This can be for the following reasons:

i) The test cases in CT – CDCT cover component functionality that is not used in the new context.

ii) The test cases in CT – CDCT cover component functionality that is used in the new context. In this case, CDCT can either be adequate or inadequate. This is because CDCT is derived from system-level test cases which use interactions with

![Diagram of testing and component acceptability](image-url)
systems whereas component testing consists of method calls. Therefore, CDCT is expected to be less thorough than CT, and we expect CT – CDCT to be non-empty in most of cases.

iii) A combination of both (i) and (ii).

In situation (i), CDCT is considered to be adequate. In situation (ii) and (iii), CDCT can either be adequate or inadequate (i.e. the second case in Table 1). We review the test cases in CT – CDCT to identify any that address component functionality that is used in the new context. If we find some functionality present in CT – CDCT which is used in the new context but not tested in CDCT, then the CDCT is inadequate.

<table>
<thead>
<tr>
<th>Component testing (CT)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Adequate</td>
<td>CDCT – CT = φ</td>
</tr>
<tr>
<td>Inadequate</td>
<td>CDCT – CT ≠ φ</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Context-dependent component testing (CDCT)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Adequate</td>
<td>CT – CDCT = φ</td>
</tr>
<tr>
<td>Potentially Inadequate</td>
<td>CT – CDCT ≠ φ</td>
</tr>
</tbody>
</table>

Table 1: Adequacy of CDCT and CT

Comparing CT and CDCT (using the tool) requires a criterion for determining the similarity of test cases. In Chapter 4, we will establish a comprehensive criterion for comparing test cases (and test suites) using equivalence classes [41]. These equivalence classes are defined by the tester. Testers can use different techniques to define these equivalence classes, such as category-partitioning [42].

4. Enriching test suite

Fourth, we extend the adequacy of the testing to test the component for the new context (task 4 in Figure 2). This can be done by:

1) Enriching CT: This will often be the easiest way since this testing is specifically targeted at the component. It usually has good controllability and observability of the component. Good controllability allows testers to control the inputs and internal states of a component. Similarly, good observability allows testers to observe behaviour and internal states of a component, which facilitates the definition of test oracles for the new test cases.
2) Enriching CDCT: If the CT is not easy to extend, it may be possible to extend CDCT and use the tool support, provided this has good controllability and observability of the component. Good controllability and observability facilitate testers to specify test inputs and monitor internal states of the system in order to define new test cases.

3) Writing some additional custom test cases to perform additional testing.

We prefer enriching component testing for better controllability and observability than CDCT. This is because CDCT is created from system-level test cases (usage scenarios of the system), whereas the component testing consists of component-level test cases. Therefore, the component testing has i) more control over input values to the component, ii) greater control and visibility of the internal states of the component during the input processing, and iii) better observability of the intermediate and final outputs of the component. Alternatively, we can extend CDCT or write some additional test cases to test the functionality of the component which was not tested well or at all during the component testing.

To extend the adequacy of component testing, we use CDCT – CT. If CDCT – CT is non-empty, this indicates an area of weakness in the developer’s component testing. This means that no component testing was done for the relevant scenarios, so we need to be extra careful and should do extra testing in this area. As the test cases in CDCT are generated from system-level usage scenarios, each of the test cases in CDCT – CT indicates an area of functionality of the component that was not tested in CT. Therefore, we expand these test cases (that are generated for CDCT) to target the functionality of the component that was not tested before. These test cases are added to CT or CDCT to enrich it.

5. Executing enriched test suite

Finally, we execute the enriched test suite and examine the behaviour of the component for the new context (task 5 in Figure 2). We do this by checking the values returned by the methods (of the component) that are invoked in the test cases.

If we chose to enrich CDCT in task 4, we can execute the enriched test suite using the model-driven tool used for generating test cases in task 2.
2.3. An Example Application

2.3.1. Vending machine system

We illustrate the application of the framework using a small Vending Machine System (VMS). This system implements CB software which uses the component Dispenser as shown in Figure 4. The notation \( \circ \) denotes that the object on the left-hand side uses the object on the right-hand side.

![Component diagram of the vending machine system](image)

**Figure 4: Component diagram of the vending machine system**

VMS allows the user to perform the following operations:

1. Insert coins into the machine.
2. Select an item to purchase.
3. Cancel the transaction. The machine returns all the coins inserted but not consumed.
4. Dispense the selected item. In this case, the machine requests the Dispenser to dispense the selected item. If the Dispenser fails to dispense the item, the VMS prints an error message and returns all the coins.

*Dispenser* has the following interface which is used by VMS.

1. `public void setCredit( int nOfCoins )`
   
   This method is used to compute the credit inserted into the vending machine. It takes an integer parameter which is the number of coins inserted. It computes the credit by multiplying the number of coins inserted with the value of coin.

2. `public int dispense( int selection )`
   
   This method takes an integer parameter representing the item which is requested to be dispensed. It performs the following actions:
   
   a) Ensures the user has made a valid selection.
   
   b) Checks for availability of requested item.
c) Checks for sufficient credit for the requested item.
d) Dispenses the item to the user.

If the selection is invalid, it returns -1. If the selection is valid but the selected item is unavailable, it returns -2. If the selection is valid but the credit is insufficient to buy the item, it returns -3. Otherwise, it dispenses the item and returns the number of coins consumed.

2.3.1.1. Implementation of the Vending Machine System

Orso et al. [43] provide an implementation of VMS and Dispenser which is shown in Figures 5 and 6 respectively. We wrote a front-end (VendingMachineGUI) to VMS to make it an executable system which is shown in Figure 7. Method insert increments the coin counter, and method cancel resets the coin counter to zero. Method vend invokes the methods setCredit and dispense of Dispenser. If the selection is valid, the selected item is available, and the credit is sufficient to buy that item, the dispense method returns a value which is greater than 0. In this case, the item is dispensed and the change (if any) is returned to the user with the following message “Take your item”. If the selection is invalid, the selected item is unavailable or the credit is insufficient to buy the item, an error message is displayed and credit is reset to 0. These error messages are: “Invalid selection”, “Item unavailable” or “Insufficient credit”.

We modified the vend and cancel methods of VMS to return the messages to VendingMachineGUI instead of displaying them on the console window using System.out.print statements. Further, we added a method reset to VMS to set it to the initial state. We invoke this method to run each test (in the test suite) independent of other tests.
Figure 8 shows an interaction diagram which illustrates one usage scenario of the VMS. In this figure, rectangles represent components of the VMS. Vertical lines show the lifelines of the components. Solid arrows represent method calls. Dotted arrows show the values returned by the method calls.

We have introduced a defect $D_2$ in Dispenser (in line 26 of Figure 6) which returns the number of coins inserted (i.e. credit / COINVALUE) instead of the number of coins consumed (i.e. COST / COINVALUE). Consequently, VMS computes an incorrect number of coins (in line 28 of Figure 5) which are left and returns the incorrect message (in line 14 of Figure 5) back to the VendingMachineGUI. Eventually, VendingMachineGUI displays an incorrect message to the user on the GUI window. We will use this defect to illustrate an increase in defect-detection capability of the enriched test suite which is obtained as a result of applying the framework.
```java
public class VendingMachine {
    private int coins;
    private Dispenser d;
    public VendingMachine() {
        coins = 0;
        d = new Dispenser();
    }
    public void insert() {
        coins++;
    }
    public String cancel() {
        String message = "";
        if (coins > 0) {
            message = "Take your change: " + coins + " coins";
        }
        coins = 0;
        return message;
    }
    public String vend(int item) {
        String message = "";
        if (coins == 0) {
            message = "Insufficient credit";
        }
        d.setCredit(coins);
        int result = d.dispense(item);
        if (result > 0) { // event OK
            message = "Take your item";
            coins -= result;
        } else switch (result) { // event NOK
            case -1: message = "Invalid selection"; break;
            case -2: message = "Item unavailable"; break;
            case -3: message = "Insufficient credit"; break;
        }
        cancel();
        return message;
    }
    public void reset() {
        coins = 0;
        d = new Dispenser();
    }
} // class VendingMachine

Figure 5: The application: Vending Machine System
```
class Dispenser {
    final private int COINVALUE = 25;
    final private int COST = 50;
    final private int MAXSEL = 4;
    private int credit;
    private int itemsInStock[] = {2, 0, 0, 5, 4};
    public Dispenser() {
        credit = 0;
    }
    public void setCredit(int nOfCoins) {
        System.out.print("setCredit(" + nOfCoins + ") ");
        if (credit != 0)
            System.out.println("Credit already set");
        else
            credit = nOfCoins * COINVALUE;
    }
    public int dispense(int selection) {
        int val = 0;
        if (selection > MAXSEL)
            val = -1; // Invalid selection
        else if (itemsInStock[selection] < 1)
            val = -2; // Selection unavailable
        else if (credit < COST)
            val = -3; // Insufficient credit
        else {
            val = credit / COINVALUE;
            itemsInStock[selection]--;
        }
        credit = 0;
        return val;
    }
} // class Dispenser

Figure 6: The component: Dispenser

Figure 7: Vending Machine GUI (front-end)
2.3.1.2. Test Suite for component testing of Dispenser

To illustrate the application of the framework, we needed a test suite for component testing of Dispenser to compare with CDCT. We devised the test suite using the following testing scenarios:

1. The number of invocations of the `dispense` method
   i) One invocation
   ii) Two consecutive invocations (i.e. `setCredit`, `dispense`, `dispense`)
   iii) Two invocations but not consecutive (i.e. `dispense`, `setCredit`, `dispense`)

2. Coins inserted before the request to dispense an item is made
   i) Zero coins: No coin is inserted before the request to dispense an item is made.
   ii) One coin (insufficient credit): As the cost of an item for this VMS is two coins, one coin is insufficient to buy an item.
   iii) Two coins (sufficient credit): Credit entered is sufficient to buy an item.
We have omitted the scenario in which the credit inserted is more than the cost of item to induce a weakness in the component testing and illustrate the potential benefit of the framework.

3. Validity of the selection of item which is requested to be dispensed
   i) Available: A valid item is selected and it is available. Available items are referred to by the parameter values 0, 3 and 4 of the vend and dispense methods of the VMS.
   ii) Not Available: A valid item is selected but is not available. Unavailable items are referred to by the parameter values 1 and 2.
   iii) Invalid: An invalid item is selected. Any parameter value other than 0, 1, 2, 3, and 4 refers to an invalid selection.

We combine these scenarios to devise test cases. The test suite for component testing of Dispenser (CT) is shown in Table 2. We make this test suite executable by writing a JUnit [25] test driver which is provided in Appendix A.1. JUnit is a unit testing frameworks for testing Java classes.

2.3.2. Applying the framework

2.3.2.1. Modelling usage scenarios of CB software

We model the usage scenarios of the VMS which use the functionality of Dispenser (task 1 in Figure 2). VMS uses the following functionality of Dispenser:

1. Setting the credit: Dispenser sets the credit inserted into the vending machine using the setCredit method.
2. Dispensing the item: Dispenser dispenses the item using the dispense method.

Orso et al. [43] have devised usage scenarios for the VMS (shown in Table 3) in which the vend method invokes the setCredit and dispense methods of Dispenser. They have defined these scenarios as sequences of method calls (SMCs). In these sequences, the calls to the constructor method of VMS are omitted for simplicity.

We create test cases for these scenarios by providing different parameter values to the vend method in the SMC model to cover the testing scenarios in which the user selects an item which is: i) available, ii) unavailable, and iii) invalid. Therefore, for each usage scenario in Table 3, we have three test cases as shown in Table 4.
<table>
<thead>
<tr>
<th>TC</th>
<th>Invocations to dispense method</th>
<th>Coins inserted</th>
<th>Availability of selected Item</th>
<th>Inputs</th>
<th>Expected output(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>One invocation</td>
<td>Zero</td>
<td>Available</td>
<td>setCredit(0) dispense(0)</td>
<td>-3</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(0) dispense(3)</td>
<td>-3</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(0) dispense(4)</td>
<td>-3</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>Unavailable</td>
<td>setCredit(0) dispense(1)</td>
<td>-2</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(0) dispense(5)</td>
<td>-1</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>One</td>
<td>Available</td>
<td>setCredit(1) dispense(0)</td>
<td>-3</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(1) dispense(3)</td>
<td>-3</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(1) dispense(4)</td>
<td>-3</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>Unavailable</td>
<td>setCredit(1) dispense(1)</td>
<td>-3</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(1) dispense(5)</td>
<td>-1</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>Two</td>
<td>Available</td>
<td>setCredit(2) dispense(0)</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(2) dispense(3)</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(2) dispense(4)</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td>Unavailable</td>
<td>setCredit(2) dispense(1)</td>
<td>-2</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(2) dispense(5)</td>
<td>-1</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>One</td>
<td>Available</td>
<td>setCredit(0) dispense(0) dispense(0)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(0) dispense(3) dispense(3)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(0) dispense(4) dispense(4)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td>Unavailable</td>
<td>setCredit(0) dispense(1) dispense(1)</td>
<td>-2 and -2</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(0) dispense(5) dispense(5)</td>
<td>-1 and -1</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>One</td>
<td>Available</td>
<td>setCredit(1) dispense(0) dispense(0)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(1) dispense(3) dispense(3)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(1) dispense(4) dispense(4)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td>Unavailable</td>
<td>setCredit(1) dispense(1) dispense(1)</td>
<td>-2 and -2</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(1) dispense(5) dispense(5)</td>
<td>-1 and -1</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>Two</td>
<td>Available</td>
<td>setCredit(2) dispense(0) dispense(0)</td>
<td>2 and -3</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(2) dispense(3) dispense(3)</td>
<td>2 and -3</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(2) dispense(4) dispense(4)</td>
<td>2 and -3</td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td>Unavailable</td>
<td>setCredit(2) dispense(1) dispense(1)</td>
<td>-2 and -2</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(2) dispense(5) dispense(5)</td>
<td>-1 and -1</td>
</tr>
<tr>
<td>31</td>
<td></td>
<td>One</td>
<td>Available</td>
<td>setCredit(1) dispense(0) setCredit(0) dispense(0)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(0) dispense(3) setCredit(0) dispense(3)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td>33</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(0) dispense(4) setCredit(0) dispense(4)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
<td>Unavailable</td>
<td>setCredit(0) dispense(1) setCredit(0) dispense(1)</td>
<td>-2 and -2</td>
</tr>
<tr>
<td>35</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(0) dispense(5) setCredit(0) dispense(5)</td>
<td>-1 and -1</td>
</tr>
<tr>
<td>36</td>
<td></td>
<td>Two</td>
<td>Available</td>
<td>setCredit(1) dispense(0) setCredit(1) dispense(0)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td>37</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(1) dispense(3) setCredit(1) dispense(3)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td>38</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(1) dispense(4) setCredit(1) dispense(4)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td>39</td>
<td></td>
<td></td>
<td>Unavailable</td>
<td>setCredit(1) dispense(1) setCredit(1) dispense(1)</td>
<td>-2 and -2</td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(1) dispense(5) setCredit(1) dispense(5)</td>
<td>-1 and -1</td>
</tr>
<tr>
<td>41</td>
<td></td>
<td>Two</td>
<td>Available</td>
<td>setCredit(2) dispense(0) setCredit(2) dispense(0)</td>
<td>2 and 2</td>
</tr>
<tr>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(2) dispense(3) setCredit(2) dispense(3)</td>
<td>2 and 2</td>
</tr>
<tr>
<td>43</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(2) dispense(4) setCredit(2) dispense(4)</td>
<td>2 and 2</td>
</tr>
<tr>
<td>44</td>
<td></td>
<td></td>
<td>Unavailable</td>
<td>setCredit(2) dispense(1) setCredit(2) dispense(1)</td>
<td>-2 and -2</td>
</tr>
<tr>
<td>45</td>
<td></td>
<td></td>
<td></td>
<td>setCredit(2) dispense(5) setCredit(2) dispense(5)</td>
<td>-1 and -1</td>
</tr>
</tbody>
</table>

Table 2: Test suite for component testing of Dispenser (CT)
<table>
<thead>
<tr>
<th>Usage Scenario</th>
<th>Test Case</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1i</td>
<td>vend(3)</td>
<td>“Insufficient credit”</td>
</tr>
<tr>
<td></td>
<td>1ii</td>
<td>vend(2)</td>
<td>“Insufficient credit”</td>
</tr>
<tr>
<td></td>
<td>1iii</td>
<td>vend(35)</td>
<td>“Insufficient credit”</td>
</tr>
<tr>
<td>2</td>
<td>2i</td>
<td>insert vend(3)</td>
<td>“Insufficient credit”</td>
</tr>
<tr>
<td></td>
<td>2ii</td>
<td>insert vend(2)</td>
<td>“Insufficient credit”</td>
</tr>
<tr>
<td></td>
<td>2iii</td>
<td>insert vend(35)</td>
<td>“Insufficient credit”</td>
</tr>
<tr>
<td>3</td>
<td>3i</td>
<td>insert insert vend(3)</td>
<td>“Take your item”</td>
</tr>
<tr>
<td></td>
<td>3ii</td>
<td>insert insert vend(2)</td>
<td>“Item unavailable”</td>
</tr>
<tr>
<td></td>
<td>3iii</td>
<td>insert insert vend(35)</td>
<td>“Invalid selection”</td>
</tr>
<tr>
<td>4</td>
<td>4i</td>
<td>insert insert insert vend(3)</td>
<td>“Take your item”</td>
</tr>
<tr>
<td></td>
<td>4ii</td>
<td>insert insert insert vend(2)</td>
<td>“Take your item”</td>
</tr>
<tr>
<td></td>
<td>4iii</td>
<td>insert insert insert vend(35)</td>
<td>“Invalid selection”</td>
</tr>
<tr>
<td>5</td>
<td>5i</td>
<td>insert insert cancel vend(3)</td>
<td>“Take your item”</td>
</tr>
<tr>
<td></td>
<td>5ii</td>
<td>insert insert cancel vend(2)</td>
<td>“Item unavailable”</td>
</tr>
<tr>
<td></td>
<td>5iii</td>
<td>insert insert cancel vend(35)</td>
<td>“Invalid selection”</td>
</tr>
<tr>
<td>6</td>
<td>6i</td>
<td>insert cancel insert vend(3)</td>
<td>“Take your item”</td>
</tr>
<tr>
<td></td>
<td>6ii</td>
<td>insert cancel insert vend(2)</td>
<td>“Item unavailable”</td>
</tr>
<tr>
<td></td>
<td>6iii</td>
<td>insert cancel insert vend(35)</td>
<td>“Invalid selection”</td>
</tr>
<tr>
<td>7</td>
<td>7i</td>
<td>insert cancel insert insert vend(3)</td>
<td>“Take your item”</td>
</tr>
<tr>
<td></td>
<td>7ii</td>
<td>insert cancel insert insert vend(2)</td>
<td>“Item unavailable”</td>
</tr>
<tr>
<td></td>
<td>7iii</td>
<td>insert cancel insert insert vend(35)</td>
<td>“Invalid selection”</td>
</tr>
<tr>
<td>8</td>
<td>8i</td>
<td>insert insert cancel insert vend(3)</td>
<td>“Take your item”</td>
</tr>
<tr>
<td></td>
<td>8ii</td>
<td>insert insert cancel insert vend(2)</td>
<td>“Item unavailable”</td>
</tr>
<tr>
<td></td>
<td>8iii</td>
<td>insert insert cancel insert vend(35)</td>
<td>“Invalid selection”</td>
</tr>
<tr>
<td>9</td>
<td>9i</td>
<td>insert insert vend(3) insert insert vend(3)</td>
<td>“Take your item” and “Take your item”</td>
</tr>
<tr>
<td></td>
<td>9ii</td>
<td>insert insert vend(2) insert insert vend(2)</td>
<td>“Item unavailable” and “Item unavailable”</td>
</tr>
<tr>
<td></td>
<td>9iii</td>
<td>insert insert vend(35) insert insert vend(35)</td>
<td>“Invalid selection” and “Invalid selection”</td>
</tr>
<tr>
<td>10</td>
<td>10i</td>
<td>insert insert insert insert vend(3) vend(3)</td>
<td>“Take your item” and “Insufficient credit”</td>
</tr>
<tr>
<td></td>
<td>10ii</td>
<td>insert insert insert insert vend(2) vend(2)</td>
<td>“Item unavailable” and “Item unavailable”</td>
</tr>
<tr>
<td></td>
<td>10iii</td>
<td>insert insert insert insert vend(35) vend(35)</td>
<td>“Invalid selection” and “Insufficient credit”</td>
</tr>
<tr>
<td>11</td>
<td>11i</td>
<td>insert insert vend(3) vend(3)</td>
<td>“Take your item” and “Insufficient credit”</td>
</tr>
<tr>
<td></td>
<td>11ii</td>
<td>insert insert vend(2) vend(2)</td>
<td>“Item unavailable” and “Item unavailable”</td>
</tr>
<tr>
<td></td>
<td>11iii</td>
<td>insert insert vend(35) vend(35)</td>
<td>“Invalid selection” and “Invalid selection”</td>
</tr>
<tr>
<td>12</td>
<td>12i</td>
<td>insert vend(3) insert vend(3)</td>
<td>“Insufficient credit” and “Insufficient credit”</td>
</tr>
<tr>
<td></td>
<td>12ii</td>
<td>insert vend(2) insert vend(2)</td>
<td>“Item unavailable” and “Item unavailable”</td>
</tr>
<tr>
<td></td>
<td>12iii</td>
<td>insert vend(35) insert vend(35)</td>
<td>“Invalid selection” and “Invalid selection”</td>
</tr>
</tbody>
</table>

Table 3: Usage scenarios of the Vending Machine System which use Dispenser

Table 4 Test Cases of CDCT of Dispenser
2.3.2.2. Deriving component test cases for CDCT

We derive component test cases from the usage scenarios using our model-driven tool (task 2 in Figure 2). We model these usage scenarios using interaction diagrams \((\text{SMC-Model}^{\text{VMS}})\).

We then generate a concrete and executable test suite \((\text{CDCT})\) from \(\text{SMC-Model}^{\text{VMS}}\) using our model-driven tool. The tool requires \(\text{SMC-Model}^{\text{VMS}}\) and test data (inputs), produces \(\text{xUnit-Model}^{\text{VMS}}\) (an intermediate output), and generates concrete and executable test cases (the final output) which are provided in Appendices A.2, A.3, A.4 and A.5 respectively. The details of how the tool works are discussed in Chapter 3.

To extract test cases from \(\text{CDCT}\), which is concrete and executable, we i) instrument it to log the method calls along with parameters and the value returned by \(\text{Dispenser}\) and ii) capture execution traces which are provided in Appendix A.6.

The \(\text{vend}\) method in CDCT invokes i) the \(\text{setCredit}\) method with an integer parameter which represents the number of coins inserted before invoking the \(\text{vend}\) method, and ii) the \(\text{dispense}\) method with the integer parameter which was passed to the \(\text{vend}\) method. For example, the test case 4i (in Table 4) produces the following trace:

\[
\text{setCredit}(3) \text{ dispense}(3) \text{ output} = \text{"Take your item"}
\]

From these execution traces we extract the test cases, which are shown in Table 5. The extraction of test cases from the execution traces is trivial as there is a one-to-one mapping between traces and test cases.

2.3.2.3. Comparing test suites

In this step, we evaluate the test adequacy of the component testing of \(\text{Dispenser}\). We compare \(\text{CT}\) and \(\text{CDCT}\) to identify gaps in the component testing of \(\text{Dispenser}\) (task 3 in Figure 2). These gaps indicate weaknesses in the component testing. We compare \(\text{CT}\) and \(\text{CDCT}\) using our tool for comparing test suites which is described in Chapter 4.

As noted before, the comparison requires a criterion for determining the similarity of test cases. In this case, we define the following criterion for the similarity of test cases: two test cases are considered the same if and only if they contain the same method calls in the same order, and the parameter values for those methods belong to the same equivalence class. For comparing the parameter values of the methods \(\text{dispense}\) and \(\text{setCredit}\), we group them into the following equivalence classes (EC):
Table 5: Test cases extracted from CDCT of Dispenser

i. Equivalence classes for the dispense method:
   1. EC-1: The item which is selected by the user is available.
   2. EC-2: The item which is selected by the user is unavailable.
   3. EC-3: The user has made an invalid selection.

Using these equivalence classes, the following pairs of the dispense method (along with parameter values) are considered equivalent:

1. dispense(0) and dispense(3): parameters belong to EC-1
2. dispense(0) and dispense(4): parameters belong to EC-1
3. dispense(3) and dispense(4): parameter values belong to EC-1
4. dispense(1) and dispense(2): parameter values belong to EC-2
5. dispense(x) and dispense(y) where x and y are ≥ 4 or < 0: parameter values belong to EC-3
ii. Equivalence classes for the `setCredit` method:
   1. EC-1: The credit inserted is insufficient to buy an item.
   2. EC-2: The credit inserted is sufficient to buy an item.

Using these equivalence classes, the following pairs of the `setCredit` method (along with parameter values) are considered equivalent:

1. `setCredit(0)` and `setCredit(1)`: parameters belong to EC-4
2. `setCredit(x)` where \( x \geq 2 \): parameters belong to EC-5

The results of the comparison of `CT` and `CDCT` are shown in Table 6, in which `CDCT - CT` shows gaps in `CT` and `CT - CDCT` shows the gaps in `CDCT`. We are interested in the gaps in `CT` i.e. gaps in component testing of `Dispenser` as it highlights an area of `Dispenser` which was not tested during component testing. The review of the gap (`CDCT - CT`) shows that `Dispenser` is not tested when the credit inserted is more than the cost of the item. As `Dispenser` is a small component providing little functionality to VMS, the gap identified in component testing is small. However, for the components providing substantial functionality, the gaps in component testing may be significant.

The framework is generic and the comparison of test suites is one step out of many. We use equivalence classes for comparing test suites. However, testers can select any criterion for comparing test suites. If equivalence classes are used, then the tester needs to define these equivalence classes for the particular component.

2.3.2.4. Enriching CT

We generate test cases to extend the testing to cover the weaknesses identified by the gaps. We enrich `CT` by adding these test cases to extend its test adequacy (task 4 in Figure 2).

We review the test cases in `CDCT - CT`. This review shows that `Dispenser` is not tested when the credit inserted into the vending machine is more than the item’s cost. Hence, we enrich `CT` by adding test cases in which the inserted coins are more than the item’s cost (i.e. more than 2 coins are inserted). We denote these test cases by `∆CT`, shown in Table 7. These test cases are added to `CT` to enrich it (`CT′`).
### Table 6: Gaps in CT and CDCT

<table>
<thead>
<tr>
<th>Gaps</th>
<th>Inputs</th>
<th>Output(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT-CDCT</td>
<td>setCredit(0) dispense(0) dispense(0)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td></td>
<td>setCredit(0) dispense(3) dispense(3)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td></td>
<td>setCredit(0) dispense(4) dispense(4)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td></td>
<td>setCredit(0) dispense(1) dispense(1)</td>
<td>-2 and -2</td>
</tr>
<tr>
<td></td>
<td>setCredit(0) dispense(5) dispense(5)</td>
<td>-1 and -1</td>
</tr>
<tr>
<td></td>
<td>setCredit(1) dispense(0) dispense(0)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td></td>
<td>setCredit(1) dispense(3) dispense(3)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td></td>
<td>setCredit(1) dispense(4) dispense(4)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td></td>
<td>setCredit(1) dispense(1) dispense(1)</td>
<td>-2 and -2</td>
</tr>
<tr>
<td></td>
<td>setCredit(1) dispense(5) dispense(5)</td>
<td>-1 and -1</td>
</tr>
<tr>
<td></td>
<td>setCredit(2) dispense(0) dispense(0)</td>
<td>2 and -3</td>
</tr>
<tr>
<td></td>
<td>setCredit(2) dispense(3) dispense(3)</td>
<td>2 and -3</td>
</tr>
<tr>
<td></td>
<td>setCredit(2) dispense(4) dispense(4)</td>
<td>2 and -3</td>
</tr>
<tr>
<td></td>
<td>setCredit(2) dispense(1) dispense(1)</td>
<td>-2 and -2</td>
</tr>
<tr>
<td></td>
<td>setCredit(2) dispense(5) dispense(5)</td>
<td>-1 and -1</td>
</tr>
<tr>
<td></td>
<td>setCredit(0) dispense(0) setCredit(0) dispense(0)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td></td>
<td>setCredit(0) dispense(3) setCredit(0) dispense(3)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td></td>
<td>setCredit(0) dispense(4) setCredit(0) dispense(4)</td>
<td>-3 and -3</td>
</tr>
<tr>
<td></td>
<td>setCredit(0) dispense(1) setCredit(0) dispense(1)</td>
<td>-2 and -2</td>
</tr>
<tr>
<td></td>
<td>setCredit(0) dispense(5) setCredit(0) dispense(5)</td>
<td>-1 and -1</td>
</tr>
<tr>
<td>CDCT-CT</td>
<td>setCredit(3) dispense(3)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>setCredit(3) dispense(2)</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td>setCredit(3) dispense(35)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>setCredit(4) dispense(3) setCredit(0) dispense(3)</td>
<td>2 and -3</td>
</tr>
<tr>
<td></td>
<td>setCredit(4) dispense(2) setCredit(0) dispense(2)</td>
<td>-2 and -2</td>
</tr>
<tr>
<td></td>
<td>setCredit(4) dispense(35) setCredit(0) dispense(35)</td>
<td>-1 and -1</td>
</tr>
</tbody>
</table>

### Table 7: Test cases for the gaps in the component testing of Dispenser (ΔCT)

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Inputs</th>
<th>Expected Outputs(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>setCredit(3) dispense(3)</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>setCredit(3) dispense(2)</td>
<td>-2</td>
</tr>
<tr>
<td>3</td>
<td>setCredit(3) dispense(35)</td>
<td>-1</td>
</tr>
<tr>
<td>4</td>
<td>setCredit(4) dispense(3) setCredit(0) dispense(3)</td>
<td>2 and -3</td>
</tr>
<tr>
<td>5</td>
<td>setCredit(4) dispense(2) setCredit(0) dispense(2)</td>
<td>-2 and -2</td>
</tr>
<tr>
<td>6</td>
<td>setCredit(4) dispense(35) setCredit(0) dispense(35)</td>
<td>-1 and -1</td>
</tr>
<tr>
<td>7</td>
<td>setCredit(5) dispense(3)</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>setCredit(5) dispense(2)</td>
<td>-2</td>
</tr>
<tr>
<td>9</td>
<td>setCredit(5) dispense(35)</td>
<td>-1</td>
</tr>
<tr>
<td>10</td>
<td>setCredit(5) dispense(3) setCredit(0) dispense(3)</td>
<td>2 and -3</td>
</tr>
<tr>
<td>11</td>
<td>setCredit(5) dispense(2) setCredit(0) dispense(2)</td>
<td>-2 and -2</td>
</tr>
<tr>
<td>12</td>
<td>setCredit(5) dispense(35) setCredit(0) dispense(35)</td>
<td>-1 and -1</td>
</tr>
</tbody>
</table>
2.3.2.5. Execute enriched test suite

Finally, we execute $CT'$ to test Dispenser for the new context (task 5 in Figure 2). $CT'$ is provided in Appendix A.7.

Test cases 1, 4, 7 and 10 failed due to the defect $D_1$. These test cases returned 3, 5, 4 and 5 respectively instead of returning 2. $CT'$ has detected $D_1$ whereas $CT$ did not detect $D_1$ because this defect only appears when the credit inserted is more than the cost of an available item. This defect remains uncovered even when the coins inserted are equal to the cost of the item (i.e. coins = COST). This toy example illustrates an extension in defect-detection capability of $CT'$ because of $\Delta CT$ which is achieved by applying $MD$-$CDCT$.

2.4. Related Work

Software testing is an important part of the software development process, which assures the quality of software products. Software applications are tested to ensure their correct working. Myers defines software testing as the “process of trying to discover every conceivable fault or weakness in a work product” [41]. With the emergence of CSD, the need for Component-Based Testing (CBT) emerged. CBT is an important activity in CSD for developing reliable CB software. A major problem in CBT is the lack of adequate information about the component, which makes it a challenging task (Section 1.2) [44]. Different techniques that have been proposed for addressing these problems are discussed below.

2.4.1. Built-in testing

Built-in testing (BIT) is a technique in which built-in tests are added in the component’s code, to add support for testing the component [44]. BIT refers to all mechanisms that add information to a component’s code for facilitating testing or checking assertions at runtime [17].

Yingxu et al. [45] propose a BIT approach for developing maintainable CB software in which built-in tests are added to the component’s code such that the component user can decide whether to execute these tests or not. The component user can run the component in “test (maintenance) mode” or “normal mode”. In test mode, the built-in tests are executed during execution of the component whereas in normal mode, these tests are not executed.

Yingxu’s approach increases the component size due to the added tests. To address this problem, Hornstein and Elder [46] propose the Component+ BIT method which separates
test cases from the component. The component provider produces a BIT-component and a test-component. The BIT-component is a component that has built-in testing capabilities. The test-component contains test cases and interacts with the built-in testing capabilities of the BIT-component through its interfaces.

Beydeda and Gruhn [47] propose a self-testing strategy for COTS components (STECC). They suggest augmenting the test component with analysis functionality and testing tools. By doing this, the information that the component user needs to generate test cases can either be encapsulated in the component, or it can be generated on demand.

Edwards [48] suggests supporting the flow of information from the component developer to the component user using wrappers. In this approach, the component developer adds the information that can help in CBT to the component and provides some wrappers that can interact with the component. The component user can use these wrappers to extract information from the component. The component user can add or remove wrappers from the component without having access to the source code.

BIT approaches increase testability of components by adding built-in tests to the component. However, they have the following drawbacks:

i) They increase the size of the component.
ii) Test cases are developer-oriented and the component user cannot influence the generation of test cases.

2.4.2. Component metadata

In the component metadata approach, the component developer equips the component with some information (component metadata) that the component user can use for performing CBT. Component metadata can be either metadata or metamethods [43]. The metadata are information about the component and metamethods are the methods that retrieve or calculate information about the component.

Orso et al. [49] propose that all the software engineering artifacts which are used for the development of a component should be shipped along with the component as component metadata. The component provider can provide control-flow and data-flow graphs of the component that increase understandability and testability of the component. These graphs are also helpful in performing coverage analysis during CBT.
Wu et al. [50] propose to deliver a UML model of the component as metadata. The UML model can be used to determine context-dependent relationships among the components, which can be helpful for CBT. Belli and Budnik [51] propose a similar approach in which they augment the component with UML statecharts. Test cases are generated from the UML statecharts, using model-based tools. Using these techniques, the component user can perform coverage-based execution of the model, to achieve greater reliability of the component. However, the component developer has to update the model each time the component is modified.

Liu et al. [52] introduce the concept of retro-components. A retro-component has a retrospector in it, which maintains testing and dynamic execution history. It records the tests that are conducted by the component developer and makes this testing information available to the component user. Retrospectors enhance the component so that the user can query the information provided and collect relevant information during their own testing activities.

In common with BIT, metadata approaches enhance the testability of components. They have the following advantages over BIT:

i) Test cases are not stored in the component’s code and thus they do not increase the size of the component.

ii) The component user can use the information, delivered as metadata, to generate dynamic test cases.

iii) Metadata may support the generation of test data.

These approaches make a compromise on implementation transparency of the component. Implementation transparency is the hiding of implementation details from the component user. It is one of the quality objectives of component-based development [53]. However, it affects testability of components by making it difficult to apply directly traditional white-box techniques [50], fully exercise the software (lack of controllability), and know the result of execution (lack of observability) [31].

2.4.3 Component certification

A component user is always concerned about the quality and reliability of a component. To increase the component user’s trust, components can be certified before their reuse in CB software [54, 55]. The following techniques have been proposed to certify components:

1. Third-party certification
2. Developer certification
3. User certification

Counsell [40] suggests that a component should be certified by a third-party. In third-party certification, an independent organisation tests the quality of the component and provides the test results, along with the test environment, to the component user. Ma et al. [56] propose a framework for third-party certification that consists of following three steps:

i) The third-party provides guidelines to the component developer.

ii) The component provider generates a test package using these guidelines.

iii) The third-party executes the test package and produces a test report.

An evaluation of this framework revealed some errors in a component, which demonstrates its usefulness. An advantage of third-party certification is that it is conducted by a neutral organisation, and hence the results are not biased.

The certification of a component through a third-party may be costly and small organisations may not be able to afford it. Morris et al. [39, 57] propose that the component developer should perform component certification in order to avoid the cost associated with third-party certification. In this approach, the component developer attaches test cases along with their results (as a proof of their execution) to the component. An advantage of this approach is that the component user can determine, by examining the test cases, how thoroughly (adequately) the component developer has tested the component.

The test cases executed during developer certification are context-independent and test results may be biased as these test cases are created by the component developer. To address this issue, Voas [58] proposes that the component user should certify the component using black-box testing, in which test cases are generated from the interface specifications of the component. The component user may use fault-injection techniques in which faults are generated instead of testing the component with the correct inputs, to determine the reliability of the component. Alvaro et al. [59] propose a Component Quality Model (CQM) to certify component. The component user (evaluation team) uses CQM to specify evaluation goals during component certification. Later, they propose a framework [60] for evaluating the quality of software components.

An advantage of user certification is that the component user defines test requirements and thus the component is certified using context-dependent test cases. A disadvantage of this approach is that it does not address the problem of adequacy of component testing (Section
1.2). Our framework is capable of determining and extending the adequacy of testing done at the time of component development and component reuse.

### 2.4.4. Component contracts

Zheng and Budell [61, 62] propose a contract-based testing technique, Test by Contract (TbC), for components. They extend the “Design by Contract” concept to leverage model-based testing to design test contracts for component testing. They introduce a novel concept of “Contract for Testability”, and devise some contract-oriented concepts such as internal and external test contracts. Later, they use TbC to propose a novel framework, Model-Based Software Component Testing (MBSCT) [63].

Briand et al. [16] propose a framework for component testing using Constraints on Succeeding and Preceding Events (CSPE) testing technique [56]. They distribute the roles and responsibilities between the component developer and the component user. The developer generates CSPE constraints for the component interface methods, and implements CSPE probes (which are built-in methods). These CPSE probes are used to increase observability and controllability during component testing (e.g. controlling and observing the internal states of the component). The user identifies the component functionalities that are used in the component-based software being developed, selects a CSPE-based testing criterion, generates test cases and executes them. As the CPSE-based approach does not require source code, it can be used for the testing of COTS components. A disadvantage of this approach is that it requires the component to provide that metadata (CPSE constraints) to derive test sequences.

Jiang et al. [32] propose a contract-based mutation to address the test adequacy criterion of components. They apply mutation testing to the contracts provided with components which can identify possible misunderstandings of requirements or implementation errors regarding contracts. They define a language to describe the interface contracts by extending the definition of Enterprise Java Bean components. They design the following mutation operators for the contract of the component: contract negation, condition exchange, precondition weakening, post condition strengthening, and contract stuck-at faults.

Delamaro et al. [34] present a mutation-based criterion, named interface mutation (IM), for integration testing of components. They devise different types of mutation operators to generate interface mutants. They apply their criterion to SPACE, a program developed for the European Space Agency. The experiments show that the fault-revealing capability of
interface mutation is greater than selecting mutants at random. Later, they propose a technique to determine the test adequacy of components using IM [64].

Gosh and Mathur [65] propose criteria to determine the adequacy of test cases developed to test components and systems using the interface mutants. The description of the component’s interface contains information about the signatures of its methods and the exceptions that these methods can raise. This information is not enough to establish an adequacy criterion. Therefore, they use the description of a component’s interface to generate interface mutants. The quality of tests is judged based on their tendency to identify interface mutants. An advantage of this approach is that the effort required to develop the tests is lower than the effort required for developing test sets that are adequate with respect to traditional code coverage criteria.

Hashim et al. [66] propose to inject interface faults to evaluate the quality of the test cases. They write wrappers around interface services of a component to perform operations such as disabling the implementation of the interface services, raising exceptions or corrupting the inputs and outputs of interface services. These wrappers not only detect errors related to component interactions, but can also handle exceptions which are raised when interface faults occur.

An advantage of contract-based techniques is that they do not require source code (as they are based on contracts), which makes them applicable to COTS component. A disadvantage of these approaches is that they only detect the contract-related defects, i.e. they are not good at detecting the defects that are related to the use of a component in a novel context.

2.4.5. Testable architecture

In the testable architecture approach, the component developer equips the component with an architecture that allows the component user to execute test cases.

Gao et al. [67] introduce testable beans to increase testability of a component. In this approach, the component developer implements an interface for testing (test interface) and codes test cases in the form of clients. The testable beans are components that are:

i) Deployable and executable.

ii) Traceable to allow the user to monitor and track their behaviour.

iii) Testable by implementing test interfaces to access their self-test capabilities.
iv) Usable with testing tools, i.e., they can interact with these tools.
An advantage of this approach is that the test cases are stored in clients and they are not part of the testable bean. However, the component developer has to do much work to maintain the testable beans.

Jabeen and Rehman [68] propose a framework for testing object-oriented components, in which the component developer, the component user and a third-party communicate test information using descriptors. These descriptors contain requirements of the component. The component developer prepares a component descriptor and attaches it to the component. The component user specifies the component’s requirements in another descriptor, the component requirement descriptor. The third-party generates test information using the information in the component descriptor and the component requirement descriptor. Advantages of this approach are:

i) It allows everyone (the component developer, the component user and the third-party) to participate in CBT.

ii) It ensures that the component developer has provided the functionality that is required by the component user.

Qiming et al. [69] emphasise the components of different platforms require their proprietary testing languages to execute test cases. They propose a testing framework based on XML-API to increase the testability of components in CB software. They developed a Component Extension Test Interface (CETI) which uses XML as test specification language. CETI provides the following extension points:

i) An XML-based API extension to test components using component interface mutation.

ii) An XACML-based API extension to test the correctness of component access control interface using mutation testing.

CETI is evaluated on different platforms (such as Linux) and component testing environments (such as CORBA) to show its applicability.

Testable architecture approaches do not increase the size of the component because the additional information is provided in the form of components’ specifications and not within the component. However, similar to the metadata approach, they affect the implementation transparency of the component.
2.4.6. Other approaches

Hill and Gokhale [70] describe a model-driven method for testing of CB software which consists of defining i) a domain-specific modelling language to express business logic of the component, and ii) a programming framework that synthesises configuration files for system simulation. They develop a domain-specific modelling language, Component Behavioural Modelling Language (CBML). CBML can capture high-level component behaviour, and can generate configuration and source files for system simulation using model interpreters.

Buy et al. [71] propose a framework for generating message sequences for component testing by considering the component’s behaviour in isolation and while integrated with other components. For testing the component behaviour in isolation, first data-flow analysis is applied to the methods of the component, second symbolic execution is applied to the methods to derive a formal specification for each method, and finally automated deduction is used to derive sequences of method invocations from the information produced in the previous two steps. For testing the component behaviour when integrated with other components, first the order of integration of components is identified, and then the pair-wise integration of components is performed according to the integration order specified.

Qian [72] proposes a method for testing web applications. He suggests a web application is a composition of interacting components. In this approach, a Component Interaction Diagram (CID) is created from the specification of the web application. From the CID, Component Test Sequences (CTSs) are generated which are in essence the interacting sequences of the components. The CID and CTS are described in XML format. CTSs along with test data form concrete test cases which make the testing of web application the testing of CB software. Finally, he evaluates the coverage of the generated test cases to demonstrate their usefulness.

Ju and Che [73] propose some measures to evaluate the CBT. They believe the validity of component software testing can improve the process of CBT and improve the quality of CB software. They propose the following measure to evaluate the testing process of CB software: i) time and budget cost of CBT, ii) defects found in CB software, and iii) validity of unit and integration testing of the component.
2.4.7. IEC Standards

The International Electrotechnical Commission (IEC) is a worldwide organization that publishes international standards in the electrical and electronic fields [74]. IEC issued a series of standards including IEC-62814, IEC-61508 and IEC-62628 which provide guidance and processes for developing reliable components and CB software.

2.4.7.1. IEC-62814

IEC-62814 [75] deals with the dependability of software containing reusable components. Fevzi [76], who chaired the realisation of this standard, defines: i) characteristics of reusable components, ii) a process for developing components, and iii) a process for developing CB software as follows.

1. Characteristics of reusable components: according to IEC-62814, a reusable component should have the following characteristics [76, pp. 147]:
   – A component should be composed of functionally independent modules.
   – A component should have a wide range of functionality to be used in a variety of contexts.
   – Component specifications should have criteria to define test cases and test oracles.
   – Component source code and documentation should be provided to facilitate its modification.
   – A component should be developed using the state-of-art techniques regarding interfaces, protocols, etc.
   – A component should have greater adaptability to be used with other systems.
   – A component should be capable of working on different platforms, i.e. independent of programming languages, operating systems, hardware, etc.

2. Process for developing components: Only a high-quality component can produce a quality CB software. Fevzi suggests that extra effort should be made to develop high-quality reusable components using the following guidelines [76, pp. 148].
   – The component developer should define the objectives of the component’s reusability.
   – The component developer should explain when and how the component can be reused.
The component developer should decide whether to build the component for single use or for reuse. They should check the library and market to find whether such a component already exists.

The component developer should ensure that the component meets the characteristics of a reusable component before storing it in a library / repository.

3. Process for developing CB software

CSD differs from normal software development as it involves selection, evaluation, modification, re-testing, and deployment of a reusable component. IEC-62814 enforces the following process for developing CB software [76, pp. 149]:

- First, the component user should check for the availability of the reusable component. If a reusable component exists, it should be evaluated.
- Second, if the component requires modifications, the side effects of these modifications should be considered.
- Finally, the requirements of CB software should be validated in the new context.

IEC-62814 assists in developing quality components, and the application of this standard increases the reliability of CB software. However, the component still needs to be tested for the new context as the use of the component may be novel and not imagined by the component developer. IEC-62814 briefly mentions testing the requirements of CB software in the new context. However, it does not provide any guidelines or techniques for identifying any weaknesses of component testing for a particular context of reuse in contrast with MD-CDCT. MD-CDCT makes use of the context of reuse, and evaluates the component testing. If some weaknesses are identified in the component testing, it provides tool support to enrich the component testing.

2.4.7.2. IEC-61508

IEC-61508 [77] is a standard for functional safety of electrical / electronic systems which contain programmable software. It defines Safety Integrity Levels (SILs) which set the requirements on the process for how the software is developed and tested. SILs have two attributes which are: i) the probability of failure, and ii) the frequency of failure. Software should be developed according to the graded requirements (high SIL means high requirements on the software process).
MD-CDCT can use SILs defined in IEC-61508 while enriching the test suites. These SILs can determine the extent to which the weakly tested parts of a component should be tested more thoroughly. The higher SIL means that any gaps identified by MD-CDCT should be expanded and tested more intensively.

2.4.7.3. IEC-62628

IEC-62628 [78] provides guidance on software aspects of dependability. It provides a generic framework for software dependability requirements. It provides some approaches for evaluating and measuring the dependability of CB software. MD-CDCT differs from these approaches in that it finds the weaknesses of component testing regarding the context of reuse, and then extensively tests the component’s functionality which is reused in the CB software.

2.4.8. IEEE Standards

The IEEE Standards Association (IEEE-SA) is an organization within IEEE that develops standards for various industries including information technology [79]. IEEE-SA has proposed different standards to increase component dependability in CB software.


IEEE Standard 982.1-1988 (IEEE-982.1) defines a range of measures for producing reliable software (components) and accessing their dependability [80]. We categorise the measures, defined in IEEE-982.1, into: analysing requirements, complexity, testing and faults / defects found in the software as follow [80]:

1. Requirements-based Measures
   - Requirement Tractability: Identifies the missing requirements.
   - Conflicting Requirements: Determines reliability resulting from a software architecture based on the complexity of an entity-relationship model.
   - Cause and Effect Graphing: Explores inputs and expected outputs of software and identifies ambiguous and incomplete requirements.

2. Complexity-based Measures
   - Architectural Complexity: Determines the complexity of software architecture as represented by entry and exit points defined in design.
– Data Flow Complexity: Measures software reliability based on information flow structure and complexity of interactions between modules.
– Cyclamate complexity: Determines the structural complexity of a coded module.

3. Testing-based Measures
– Minimal Unit Test Case Determination: Determines independent paths through a module to create a minimum number of unit test cases for coverage.
– Test Coverage: Quantifies software test coverage.
– Test Accuracy: Determines the accuracy of the testing in detecting faults.
– Testing Sufficiency: Determines the sufficiency of software testing by comparing actual to predicted faults.

4. Faults and Defects-based Measures
– Fault Density: Establishes standard fault densities, and predicts remaining faults by comparing with the expected fault density.
– Defect Density: Indicates the reliability of a component.
– Cumulative Failure Profile: Predicts reliability through the use of failure profiles.
– Estimated Number of Faults Remaining: Determines the estimated number of remaining faults by defect seeding which indicates the reliability of software.
– Fault-Days Number: Represents the number of days faults remains in the software from their creation to removal.

We can use the testing-based measures for evaluating the effectiveness of MD-CDCT. The test coverage measure can be used to measure the coverage of the enriched test suite. The test accuracy measure and the estimated number of faults remaining measure can determine the defect-detection capability of the enriched test suites which are produced by MD-CDCT. In general, IEEE-982.1 assists in determining the reliability of a component at the time of component development. However, it does not take advantage of the context of reuse to address the adequacy of component testing. Further, these measures lack implementation in contrast with MD-CDCDT which provides tool support to assess and augment the reliability of a component in CB software.

2.4.8.2. IEEE Standard 1517-2010

One of the problems with CBD is the absence of any standard process for component reuse in the software life-cycle process. IEEE Standard 1517-2010 (IEEE-1517) provides a process for component’s reuse while developing CB software [81]. This standard provides additional
activities which are applied during each phase of the software life-cycle to produce a CB software from reusable components. IEEE-1517 proposes the following processes for software development and software support [81, pp. 22-29]:

1. Domain Engineering Process
   - The developer should select the domain architecture by consulting domain experts, developers, and users of the CB software.

2. Implementation Process
   - The developer should define a software life-cycle model which satisfies reuse requirements of components.
   - The developer should use standards, tools, and programming languages that facilitate the practice of reuse of components.

3. Requirements Analysis Process
   - The developer should include software reusability requirements in the quality specifications of the components.

4. Architectural Design Process
   - The developer should use a software architecture that is based on the selected domain architecture.
   - The developer should use a software architecture which can describe the structures of the CB software and the components being reused.
   - External interfaces and internal interfaces (interfaces between components) of the CB software should comply with the domain architecture interfaces.

5. Detailed Design Process
   - The developer should use the language and concepts from the selected domain model.
   - The developer should use data structures and naming conventions from the selected domain model.
   - The developer should develop external and internal interfaces of the CB software that are compliant with the selected domain interface standards.
   - The developer should evaluate and document the software detailed design and test requirements.

7. Documentation Management Process: The developer should reuse existing documentation and test data.

8. Configuration and Asset Reuse Management Process: The developer should document, archive and store reusable components in the repository which component users could access through an asset storage and retrieval mechanism.

IEEE-1517 specifies process requirements for a component’s reuse and describes the relationship of reuse processes to software life cycles. It describes a high-level framework for reuse activities but does not provide details on how to perform the activities. Similarly, it specifies the reuse activities at an abstract level but does not prescribe any specific life-cycle model or methodology. In contrast, MD-CDCT provides a solution for evaluating and extending the reliability of the reusable component. Moreover, IEEE-1517 specifies provisions for acquiring reusable components but does not have provisions for using COTS components whose source code is not supplied, whereas MD-CDCT can be applied to CB software which reuses COTS components.

2.4.9. Summary of the related work

A summary of the advantages and disadvantages of the approaches discussed in Section 2.4 is shown in Table 8.

The existing techniques discussed above cannot determine the test adequacy of components especially when design and source code are not available to the component user because:

i) The component user cannot apply traditional test adequacy techniques such as coverage of models or source code.

ii) The component developer can use traditional CDCT techniques to certify the component. However, the test cases executed to certify the component are independent of the usage context of the component. Moreover, the testing can be biased.

Even if the source code is available, all these techniques, except user certification, do not test the component for the context of reuse.
Table 8: Comparison of CBT approaches

<table>
<thead>
<tr>
<th>Approach</th>
<th>✓ Pros</th>
<th>X Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Built-in testing</td>
<td>• Increase testability</td>
<td>• Increase component size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Static test cases</td>
</tr>
<tr>
<td>Component metadata</td>
<td>• Increase testability</td>
<td>• Affect Implementation transparency</td>
</tr>
<tr>
<td></td>
<td>• Dynamic test cases</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• No increase in component size</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Generate test data</td>
<td></td>
</tr>
<tr>
<td>Third-party certification</td>
<td>• Impartial testing</td>
<td>• May be too costly for small organisations to afford</td>
</tr>
<tr>
<td>Developer certification</td>
<td>• Test cases are available to user to re-execute</td>
<td>• Context-independent testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Testing is biased by the component developer</td>
</tr>
<tr>
<td>User certification</td>
<td>• Context-dependent testing</td>
<td>• No test adequacy criteria for component reuse</td>
</tr>
<tr>
<td>Component contracts</td>
<td>• Does not require source code</td>
<td>• Detects only contract-related defects</td>
</tr>
<tr>
<td></td>
<td>• Lesser effort is required to develop tests compared to traditional coverage techniques</td>
<td></td>
</tr>
<tr>
<td>Testable architecture</td>
<td>• Increase testability</td>
<td>• Affect Implementation transparency</td>
</tr>
<tr>
<td></td>
<td>• No increase in component size</td>
<td></td>
</tr>
<tr>
<td>Application of IEC Standards</td>
<td>• Increase dependability</td>
<td>• May require extra effort for developing components</td>
</tr>
<tr>
<td>Application of IEEE Standards</td>
<td>• Increase reliability</td>
<td>• Context-independent testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• IEEE -1517 does not cover COTS components without source code</td>
</tr>
<tr>
<td>MD-CDCT</td>
<td>• Model-driven</td>
<td>• Requires component certification (i.e. test cases used to certify the component)</td>
</tr>
<tr>
<td></td>
<td>• Context-dependent testing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Addresses the problem of test adequacy of component</td>
<td></td>
</tr>
</tbody>
</table>

2.5. Discussion

The techniques [49-52] which use component-metadata to perform CDCT neither evaluate nor extend the adequacy of CDCT in contrast to our approach. Moreover, our approach takes advantage of the testing done at the developer’s end which is an advance over existing work.
Built-in testing techniques [17, 44-48] test the context-dependent behaviour of the component in common with our approach. However, these approaches do not have any mechanism to extend the test adequacy of component testing.

Existing mutation-based approaches [32, 34, 64-66] mutate the interface (contract) of components to evaluate the adequacy of component testing. These approaches detect defects which are related to understanding and implementation of the interface (contract). However, the defect-detection capability of the enriched test suite (which is produced by our approach) is not limited to interface or contract-related defects.

Techniques based on testable architecture [67-69] facilitate the component user to execute test cases. However, these techniques do not address the problem of test adequacy of components.

A salient feature of MD-CDCT is the reuse of existing testing information for testing purposes. Testing is a costly activity, and testing a component for a specific context from scratch requires covering a lot of the ground already covered by the developers. MD-CDCT allows us to use the existing testing information for the component to identify gaps and to focus on the usage context, and hence it is efficient in achieving our testing aims. Moreover, MD-CDCT evaluates the test adequacy of components and extends it, and tests the component with context-dependent test cases.

In this chapter, we only provide an overview of the framework, with details following in later chapters in order to i) make it easier to understand, and ii) to illustrate using a simple example, before going into the detail. MD-CDCT is based on MTCG (Section 3.3) and TSC (Section 4.2) and it inherits the advantages and dis-advantages of the underlying tools. MD-CDCT has the following potential advantages which will be discussed and evaluated later (Sections 3.4.7, 3.6, 4.3.1.6, 4.3.2.8 and 4.5):

i) It reuses the existing testing information (test suites) for testing purposes.

ii) It evaluates the component testing done at the developer’s end, and extends it for the context in which the component is reused.

iii) It can be used for the testing of COTS components as it does not require the source code of the component.

iv) It is based on use-case scenarios and interaction diagrams. Since these descriptions of behaviour are constructed at an early stage, testing based on them can start early in the life cycle.
Similarly, MD-CDCT has the following limitations which will be discussed and evaluated later (Sections 3.7 and 5.6.2):

i) It requires the test cases used to certify the component to evaluate and extend test adequacy of component testing.

ii) It is based on comparison of test suites. For complex systems, the extraction of abstract test cases from concrete test suites is a non-trivial task as discussed in Chapter 4.

The framework introduced in this chapter is the core of our research. In the next two chapters, we introduce and explain the tools (MTCG and TSC) which support the framework. In Chapter 5, we apply the framework to a real case study for evaluation purposes.
CHAPTER 3 - MODEL-DRIVEN TEST CASE GENERATION

3.1. Introduction

In this chapter, we describe our model-driven method (MTCG) for generating executable test cases (task 2 in Figure 2). We apply MTCG to a case study to demonstrate its viability, and discuss the threats to the validity of this case study. We present the related work in the area of model-based and model-driven testing. Finally, we discuss advantages and limitations of MTCG.

MTCG (Model-Driven Test Case Generation) is a novel method that uses the model transformation technology of MDA to automate the generation of unit test cases from software models. While executing the generated test cases, MTCG can monitor the method invocation chains to verify the behaviour of systems for which the source code is available. We use MTCG as a part of our model-driven framework for context-dependent testing of components (MD-CDCT) proposed in Chapter 2. In MD-CDCT, we devise CDCT from the component usage model, compare it with CT to identify the weaknesses of CT, and enrich component testing or CDCT to target the component functionality which is not tested during CT. We can then use MTCG to generate concrete and executable test cases for the enriched test suite. These test cases are used to test the component for the new context.

This chapter is partially based on the publication regarding the automated generation of test cases [82]. However, the work presented in the published material was updated during the research. The proposed tool was refined to generate readable test cases. The related work was revisited to cover the recent work done in this area. Finally, some limitations of the tool were investigated and discussed.

3.2. Model-Driven Testing

Traditionally, software products have been tested based upon either their specifications or implementations [83]. Recently, model-based testing has become popular [84-90]. Model-based testing refers to processes, methods or technologies that use software models to support testing activities. Some of the reasons for its popularity are the following:

1. The software models can contain both static and behavioural information, which provides a sound base for conducting testing activities [91].
2. As the software models are constructed at an early stage of software development, testing activities (e.g. generation of test cases) can start early in the software life cycle.

3. The software models are mostly defined using the Unified Modelling Language (UML) which is the de facto industry standard [35] for software modelling.

4. Testing based on software models is independent of the software implementation, which increases the likelihood of discovering implementation-related defects.

More recently, model-driven testing has started to emerge. Model-driven testing is a form of model-based testing that is based on models, meta-models, transformation specifications, and model transformation tools (MTTs) [92]. Models are the basic artifacts that are manipulated for automating software development activities. The meta-models are the definitions of these models and they are used by MTTs to interpret the models. Transformation specifications are the rules that specify the model transformations. MTTs are used in model-driven (MDA-based) techniques to execute the transformation rules on the models to carry out their transformations. Model transformations can be horizontal transformations or vertical transformations. A horizontal transformation is one that maintains the abstraction level, e.g., a transformation from a Platform Independent Model (PIM) to another PIM. A vertical transformation is one that changes the abstraction level, e.g., a transformation from a PIM to a Platform Specific Model (PSM). Some of the advantages of model-driven testing are an increase in productivity and quality [93].

3.3. MTCG: A Method for Automating Test Case Generation

We propose a model-driven method (MTCG) to generate test cases from UML diagrams [82], and support it by a prototype tool (MTCG Prototype). It uses the model transformation technology of MDA to generate unit test cases from a PIM of the system.

To demonstrate its viability, we chose the generation of test cases for xUnit family members from sequence diagrams [35] that describe dynamic interactions among the components of a system.

Sequence diagrams are a kind of UML interaction diagram [35] which show interactions among components of a system in a time sequence manner. The UML interaction diagrams represent dynamic interactions among objects, components or sub-systems. Sequence diagrams play an important role in the software development processes that are
use-case driven [36], such as in the Rational Unified Process [37]. A sequence diagram specifies a run-time scenario in a graphical manner. It shows processes or objects as parallel vertical lines which are called lifelines. The messages exchanged between them are shown as horizontal arrows in the order in which they are sent.

xUnit [94] is a family of unit-testing frameworks used to write and run repeatable tests for software applications. Developers use these frameworks for developing and executing unit test cases, and for regression testing. Amongst the most popular family members of xUnit are JUnit [25] and SUnit [95], which are unit testing frameworks for testing Java and Smalltalk components respectively.

In MTCG, first we model the usage of a system using sequence diagrams and then this model is automatically transformed into a general unit test case model (an xUnit model which is independent of a particular unit testing framework), using model-to-model transformations. Then model-to-text transformations are applied on the xUnit model to generate platform-specific (JUnit, SUnit etc.) test cases that are concrete and executable.

The model-to-model transformations are the horizontal transformations which maintain the abstraction level. The model-to-text transformations are the vertical transformations which produce textual outputs from a structured model.

An overview of the approach is shown in Figure 9. The generation of test cases is performed in two steps. In the first step, a UML sequence diagram is translated into a testing model using a horizontal transformation. In the second step, the testing model is converted

---

**Figure 9: Overall process for generating test cases**

An overview of the approach is shown in Figure 9. The generation of test cases is performed in two steps. In the first step, a UML sequence diagram is translated into a testing model using a horizontal transformation. In the second step, the testing model is converted
into a concrete and executable test case using a vertical transformation. This two-step approach differs from existing techniques which generate test cases for a particular platform. Further, they do not fully benefit from the model transformation technology unlike our approach. Some of the researchers have used transformation specifications but they have not provided any tool support. The intermediate xUnit model facilitates the reuse of code (Section 3.4.6) and the developer only provides vertical transformations for generating test cases for new platforms. The novelty of this technique is discussed in Sections 3.5.2 and 3.6 in detail.

Figure 10 provides an architectural view of the concept presented in Figure 9 for generating test cases. Artefacts 1 and 2 in Figure 10 are the meta-models for a UML diagram and xUnit models respectively. Artefacts 3 and 4 are the transformation rules for horizontal and vertical transformations. Artefacts 5 and 6 are the two transformation engines (MDA tools) that we use in our methodology to perform horizontal and vertical transformations (Figure 9). Artefact 7 is the source model of the application (an instance of Artefact 1) from which we generate test cases. Artefact 8 is the xUnit model (an instance of Artefact 2) which is a testing model. The horizontal transformation engine executes horizontal (model-to-model) transformations on the source model (7) to generate this xUnit model (8). Artefacts 9 is the final output which is a concrete and executable unit test case produced by the vertical transformation engine. The vertical transformation engine executes vertical (model-to-text) transformations on the xUnit model (8) to generate the unit test case (9).

Figure 10: Architectural Diagram
We have implemented the transformations in a prototype tool based on the Tefkat [96] and MOFScript [97] MTTs, which are both implemented as Eclipse [88] plugins. We have two versions of the implementation (for JUnit and SUnit). As an example, the JUnit implementation is discussed in this section; the SUnit implementation is similar.

We test a system using sequence diagrams at two levels. At the first level, we generate test cases from a sequence of method calls (messages) that are selected by the tester from the sequence diagram. Typically, the selected method calls originate from a particular lifeline in the sequence diagram and they appear as method invocations in the generated test case. Note that method invocations that originate from subsequent lifelines are invoked indirectly by the selected method calls. At the second level, we capture method execution traces during the execution of test cases to ensure that this happens as specified in the sequence diagram. Test results are checked by comparing expected and actual return values of the selected method calls, and by comparing the execution traces with the method calls in the sequence diagram.

### 3.3.1. Generating test cases using model-driven architecture

As shown in Figure 9, the model-driven approach that we use for generating unit test cases consists of two steps. The first step is the creation of a test case which is generic to all xUnit family members and the second step is the transformation of the generic test case into a concrete one, specific to a particular xUnit family member, e.g. a JUnit test case. In the first step, we model a sequence diagram as a sequence of methods calls which is then automatically transformed into an xUnit model by applying model-to-model transformations using Tefkat. Tefkat is a model transformation engine which defines and executes mappings from a set of source metamodels to a set of target metamodels. In the second step, JUnit test cases are generated from the xUnit model by applying model-to-text transformations using MOFScript. This process is shown in Figure 11.

Artefacts 1 and 2 in Figure 11 are the meta-models for a SMC and xUnit respectively. Artefacts 3 and 4 are transformation rules for model-to-model and model-to-text transformations. Artefacts 5 and 6 are the two transformation engines (MTTs) that execute horizontal and vertical transformation specifications (Figure 10). Artefact 7 is the source model of the application from which we generate test cases. Artifact 8 is the xUnit model which is an intermediate output. Tefkat executes horizontal transformations on the source model (7) to generate this xUnit model (8).
Artifact 9 specifies test data, for the SMC model, which contains the parameter values and expected return values of the method calls in the sequence diagram. By changing the contents of the test data file, different test cases can be generated for the same sequence diagram. Artifact 10 is a simple text file containing code which is copied to the top of the file containing the test case. It can be used to define packages, import classes, etc., which are needed for compiling and executing the generated test cases. Artifact 11 is the final output which is a concrete and executable unit test case produced by MOFScript. MOFScript reads the test data (9) and code header (10) while executing the vertical (model-to-text) transformations (4) on the xUnit model (8) to generate the unit test case (11).

These artifacts are generic at different levels. Artefacts 1, 2 and 3 are independent of platform, application and sequence diagrams. Artifact 4 is specific to a platform but independent of application and sequence diagram. None of these artifacts (1, 2, 3 and 4) need to be modified when testing different applications on the same platform. To test an application from a sequence diagram, the tester must provide the SMC model (7), test data (9) and the code header (10) file. Note that by altering the test data file, the same testing scenario can be executed with different test data.

![Figure 11: Overview of methodology](image-url)
3.3.1.1. Step 1: Transforming SMC into xUnit

We transform the SMC model into an xUnit model by using Tefkat transformation rules (artefact 3 in Figure 11). Transformation rules (specifications) define how the elements of a source model should be translated into elements of a target model. As an example, the rule in Figure 12 creates a test case in the xUnit model for every SMC in the model. The test case is given the same name as the SMC. The detailed syntax of the transformation specification language of Tefkat is available online [98]. All the Tefkat rules that we have devised for the transformation are provided in Appendix B.1.

<table>
<thead>
<tr>
<th>RULE</th>
<th>SMC_2_TestCase ( smc, testCase )</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORALL</td>
<td>SMC smc</td>
</tr>
<tr>
<td>MAKE</td>
<td>TestCase testCase</td>
</tr>
<tr>
<td>SET</td>
<td>testCase.name = smc.name ;</td>
</tr>
</tbody>
</table>

Figure 12: An example Tefkat rule

3.3.1.2. Step 2: Generating JUnit from xUnit

MOFScript transformation rules are used to generate JUnit test cases from the xUnit model. Two example MOFScript transformation rules are presented in Figure 13. The rule `model.TestSuite::main` is the entry-point rule where the transformation starts. The expression `self.name` is the name of the object on which the rule is being executed, i.e. the name of the test suite in this case. The `forEach` keyword iterates over the collection of test cases in the test suite and invokes the rule `model.TestCase::mapTestCase` to process them. This rule creates a JUnit specific test case and invokes other rules (that are not discussed in detail) to complete the body of the test case. The detailed syntax of the transformation specification language of MOFScript is available online [99].

We have implemented MOFScript transformation rules for generating JUnit and SUnit test cases from the xUnit model. All the MOFScript rules are provided in Appendix B.2.
To generate test cases from sequence diagrams, we needed methods calls, parameter values and expected values. Therefore, we confined our implementation to a meta-model of Sequences of Method Calls (SMCs) abstracting away the unnecessary details such as connectors, message-occurrence-specifications, message-ends and message-events [85].

Our meta-model for SMCs is shown in Figure 14. It consists of interactions, messages, classes, parameters, expected values and literal strings. In this model, NamedElement represents a named value and LiteralString represents a string value. An Interaction represents part of a sequence diagram. The Messages contained in the interaction are a subset of the method calls of the sequence diagram selected by the tester. The messages can have Parameters and an optional ExpectedValue in them. The parameters and the expected value are of type ScalarValue or ComplexValue. The ScalarValues are atomic data values that do not contain any other data values. Instances of ScalarValues in Java are integer, float, String, etc. The ComplexValues are the values that contain other values, i.e., they act as data structures. The ComplexValues in Java are all classes except String. Moreover, every message is associated with an OwnerClass (to which the methods belongs), which is a class that receives the message. The owner class has parameters for its constructors that are required to create an instance of the class in the generated test case.
3.3.1.4. xUnit Meta-model

The meta-model for xUnit test cases is shown in Figure 15. No meta-model for xUnit was available, so we derived it by studying the architecture of test cases written in different unit testing frameworks such as JUnit and SUnit. In this model, the Test Suite acts as a container for Test Case(s). A test case can have Assertions in it. An assertion is a condition that should hold true after executing the test case. An assertion can be of different types which are specified by its attribute type, e.g. the JUnit framework has Equal, Not Equal, Same, Not Same, True and False assertions. For testing using sequence diagrams, an assertion has a method call and an expected value. The method call is the code to be tested.

After executing a test case, the unit testing framework compares the actual value (the value returned after executing the code) with the expected value to decide on the success or failure of the test case. As an example, the JUnit’s Equal assertion compares the actual value and the expected value. If both values are equal, the assertion holds. Conversely, the Not Equal assertion holds if the values are not equal. Moreover, the method can have parameters.
that are either scalar values or complex values as discussed in the meta-model of SMC. The elements **Message**, **OwnerClass**, **ComplexValue**, **ScalarValue** and **DataValue** are the same as in the SMC meta-model.

![xUnit meta-model](image)

**Figure 15: xUnit meta-model**

### 3.3.2. Tracing

Apart from comparing the expected values with the values returned by method calls (to decide on the success of a test case), we also monitor the method invocation chains to have a second check on the behaviour of the system. We do this by means of the Daikon [100] tracing tool. We compare the observed method execution chain with the expected method execution chain in the sequence diagram. Currently, we compare the traces manually, but this activity can be automated in the future. However, the tracing will not be possible for components whose source code is not available.

### 3.4. Case Study: Determining Applicability of MTCG

#### 3.4.1. Introduction

The objective of this case study is to demonstrate the applicability of MTCG. We use a system whose source code is available so that we could apply MTCG fully by performing tracing as well as generating test cases.
3.4.2. Object description

We have selected an Automatic Teller Machine (ATM) simulation system for this case study [101]. An ATM allows its users to perform basic banking operations like withdrawal, deposit, transfer and checking their balance, without having to go to a bank. In an ATM, the user inserts an ATM card, enters a PIN (a personal identification number), selects a transaction to be performed and provides input needed for the transaction, e.g. amount and account in case of withdrawal. In response to the user’s actions, the ATM reads the card, reads and validates the PIN, processes the transaction, dispenses cash, prints the receipt and ejects the card at the end of the session.

3.4.3. Interface of ATM

The ATM system is implemented using the following classes: ATM, Simulation, Session and Transaction. The ATM class represents an ATM. The Simulation class simulates the process of reading an ATM card and PIN. The Session class represents a session which is a period of time used for a particular banking activity. The Transaction class creates and executes banking transactions which are deposit, withdraw, transfer and balance inquiry. The interface of ATM consists of creating (or invoking) the following objects (or methods):

1. Create an ATM object
   
   \[ATM \text{ atm} = \text{new ATM} (id, \text{ name, place, address});\]

2. Create a Simulation object
   
   \[\text{Simulation simulation} = \text{new Simulation (atm)};\]

3. Create a Session object
   
   \[\text{Session session} = \text{new Session (atm)};\]

4. Invoke \text{readCard()} method of the simulation object
   
   \[\text{Card card} = (\text{Card}) \text{simulation.readCard}();\]

5. Invoke \text{readPIN()} method of the simulation object
   
   \[\text{int pin} = \text{simulation.readPIN}();\]

6. Invoke \text{makeTransaction()} method of \text{Transaction} class
   
   \[\text{Deposit d=(Deposit)Transaction.makeTransaction(atm, session, card, pin);}\]

7. Invoke \text{performTransaction()} method of the desired transaction (e.g. Deposit) object
   
   \[\text{Deposit.performTransaction}();\]
3.4.4. Experimental planning

In this section, we provide only an overview of the tasks performed in this case study with the details presented in subsequent sections. We execute the following tasks to generate test cases:

1. Identify sequence of method calls (SMC)
   We select messages that originate from a particular lifeline to observe the behaviour originating from this lifeline. The details are provided in Section 3.4.5.1.

2. Create SMC model using these method calls (messages)
   The SMC model is created manually using the graphical user interface provided by Eclipse. The details are provided in Section 3.4.5.2.

3. Devise test data for the SMC model
   The test data consists of the parameters and return values for the selected messages. The details are provided in Section 3.4.5.4.

4. Execute horizontal and vertical transformations to generate test cases
   We execute horizontal and vertical transformations to generate SUnit and JUnit test cases. The details are provided in Sections 3.4.5.3 and 3.4.5.5.

3.4.5. Applying MTCG to the case study

To test the ATM system, we generate test cases (using MTCG) from the following sequence diagrams: withdrawal, deposit, transfer and balance inquiry. These sequence diagrams are derived from the information available with the ATM system [101]. For instance, the withdrawal sequence diagram shown in Figure 16 was based on the interaction diagram ‘Withdrawal Transaction Diagram’ in the design section [101].

3.4.5.1. An example sequence diagram

Figure 16 shows the sequence diagram for the withdrawal operation. The details like validation of the PIN and interaction with the bank are omitted to simplify the example. The sequence diagram consists of the following classes. The class Session represents a particular session, i.e. the operations that the user performs between inserting a card and the card being ejected. The Simulation class is used to represent the interaction between the ATM system and the devices attached to it, which are the card reader, cash dispenser, and customer console. The class ATM represents a particular ATM terminal. The class Transaction is a base class that instantiates all the transactions. The class Withdrawal represents a withdrawal
operation, which is instantiated when the user opts to withdraw an amount. The CardReader class reads the ATM card and ejects it at the end of the session. The CustomerConsole gets input from the user, like amount, PIN etc. The class CashDispenser dispenses cash as a result of a valid withdrawal request from the user.

In this sequence diagram, the first three messages are setup messages that are required to create ATM, Session and Simulation objects. The messages to the devices, attached to the ATM system, are invoked on a Simulation object. The Simulation object delegates these messages to the representation classes of these devices. The message readCard is invoked on CardReader objects and returns an ATM card object. The card object has an integer attribute. The message readPIN reads the PIN from the CustomerConsole. It takes a string parameter promptMessage (that is displayed on the customer console) as a parameter and returns the pin (entered by the user) as an integer. The method makeTransaction creates a Withdrawal object when the customer selects an amount. It takes ATM, session, card and PIN as parameters. The method performTransaction reads the amount to withdraw and the account to withdraw from. It ensures that the amount is within the daily withdrawal limit and that the cash dispenser has enough cash to satisfy the request. It then dispenses the cash.

To generate unit test cases for CT using sequence diagrams, we select messages that originate from a particular lifeline to observe the behaviour originating from this lifeline. For this case study, we select the messages that originate from the high-level Session object (that are represented with bold arrows in Figure 16). They are transformed to method invocations in the generated test case. The messages that originate from other objects (that are represented with thin arrows in Figure 16) are invoked indirectly and we use tracing to verify their execution chain (as discussed in Section 3.3.2).
3.4.5.2. SMC model

We create the input instance (SMC model) in the Eclipse editor, where it can be accessed by the Tefkat plugin. We have created this instance manually using the graphical user interface provided by Eclipse. The Eclipse definition for the above SMC (in Figure 16), is shown in
Figure 17. The message \textit{SETUP-1} is a setup message that creates an \textit{ATM} object with id, name, place and address as its constructor’s parameters. Similarly the messages \textit{SETUP-2} and \textit{SETUP-3} create \textit{Simulation} and \textit{Session} objects. Setup messages do not generate any method calls in the test case. They are a part of configuration and environment setup. The rest of the messages have parameters and return values.

To keep the creation of the test data file simple, we define the sub-structure (attributes) of objects only once. Therefore, when an object that is not a simple data type is used in a message as an owner class or a parameter for the first time, its sub-structure is specified along with the types of its elements e.g. \textit{int, float} or \textit{objects}. The reason for this is that \textit{MTCG} generates code for the object the first time it encounters the object and stores it in a hash table. Later when this object is referred to in a message call, its sub-structure does not need to be specified. For example, in Figure 17, when the object \textit{ATM} is used for the first time in the message \textit{SETUP-1}, its id, place, name and address are specified as the constructor’s parameters. But when this object is passed as a parameter in the method \textit{makeTransaction}, its sub-structure is not specified again. Whenever a reference is made to this object during method invocation, the variable stored in the hash table is retrieved. However, for an expected value, the sub-structure is specified each time, as a new object needs to be created with different data values in it.

3.4.5.3. \textbf{xUnit model}

Tefkat applies the horizontal transformations which were devised in Section 3.3.1.1 to generate an output instance, in the form of an xUnit model, from the input instance. The output instance for the input instance in Figure 17 is shown in Figure 18. In the xUnit model, a \textit{test case} is generated for each SMC. For each message in the sequence diagram, an \textit{assertion} and a \textit{method} (within the assertion) are generated. For each parameter in a \textit{message} (in the source model), a parameter is created in the \textit{method} (in the target model). Similarly for the \textit{return value} and \textit{owner class} (which represents the class that the method belongs to), corresponding elements in the target are created. The objects \textit{SETUP-1}, \textit{SETUP-2} and \textit{SETUP-3} are created just for setting up the environment and we do not generate assertions for these objects in the generated test case.
3.4.5.4. Test data file

The test data file that has the parameters and return values for the example in Figure 16 is shown in Figure 19. It has data for the calls to SETUP-1, readCard and readPIN only. The reason is that the parameters for other method calls (e.g. the parameters ATM, Session, Card and PIN of the method makeTransaction) are created by previous methods.

The values for id, place, name, card number and pin (which are 41, Gordon College, National Bank, 1 and 42 respectively) in Figure 19 are provided by the tester.

In the test data file, the class name (to which the method belongs) and a double-colon (i.e. ::) are appended before the method’s name. This is because different classes can have the same method name.
Figure 19: Test data file for the example sequence diagram

3.4.5.5. JUnit test case

MOFScript executes the vertical transformations which were devised in Section 3.3.1.2 to produce a JUnit test case. The JUnit test case is shown in Figure 20. Lines 1-8 are copied by MOFScript from the Code Header file. The remaining lines are generated by MOFScript rules. Lines 12-16 create a JUnit test suite. Line 17 contains the test method for the sequence of calls identified. Lines 20-23 create variables that are used as the constructor’s parameters for the ATM object. The data values for these variables, i.e. 41, Gordon College and National Bank and null, are read from the test data file (Figure 19). The types of these variables (i.e. int, String, InetAddress) are read from the xUnit model and originally come from the input model during the UML to xUnit transformation. Lines 20-25 contain the code generated for the method SETUP-1. As this is a setup message, it creates an ATM object for later use. The attribute values of the ATM object are read from the test data file. Lines 26-27 contain the code generated for the methods SETUP-2 and SETUP-3 respectively. They generate Simulation and Session objects that use the previously created ATM object as their constructor’s parameter. Lines 29-34 show the code generated for the call to readCard. As this method returns a Card object, an expectedCard object is generated by reading the card number from the test data file.

Lines 36-40 show the code generated for the call to readPIN. This method returns pin as an integer value. A variable expectedPIN is generated whose value is read from the test
data file. Lines 42-46 show the code generated for the method `makeTransaction`. As it returns a `Withdrawal` object, an `expectedWithdrawal` object is created. Lines 47-48 show the code generated for the call to `performTransaction` and lines 50-51 show the code generated for the call to `ejectCard`. Assertions are generated in lines 34, 40 and 45 for the methods that return a value in order to compare them with the expected values.

The assertions in lines 34 and 45 use the `equals` method, which is supplied by the tester, to compare the expected and the actual return value of a method. The `equals` method is used for comparison of non-scalar values and user-defined types. For the comparison of scalar values, the “`= =`” operator is used for comparison, as shown in line 40.

We then run these test cases to make sure they are syntactically correct and executable.
```java
package com.md;

import atm.*;
import atm.physical.*;
import atm.transaction.*;
import banking.*;
import simulation.*;
import junit.framework.*;

public class Test_Withdrawal extends TestCase {

    public static void main( String args[] ) {
        TestSuite testSuite = new TestSuite( Test_Withdrawal.class );
        testSuite.run( new TestResult( ) );
    }

    public void test_Withdrawal() {
        try {
            int id = 41;
            String name = "Gorden College";
            String place = "National Bank";
            InetSocketAddress address = null;

            ATM atm = new ATM ( id, name, place, address );
            Simulation simulation = new Simulation ( atm );
            Session session = new Session ( atm );

            // Code for method readCard()
            Card expectedCard = new Card();
            int number = 1;
            expectedCard.setNumber( number );
            Card card = (Card) simulation.readCard();
            assertTrue( expectedCard.equals( card ) );

            // Code for method readPIN()
            String parameterString1 = "Please enter PIN."
            int expectedPIN = 42;
            int pin = simulation.readPIN( parameterString1 );
            assertTrue( expectedPIN == pin );

            // Code for method makeTransaction()
            Withdrawal expectedWithdrawal = new Withdrawal( atm, session, card, pin );
            Withdrawal withdrawal = (Withdrawal) Transaction.makeTransaction( atm, session, card, pin );
            assertTrue( expectedWithdrawal.equals( withdrawal ) );

            // Code for method performTransaction()
            withdrawal.performTransaction();

            // Code for method ejectCard()
            simulation.ejectCard();
        } catch ( Exception exp ) {
            System.out.println( exp.toString() );
            fail("Exception");
        }
    }
}
```

Figure 20: JUnit test case for the example sequence diagram
3.4.5.6. Traces

During the execution of these test cases, we capture their traces using the Daikon tracing tool. The trace captured during execution of the test case in Figure 20 is shown in Figure 21. The method invocations with grey background are those that are not included in the test case but that are invoked by the methods that are called in the test case. For example, the test case (in Figure 20) executes the method `readCard` of `Simulation`. This method further invokes the method `readCard` of `CardReader` which is responsible for reading the card. This allows us to check that the classes interact as specified in the sequence diagram for this test case.

```
Simulation.readCard::ENTER
CardReader.readCard::ENTER
CardReader.readCard::EXIT
Simulation.readCard::EXIT

Simulation.readPIN::ENTER
CustomerConsole.readPIN::ENTER
CustomerConsole.readPIN::EXIT
Simulation.readPIN::EXIT

Transaction.makeTransaction::ENTER
Simulation.readMenuChoice::ENTER
CustomerConsole.readMenuChoice::ENTER
CustomerConsole.readMenuChoice::EXIT
Simulation.readMenuChoice::EXIT
Transaction.makeTransaction::EXIT
...
```

Figure 21: Execution trace for the example sequence diagram

3.4.6. Reuse of vertical transformation rules

We analyse the vertical transformation rules for their similarity to find the possibility of code reuse. The code reuse shows the advantage of having a two-step strategy for generating test cases.

Most of the vertical transformation rules (xUnit to SUnit and xUnit to JUnit) have similar logical structure that makes them reusable. They differ only in the text that is embedded in them, e.g. for SmallTalk the statement terminator is a dot (.) whereas in Java it is a semi-colon (;). The similarity of the logical structure of three example rules, `mapAssertion`, `mapExpectedValue` and `mapMethod` is illustrated in Figure 22. For the rules having similar structure, the implementer only needs to copy and change the language-specific syntax in these rules. Table 9 shows the reusability in terms of non-commented lines of code that are the same. This reusability is achieved at minimal cost due to the intermediate
xUnit model. The structural mapping between SMC and xUnit is addressed during the horizontal transformations, leaving the vertical transformations linear and almost identical except for language-specific syntax.

**Figure 22: M2T rules for mapAssertion, mapMethod and mapExpectedValue**
The high degree of similarity of the vertical rules allows for reuse of these rules while generating test cases for different target languages. This high reusability, which is 85%, is achieved using the intermediate xUnit model. The reuse of models and transformation rules is not uncommon in model-driven development but the level of reuse in this particular application is over 80%, which is significant. Therefore, our technique has an advantage over existing techniques in terms of code reusability.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Structure</th>
<th>xUnit-SUnit</th>
<th>xUnit-JUnit</th>
<th>Same</th>
<th>Reuse-%</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>Different</td>
<td>27</td>
<td>25</td>
<td>17</td>
<td>63</td>
</tr>
<tr>
<td>mapTestCase</td>
<td>Different</td>
<td>13</td>
<td>20</td>
<td>11</td>
<td>55</td>
</tr>
<tr>
<td>mapAssertion</td>
<td>Same</td>
<td>48</td>
<td>47</td>
<td>44</td>
<td>92</td>
</tr>
<tr>
<td>mapMessage</td>
<td>Same</td>
<td>32</td>
<td>28</td>
<td>25</td>
<td>78</td>
</tr>
<tr>
<td>mapOwnerClass</td>
<td>Same</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>100</td>
</tr>
<tr>
<td>mapParameter</td>
<td>Same</td>
<td>58</td>
<td>56</td>
<td>51</td>
<td>88</td>
</tr>
<tr>
<td>mapConstructorParameter</td>
<td>Same</td>
<td>54</td>
<td>52</td>
<td>48</td>
<td>89</td>
</tr>
<tr>
<td>mapExpectedValue</td>
<td>Same</td>
<td>36</td>
<td>34</td>
<td>30</td>
<td>83</td>
</tr>
<tr>
<td>mapComplexAttribute</td>
<td>Same</td>
<td>28</td>
<td>28</td>
<td>26</td>
<td>93</td>
</tr>
<tr>
<td>mapSimpleAttribute</td>
<td>Same</td>
<td>26</td>
<td>25</td>
<td>22</td>
<td>85</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>330</strong></td>
<td><strong>323</strong></td>
<td><strong>282</strong></td>
<td><strong>85</strong></td>
</tr>
</tbody>
</table>

Table 9: Code reusability matrix of M2T rules

3.4.7. Discussion

This case study shows that MTCG can generate concrete and executable test cases by making use of model transformation tools (MTTs). We have generated test cases for withdrawal, deposit, transfer and balance inquiry operations of ATM using MTCG.

We have two versions of the implementation of MTCG to generate test cases for two different platforms (JUnit and SUnit) which are provided in Appendices B.3 and B.4 respectively. We generated test cases for the JUnit platform as the ATM system (which is used in this case study) was written in Java. We generated test cases for the second platform to show that the vertical transformations for JUnit can be modified to drive vertical transformations for other platforms with ease. For this purpose, we needed a platform whose syntax differs greatly from Java. Therefore, we selected SUnit because the syntax of
Smalltalk (the language of SUnit) differs substantially from Java, e.g. declaring and initialising variables, passing arguments to methods, etc.

*MTCG* can generate test cases for CT and CDCT. When *MTCG* was proposed and implemented, our objective was to generate unit test cases for CT using sequence diagrams. Therefore, we picked messages that originate from a particular lifeline to observe the behaviour originating from the lifeline. To use this method for CDCT, we can select the method calls that are directed to the lifeline which represents the component (in the sequence diagram), i.e., we focus on incoming calls to the component instead of selecting outgoing method calls.

### 3.4.8. Threats to validity

This case study has the following threats to its validity which affect the application of *MTCG* to other systems:

1. We have generated test cases for SUnit and JUnit only. Hence, we cannot generalise the ability of *MTCG* to generate test cases for unit testing frameworks other than xUnit ones.
2. We have generated test cases from sequence diagrams which does not demonstrate the applicability of *MTCG* to systems which are modeled using other UML diagrams. However, *MTCG* derives SMCs as an intermediary to generate test cases. We can therefore generalise the applicability of *MTCG* to other modelling diagrams, such as statecharts [43], that can be used to derive SMCs.
3. We have generated test cases for one small system, and hence the results cannot be generalised to other systems.

### 3.5. Related Work

As our research focuses on model-driven testing of component-based software using interaction diagrams, we will review related work in the following areas:

1. Model-based testing in general, and testing using interaction diagrams in particular
2. Model-driven testing
3.5.1. Model-based testing

The Unified Modelling Language (UML) is the de facto industry standard for modelling software applications [35]. Researchers are investigating different types of UML diagrams to support software testing activities. The UML diagrams that are widely used to automate software testing are: use case, activity, state machine and interaction diagrams. Use cases represent a specific use of a system. They are used for automatic generation of test cases for implementation verification [102, 103]. Implementation verification determines whether the software is developed according to its specifications (requirements). Activity diagrams are an object-oriented equivalent of Flow Charts and Data Flow Diagrams. They are used for validating software workflows [104] and generating test cases for implementation verification [105-107]. State machines represent the state-dependent behaviour of a system. They are used for validating software models [108-110] and implementation verification [111-117].

UML interaction diagrams [35] represent dynamic interactions among the components of a system. They consist of collaboration diagrams and sequence diagrams. The former emphasise the organisational structure of interacting components, and the latter show interactions in a time sequence manner. Interaction diagrams are used to validate software models, verify software implementations, determine test requirements and generate test data.

3.5.1.1. Model validation using interaction diagrams

Pilskalns et al. [118] present an approach to generate test cases from sequence diagrams. They convert a sequence diagram into an Object Method Directed Acyclic Graph (OMDAG) such that its objects become the nodes and its method calls become the edges of the graph. The paths in the OMDAG are augmented with test information (different attribute values and parameter values of methods) that is used to generate test cases.

In common with MTCG, they generate test cases from sequence diagrams. However, the test cases generated using this approach validate the software model, i.e. they are aimed at finding defects in software models. However, MTCG generates test cases to verify software implementations.

3.5.1.2. Implementation verification using interaction diagrams

Basanieri and Bertolino [119] propose the UIT (Use Interaction Testing) methodology to generate test cases by analysing use case diagrams and interaction diagrams. Later, they
proposed a test strategy, named CoWTeST (Cost Weighted Test STrategy) [120], for selecting and prioritising test cases using the UIT methodology. They automated this strategy by implementing a tool, named CowSuite [121], that generates test cases using use case and sequence diagrams.

Wittevrongel and Maurer [122] develop a model-based tool, SCENTOR, which creates functional test drivers for e-business applications from sequence diagrams that have test data (parameters and expected values of method calls) embedded in them.

Fraikin and Leonhardt [36] develop another model-based tool, SeDiTeC, which generates test stubs using sequence diagrams that are augmented with test data. These stubs enable testing even before the completion of the system implementation.

Ashalatha et al. [123] propose to generate test cases from interaction diagrams using the flow graph. The process consists of three steps which are i) scenario graph synthesis, ii) test case synthesis, and iii) test data synthesis. In the first step, a scenario graph is derived from an interaction diagram using the objects, messages and fragments (i.e. alternate path, loops, parallel blocks). In the second step, the scenario graph is converted into an intermediate testable model from which test scenarios (abstract test cases) are generated. In the third step, the domain definitions for each scenario are identified which are used to determine input and output values (test data) for test scenarios to form a test case.

Pasternak et al. [124] propose a tool, GenUTest, to generate unit tests using the program execution traces. GenUTest logs inter-object interactions (that occur during the execution of Java programs) using AspectJ. These interactions are used to generate unit tests and mock objects. The mock objects simulate the behaviour of real objects to allow for unit testing in isolation.

Sarma et al. [125] present a model-based approach for generating test cases using UML sequence diagram. They transform use case diagram Use case Diagram Graph (UDG), and sequence diagram into Sequence Diagram Graph (SDG). Then, they generate System Testing Graph (STG) by integrating UDG and SDG. Finally, the test cases are generated by traversing the STG. The test cases can uncover interaction and scenario faults during system testing.

Swain et al. [126] propose a technique for generating test cases using the features of UML 2.0 sequence diagram such as conditions, iterations, asynchronous messages and concurrent components. In this approach, test cases are derived from analysis artefacts such
as use cases, their corresponding sequence diagrams and constraints specified across these artefacts. Test case generation consists of the following steps:

1. Create activity diagram form the use case diagram.
2. Construct Use case Dependency Graph (UDG) using the activity diagram.
3. Construct Concurrent Control Flow Graph (CCFG) from the sequence diagrams.
4. Generate test sequences from the UDG and CCFG.

The test case generation technique can be used for integration and system testing. The test cases generated are suitable for detecting object interaction and operational faults. They derive test cases using full predicate coverage criteria unlike [125] which generates test case using structural coverage such as message path criterion from sequence diagram. They implement their approach in a prototype tool called ComTest.

Cartaxo et al. [127] present a technique for feature testing of mobile phone applications using UML sequence diagrams and Labeled Transition Systems (LTS). A feature is an increment of functionality that is added on top of a basic system. A feature is usually developed separately from the basic system as an independent component (module). They translate UML sequence diagrams translated into LTSs. Test cases are automatically derived from the LTSs. They evaluate their approach on Motorola mobile phone applications.

Li et al. [128] propose an approach for generating test cases using UML sequence diagrams and Object Constraint Language (OCL). To generate a test case, first they construct a tree representation of a sequence diagram. Then, they traverse the tree and identify conditional predicates on the sequence diagram. Finally, they generate test data from the predicates. The pre- and post- conditions are derived using OCL. This technique can be used in MTCG to generate test data for its test cases which is currently provided by testers.

Nayak and Samanta [129] suggest the UML 2.0 interaction diagrams use operation fragments which require testing approach to derive a comprehensive system behaviour for test case generation in the presence of multiple nested fragments. They propose an approach to derive flow of controls from interaction diagrams. They simplify the flow of controls using control primitives of the UML fragments to transform it to a testable form known as Intermediate Testable Model (ITM). Later, they [130] proposed an approach of synthesizing test data from the information embedded in model elements such as class diagrams and sequence diagrams.
Shanthi et al. present [131] a technique for generating test cases from sequence diagram. They create a sequence diagram using IBM Rational Rose. Then, they extract the necessary information from the sequence diagram using a parser written in Java. Based on the extracted information, a Sequence Dependency Table (SDT) is generated. Test cases are generated from the SDT by applying the genetic algorithm.

In the work presented above [36, 119-124, 127, 129], researchers have used sequence diagrams to perform implementation verification similar to MTCG. Some have developed tools to support their approaches, however these tools are model-based but not model-driven. They do not take advantage of the model-transformation technology (MDA). Similarly, some researchers have used interaction diagrams for generating test data [124-126, 128, 129, 131]. Others have used these diagrams for devising test requirements [132-135].

3.5.2. Model-driven testing

Model-driven testing uses the emerging MDA technology to automate testing activities. The MDA technology uses models, meta-models and transformation specifications to leverage automation of software development activities. In this approach, models are the basic software development artefacts. The meta-models are definitions that are used for interpreting models. Transformations between models are defined by mappings between the meta-models that define the source and target models. MDA uses model-transformation tools (MTT) to execute transformations defined in this way. By automating transformations, such as from activity diagrams to test cases, MDA-based tools can reduce development time and maintenance effort in model-based testing.

Dai [136] discusses the transformation of a UML model into a UML 2.0 Testing Profile (U2TP) model. U2TP is a general meta-model for testing, proposed by the Object Management Group (OMG). Dai proposes generating test cases using three transformations: i) UML model to U2TP model, ii) U2TP model to platform-specific model (PSM) and iii) PSM to a JUnit test case. However, no tool support is provided for the proposed approach.

Zander et al. [137] propose the transformation of a U2TP model into executable test cases for TTCN-3, which is a standardised test technology for test definition, implementation and execution [138]. They provide transformation rules between the source U2TP meta-model and the target TTCN-3 meta-model. However, the implementation of the tool was left for future work.
Dinh-Trong et al. [139] develop an Eclipse Plug-in for Testing UML Designs (EPTUD) that generates and executes test cases, using sequence diagrams. EPTUD transforms a UML model into an executable form (EDUT, executable design under test), adds test scaffolding (TDUT, testable design under test), executes tests and reports failures. The test cases generated by EPTUD validate the UML model whereas the model-driven approach that we propose verifies (tests) the implementation of CB software.

Engels et al. [140] present a model-driven monitoring approach in which assertions are used to monitor the behaviour (implemented by the developer) during execution. These assertions are generated from the contracts that are added to the model. These contracts represent the behaviour of the model and they consist of pre- and post-conditions of operations. They use MDA to monitor (check) contracts during program execution, whereas MTCG generates test cases for implementation verification.

Felderer et al. [141] propose a model-driven tool, Telling Test Stories (TTS), for system testing. TTS automates the generation of tests, execution of tests and generation of log files and test reports. TTS consists of a system model, test model, system implementation and test implementation. The system model contains formal system requirements at a business level based on a metamodel. A test model contains the test case specifications. The system implementation provides services callable by the test implementation which contain business logic and configuration services for testing purposes. The test implementation is generated by a compiler which transforms test story files into source code files, so called test code, of the execution language. TTS is applied to an industrial Telephony Connector system. However, this tool does not follow the MDA-style of development which executes transformations specifications for generating one model from another.

Later, they extend their work by devising a model-driven approach for testing of service-oriented systems [142]. They devise a meta-model for service-oriented systems. Their approach consists of a system model (based on system meta-models) and a test model (based on test meta-model). The test model is transformed into Java test code by a compiler using the meta-model and model-to-text transformations. Their implementation is based on the Eclipse platform. Their work is closest to MTCG in that it is uses models, meta-models and generates test cases by executing transformations. This work is different to ours as we apply model-to-model transformations to generate xUnit test cases which are platform independent. We then apply model-to-model transformations to generate test cases for a particular platform. However, their approach applies model-to-text transformations on the
test model to generate Java test cases. They use adapters for integration with different target technologies. Adapter provides an interface realisation which allows a class to communicate with an incompatible class. Moreover, they use a meta-model specific to service-oriented systems, whereas MTCG is based on a meta-model of sequence diagrams.

Schurr et al. [143] propose a model-driven approach for black-box testing of Software Product Lines (SPL). They construct a Feature Model Tree (FMT) which is based on a Feature Model (FM) and Classification Tree (CT). FM is a representation of all the products of the SPL in terms of features. CT [144] decomposes system functions into input parameters. Test case generation using FMT consists of the following steps:

1. Define equivalence classes for input parameters.
2. Apply some heuristics to select input parameters.
3. Select representative subset of all feature combinations.
4. Derive CT for selected subset of products of SPL.
5. Define test suite for the FMT approach using CT-based black-box testing.

To automate the generation of test cases, they define a meta-model for FMT and implement a tool MOFLON using MOF2.x.

Ridene et al. [145] propose MATeL (Mobile Applications Testing Language) to automate testing of mobile applications. MATel is a domain-specific modeling language built upon an industrial platform (a test bed). Testing of different applications for a variety of mobile handsets is repetitive and costly. MATeL provides a solution to this problem by allowing testers to describe test scenarios in which commonalities and differences between mobile phones can be expressed in an efficient way. These scenarios are in essence the models that conform to this meta-model (MATeL) similar to the class-instance principle in the object-orientation. MATeL is available as an Eclipse plug-in.

3.6. Discussion

MTCG has been implemented under Eclipse 3.1 [146]. It has been validated on an Automatic Teller Machine (ATM) simulation system [101]. This method is model-driven and uses model transformation technology. This is an advance over existing model-based testing approaches that do not take advantage of the emerging MDA technology. The overall process (Figure 9) is quite general in that it can be applied to different UML diagrams such as use cases, sequence diagrams or state machines.
The genericity of our method is extended by targeting xUnit testing frameworks and incorporating an intermediate phase which generates test cases in a platform-independent xUnit format (a two-step approach). Thus, the method can be used to generate test cases in any of the xUnit family by varying the backend (artefact 4 in Figure 11). This demonstrates the versatility and utility of the MDA approach to software development and tool construction. It also distinguishes our tool from other tools which typically generate test cases for one particular platform.

The SMC meta-model, the xUnit meta-model and the horizontal transformation are created only once and do not change for different platforms, systems and sequence diagrams. The developer needs to provide vertical transformations for each new platform. The tester needs to provide the SMC model, the test data file and the code header to generate test cases using this tool.

3.7. Limitations of MTCG Prototype

The prototype implementation of MTCG has the following limitations:

1. Each SMC model is created manually using the Eclipse editor. However, the creation of SMC models could be automated by reading sequence diagrams from their graphical representations.
2. The comparison of the actual trace with the sequence diagram is done manually but it can also be automated.
3. As our focus was to investigate the use of models for automating software testing, we devised our own meta-model for SMC instead of using the UML 2.0 meta-model of sequence diagrams, which is much more complex.
4. Tracing will not be possible for the components whose source code is not available.

We use MTCG in our framework (MD-CDCT) to automate the generation of concrete test cases for context-dependent testing of components. In Chapter 4, we shall introduce a method for evaluating and extending test adequacy of CDCT. The test cases of the enriched CDCT can be transformed into concrete and executable ones using MTCG.
CHAPTER 4 - COMPARISON OF TEST SUITES

4.1. Introduction

In chapter 3, we presented a model-driven method for generating a concrete and executable test suite from software models. We use this method to generate a test suite for CDCT (as shown in Figure 2). In this chapter, we describe a method for comparing the test suite for CDCT with the test suite used for CT to determine the adequacy of component testing.

In Chapter 2, we discussed the usefulness of comparing the test suites for CDCT and CT in the context of component-based software. This comparison can highlight weaknesses of the CT. If a test case relating to some functionality is present in the CDCT test suite but the test suite for CT does not have any test cases for that functionality, it shows that the component is not properly tested for the context in which it is being reused. This suggests a retesting of the component with an enriched component test suite.

In this chapter we describe a method, TestSuiteComparator (TSC), for comparing two test suites (task 3 in Figure 2) using equivalence classes. We discuss the application of this method to some case studies to show its viability and effectiveness. We then discuss the related work in the area of test suite comparison. Finally, we discuss the strengths and weaknesses of this method.

4.2. TestSuiteComparator: A Method for Comparing Test Suites

To compare two test suites, we need a criterion to determine the similarity of test cases. We use equivalence class partitioning (ECP) in TSC to compare test cases of the test suites. ECP as proposed by Myers [41] partitions the program’s input(s) and executing conditions into a finite number of equivalence classes. This partitioning reduces the total number of potential test cases to a minimal set of tests that will uncover as many errors as possible. In ECP, which is also known as category partitioning [42], testers select a representative test value for each equivalence class. The test case that results from the representative value for a class is considered "equivalent" to the test cases which are created from the other values in the same class. If the test case of the representative value does not discover any error, it is reasoned that all the other "equivalent" test cases would not identify any errors either. ECP is useful for the following reasons:
1. The large number of input values and executing conditions of a program make it practically impossible to test the program for all of them due to time and resource constrains.

2. Some of the input values and executing conditions are the same from a testing point of view.

These equivalence classes form a partitioning of a program's input domain and executing conditions for which program's behaviour is assumed to be the same [41]. These equivalence classes are defined by the tester.

Using ECP, we can devise a set of equivalence classes in different ways:

1. Two test cases are equivalent if and only if they contain the same method calls in the same order with the same executing conditions for those methods. This choice is too weak because it ignores the parameter values. Two methods with different parameters values may execute different behaviours of the system even under the same executing conditions, even though the two test cases would be considered equivalent according to this criterion.

2. Two test cases are equivalent if and only if they contain the same method calls in the same order with the same parameter values for those methods. However, this is too strong because it requires the parameter values of the two methods to be the same. It is possible that two test cases that have different parameter values execute the same behaviour, even though the two test cases which execute the same behaviour would be considered different according to this criterion.

3. Two test cases are equivalent if and only if they contain the same method calls in the same order, the parameter values for those methods belong to the same equivalence class, and the executing conditions of the test cases belong to the same equivalence class [41]. This criterion seems appropriate for our purposes in that it analyses the parameter values and executing conditions to decide on their equivalence.

In TSC, we compare test suites by extracting and comparing the test cases which are present in them. The reason for extracting test cases is that different developers code test suites (or test drivers) in different ways which makes the comparison of test suites complicated. We then replace each test case by its equivalence class, which results in a set of equivalence classes for each test suite. The comparison of test suites is then performed by comparing these sets of equivalence classes. For a test suite $TS$, $EQ(TS)$ is used to denote the set of equivalence classes.
Our test suite ($TS$) consists of a set of test cases. We define the set of equivalence classes for a test suite as the set of equivalence classes of the test cases in the test suite. Two test suites are equivalent if the test cases in them belong to the same set of equivalence classes, i.e., $TS_1 \sim TS_2 \iff EQ(TS_1) = EQ(TS_2)$.

An overview of $TSC$ is shown in Figure 23. Solid arrows represent tasks performed by humans (manual or automated). Dotted arrows represent the information which is required to perform a task. Solid rectangles represent input, intermediate or final outputs of a task. $TSC$ consists of four main tasks which are represented by the dotted rectangles as follows:

Task A: Extracting test cases from test suites using tracing

First, we extract test cases from the test suites (artefact A) which are being compared. This task consists of the following subtasks:
- Subtask A$_1$: Instrument test suites (artefact B).
- Subtask A$_2$: Execute the instrumented test suites and capture traces (artefact C).
- Subtask A$_3$: Extract test cases (artefact D) from the traces.

In Chapter 3, we used Daikon to extract method execution traces to test whether the method calls happen as specified in sequence diagrams. In here, we need to extract the method calls, parameter values and expected values to extract test cases from a test suite. However, using Daikon we can only extract method calls and parameter values but not the expected values which are embedded in test cases. Therefore, we decided to instrument the test suites manually.

Task B: Devising equivalence classes

Second, we devise equivalence classes for the extracted test cases. Each equivalence class represents a group of test cases in a test suite. This task consists of the following subtasks:
- Subtask B$_1$: Define a criterion (artefact F) for ECP of the test cases.
- Subtask B$_2$: Devise a set of equivalence classes (artefact G) using the criterion.

Task C: Transforming test suites into sets of equivalence classes

Third, we associate each test case to an equivalence class which results into a set of equivalence classes for each test suite. These sets of equivalence classes (artefact H) are the representatives of the test suites.
Task A. Extract test cases

A. Test Suites

Subtask A1. Instrument test suites

B. Instrumented Test Suites

Subtask A2. Capture execution traces

C. Execution Traces of Test Suites

Subtask A3. Extract test cases

D. Test Cases of Test Suites

Task B. Devise equivalence classes

E. System Under Test

Subtask B1: Define criteria for equivalence class partitioning

F. Criteria for Equivalence Class Partitioning

Subtask B2: Devise equivalence classes

G. Equivalence Classes

Task C. Transform test suites into sets of equivalence classes

H. Equivalence Class Representation of Test Suites

Task D. Compare sets of equivalence classes

I. Gaps in Test Suites

Figure 23: Overview of TestSuiteComparator
Task D: Comparing test suites using their representative sets of equivalence classes

Finally, we compare the sets of equivalence classes (artefact H) to compare the test suites (artefact A). This comparison identifies gaps (artefact I) in the test suites.

Task A is performed manually but it can be automated using some tracing tool such as Daikon. Task B is performed manually by the tester as it requires judgment and expertise. Task C is performed manually but it can be automated if the extracted test cases are represented in a structured format. Task D is automated as it involves a simple set comparison.

4.3. Case Studies

4.3.1. Case Study: Determining applicability of TestSuiteComparator

4.3.1.1. Introduction

The objective of this case study is to demonstrate the applicability (usability) of TSC for comparing test suites.

4.3.1.2. Object description

In this case study, we use an implementation of Dijkstra’s shortest-path algorithm [147]. This algorithm computes a shortest path from a node (vertex) of a graph to all other nodes of the graph. The implementation is provided in Appendix C.1.

The algorithm finds a shortest path for directed graphs with non-negative weights. The shortest path is defined as the path with minimum cost (weight). If there are two or more paths with the same cost, the one having the least number of edges is taken as the shortest path. If there are two or more paths with the same cost and the same number of edges, any of them can be returned as the shortest path.

The interface of the program consists of the following three methods:

1. The addEdge method is used to construct the graph. It takes three parameters: the source node, the destination node and the cost of the edge. If the source and destination nodes are not created already, they are added to the graph. It then adds the edge to the graph even if an edge already exists between the nodes, i.e. this implementation allows multiple edges between two nodes. The multiple edges can
represent the cost from the source node to the destination node using a certain medium such as cost of travel using a car. This method has the following signature:

\[
\text{void addEdge(String sourceName, String destName, double cost)}
\]

2. The \textit{dijkstra} method computes shortest paths from a particular node (source node) to all other nodes. If the source node does not exist in the graph, it throws \textit{NoSuchElementException}. If a negative weight edge exists in the graph, it throws \textit{GraphException}. This method has the following signature:

\[
\text{public void dijkstra(String startName)}
\]

3. The \textit{getPath} method returns the shortest path, from a source node to the destination node, as a string concatenation of nodes separated by a whitespace character (e.g. “A B C”). It returns “0” if the source and destination nodes are the same, and “-1” if the source and destination nodes are not connected. This poses the following restrictions on the names of nodes:

i. A node cannot be named “0” or “-1”.

ii. A node cannot have a space in its name.

This method requires that Dijkstra’s algorithm has been applied on the source node. Otherwise it returns “-1”. This method has the following signature:

\[
\text{public String getPath(String destination)}
\]

4.3.1.3. Experimental planning

We demonstrate the capability of \textit{TSC} to differentiate between test suites. We execute the following steps during the case study:

1. Devise test suites to be compared
   - We devise three test suites to test the implementation of Dijkstra’s algorithm. As the implementation is Java-based, these test suites are coded in JUnit. These test suites are provided in Appendix C.2.
   - We chose to devise three test suites so that we had three pairwise comparisons to evaluate \textit{TSC}.
   - Different testers (postgraduate students) were asked to develop these test suites so that they may vary in structure and style in order to imitate a real-world scenario in which different people code the same test cases in different manners. The diversity in thinking and coding styles of testers may highlight non-trivial challenges of comparing test suites.
2. Apply TSC
3. Analyse the results of the comparison

4.3.1.4. Applying TestSuiteComparator to the case study

The application of TSC is illustrated in this section. We perform the four tasks of TSC over the three test suites and three pairwise comparisons, and show the intermediate and final outputs.

Task A: Extract test cases (from the test suites) using tracing

We extract test cases from the concrete and executable test suites using tracing. For this purpose, we instrument the test suites and execute them to capture execution traces. The traces are shown in Appendix C.3. The test cases, which are extracted from the traces, are shown in Table 10.

In this case study, the description of a graph (i.e. a set of edges along with their weights), the source and the destination nodes are the inputs, and the shortest path is the expected result of a test case.

Task B: Devising equivalence classes

We devise equivalence classes using a black-box testing approach. Black-box testing is a technique in which we ignore the internal mechanism of a system (or component) and focus on the outputs generated in response to selected inputs and executing conditions [148].

For this purpose, we define a criterion for partitioning of test cases, and devise equivalence classes using this criterion (Sub-Tasks B1 and B2 in Figure 23). We partition the test cases at three levels:

1. Level 1 (L1): The number of nodes in the graph
2. Level 2 (L2): The number of edges on the shortest path
3. Level 3 (L3): The presence of paths other than the shortest path (i.e. multiple paths exist between source and destination nodes)

We then devise equivalence classes for these levels. We denote a path from node A to node B as Path_{A\rightarrow B}. Similarly, Edge_{A\rightarrow B} denotes an edge from node A to node B.
### Test Suite 1

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Graph</th>
<th>Source Node</th>
<th>Destination Node</th>
<th>Shortest Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>A B 2</td>
<td>A</td>
<td>B</td>
<td>AB</td>
</tr>
<tr>
<td>T2</td>
<td>A B 6</td>
<td>A</td>
<td>B</td>
<td>AB</td>
</tr>
<tr>
<td>T3</td>
<td>B C 1</td>
<td>C</td>
<td>A</td>
<td>ABC</td>
</tr>
<tr>
<td>T4</td>
<td>A B 6</td>
<td>A</td>
<td>B</td>
<td>AB</td>
</tr>
<tr>
<td>T5</td>
<td>B C 1</td>
<td>C</td>
<td>A</td>
<td>ABC</td>
</tr>
<tr>
<td>T6</td>
<td>A B 6</td>
<td>A</td>
<td>B</td>
<td>AB</td>
</tr>
<tr>
<td>T7</td>
<td>B C 1</td>
<td>C</td>
<td>A</td>
<td>ABC</td>
</tr>
<tr>
<td>T8</td>
<td>A B 5</td>
<td>A</td>
<td>B</td>
<td>AB</td>
</tr>
<tr>
<td>T9</td>
<td>B C 4</td>
<td>A</td>
<td>B</td>
<td>AB</td>
</tr>
<tr>
<td>T10</td>
<td>C D 2</td>
<td>C, D, AD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T11</td>
<td>A B 6</td>
<td>A</td>
<td>B</td>
<td>ACB</td>
</tr>
<tr>
<td>T12</td>
<td>B C 2</td>
<td>B</td>
<td>A</td>
<td>AB</td>
</tr>
<tr>
<td>T13</td>
<td>B D 1</td>
<td>C</td>
<td>B</td>
<td>AC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>

### Test Suite 2

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Graph</th>
<th>Source Node</th>
<th>Destination Node</th>
<th>Shortest Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>A B 2</td>
<td>A</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>T2</td>
<td>A B 3</td>
<td>A</td>
<td>B</td>
<td>AB</td>
</tr>
<tr>
<td>T3</td>
<td>B C 1</td>
<td>B</td>
<td>A</td>
<td>BA</td>
</tr>
<tr>
<td>T4</td>
<td></td>
<td>B</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>A B 2</td>
<td>A</td>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>T6</td>
<td>B C 2</td>
<td>B</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>T7</td>
<td>A B 6</td>
<td>C</td>
<td>A</td>
<td>AC</td>
</tr>
<tr>
<td>T8</td>
<td>B C 6</td>
<td>B</td>
<td>A</td>
<td>-1</td>
</tr>
<tr>
<td>T9</td>
<td>B C 4</td>
<td>B</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>T10</td>
<td>B C 4</td>
<td>C</td>
<td>B</td>
<td>BC</td>
</tr>
<tr>
<td>T11</td>
<td>B C 4</td>
<td>C</td>
<td>A</td>
<td>-1</td>
</tr>
<tr>
<td>T12</td>
<td>B C 4</td>
<td>C</td>
<td>B</td>
<td>-1</td>
</tr>
<tr>
<td>T13</td>
<td></td>
<td>C</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>T14</td>
<td>A B 2</td>
<td>A</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>T15</td>
<td>B C 4</td>
<td>A</td>
<td>B</td>
<td>AB</td>
</tr>
<tr>
<td>T16</td>
<td>A C 8</td>
<td>A</td>
<td>B</td>
<td>ABC</td>
</tr>
<tr>
<td>T17</td>
<td></td>
<td>A</td>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>T18</td>
<td></td>
<td>B</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>T19</td>
<td></td>
<td>B</td>
<td>C</td>
<td>BC</td>
</tr>
<tr>
<td>T20</td>
<td></td>
<td>B</td>
<td>C</td>
<td>-1</td>
</tr>
<tr>
<td>T21</td>
<td></td>
<td>B</td>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>T22</td>
<td></td>
<td>C</td>
<td>B</td>
<td>-1</td>
</tr>
<tr>
<td>T23</td>
<td></td>
<td>C</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>T24</td>
<td></td>
<td>D</td>
<td>C</td>
<td>CD</td>
</tr>
<tr>
<td>T25</td>
<td></td>
<td>D</td>
<td>A</td>
<td>-1</td>
</tr>
<tr>
<td>T26</td>
<td></td>
<td>D</td>
<td>B</td>
<td>-1</td>
</tr>
<tr>
<td>T27</td>
<td></td>
<td>D</td>
<td>C</td>
<td>-1</td>
</tr>
<tr>
<td>T28</td>
<td></td>
<td>D</td>
<td>D</td>
<td>0</td>
</tr>
</tbody>
</table>

### Test Suite 3

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Graph</th>
<th>Source Node</th>
<th>Destination Node</th>
<th>Shortest Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>A B 6</td>
<td>A</td>
<td>B</td>
<td>AB</td>
</tr>
<tr>
<td>T2</td>
<td>A B 2</td>
<td>A</td>
<td>B</td>
<td>AB</td>
</tr>
<tr>
<td>T3</td>
<td>B C 1</td>
<td>C</td>
<td>A</td>
<td>ABC</td>
</tr>
<tr>
<td>T4</td>
<td>A B 9</td>
<td>A</td>
<td>B</td>
<td>AB</td>
</tr>
<tr>
<td>T5</td>
<td>B C 1</td>
<td>C</td>
<td>A</td>
<td>AC</td>
</tr>
<tr>
<td>T6</td>
<td>A B 1</td>
<td>A</td>
<td>B</td>
<td>AB</td>
</tr>
<tr>
<td>T7</td>
<td>B C 1</td>
<td>C</td>
<td>A</td>
<td>ABC</td>
</tr>
<tr>
<td>T8</td>
<td>A B 8</td>
<td>A</td>
<td>B</td>
<td>AB</td>
</tr>
<tr>
<td>T9</td>
<td>B C 2</td>
<td>C</td>
<td>A</td>
<td>ABC</td>
</tr>
<tr>
<td>T10</td>
<td>C D 7</td>
<td>D</td>
<td>C</td>
<td>ACD</td>
</tr>
<tr>
<td>T11</td>
<td>A B 1</td>
<td>A</td>
<td>B</td>
<td>AB</td>
</tr>
<tr>
<td>T12</td>
<td>B C 2</td>
<td>C</td>
<td>A</td>
<td>ABC</td>
</tr>
<tr>
<td>T13</td>
<td>C D 7</td>
<td>D</td>
<td>A</td>
<td>ABCD</td>
</tr>
<tr>
<td>T14</td>
<td>A B 8</td>
<td>A</td>
<td>B</td>
<td>AB</td>
</tr>
<tr>
<td>T15</td>
<td>B C 2</td>
<td>C</td>
<td>A</td>
<td>AC</td>
</tr>
<tr>
<td>T16</td>
<td>C D 7</td>
<td>D</td>
<td>A</td>
<td>ACD</td>
</tr>
<tr>
<td>T17</td>
<td>A B 2</td>
<td>A</td>
<td>B</td>
<td>AB</td>
</tr>
<tr>
<td>T18</td>
<td>B C 2</td>
<td>C</td>
<td>A</td>
<td>AC</td>
</tr>
<tr>
<td>T19</td>
<td>C D 7</td>
<td>D</td>
<td>A</td>
<td>ABD</td>
</tr>
<tr>
<td>T20</td>
<td>A B 1</td>
<td>A</td>
<td>B</td>
<td>AB</td>
</tr>
<tr>
<td>T21</td>
<td>B C 2</td>
<td>C</td>
<td>A</td>
<td>AC</td>
</tr>
<tr>
<td>T22</td>
<td>C B 1</td>
<td>A</td>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>T23</td>
<td>B C 2</td>
<td>A</td>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>T24</td>
<td>B C 2</td>
<td>A</td>
<td>D</td>
<td>0</td>
</tr>
<tr>
<td>T25</td>
<td>B C 2</td>
<td>A</td>
<td>D</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table 10:** Test cases extracted from test suites for shortest-path case study
In L1, graphs with four or more nodes are put into the same category (equivalence class). We believe four vertices are enough to present scenarios which are interesting from a testing perspective, such as a path from one node to another containing multiple edges. Therefore, the L1 equivalence classes are the following:

1. Graphs with zero nodes
2. Graphs with one node
3. Graphs with two nodes
4. Graphs with three nodes
5. Graphs with four or more nodes

In L2, shortest paths containing three or more edges are put into the same category (equivalence class). We believe three edges are enough to present interesting scenarios from a testing perspective, such as paths from one node to another containing one, two or more edges. Therefore, the L2 equivalence classes are the following:

1. The shortest path contains zero edges, i.e. the source and destination nodes are the same
2. The shortest path contains one edge
3. The shortest path contains two edges
4. The shortest path contains three or more edges

In L3, equivalence classes are devised based on the presence of paths, between the source and destination nodes, other than the shortest path. This classification represents the scenarios in which the program finds the shortest path in the presence of multiple paths between the two nodes. As noted before, the shortest path is the one having the lowest cost and not the one having the least number of edges. The notation “Path_{A \rightarrow B} = \text{Edge}_{A \rightarrow B} + \text{Edge}_{B \rightarrow C}” is used to indicate that there is a path from node A to node B consisting of the edges AB and BC. The L3 equivalence classes are the following:

1. Source and destination nodes are the same (Path_{A \rightarrow A} = “0”)
2. No path exists between the source and the destination nodes (Path_{A \rightarrow B} = “-1”)
3. The shortest path is the only path that exists between the two nodes
4. There exists a path between the two nodes (which is not the shortest one) with more edges than the shortest path. E.g.,

$$Path_{A \rightarrow B} = \text{Edge}_{A \rightarrow B} \text{ (the shortest)}$$

$$Path_{A \rightarrow B} = \text{Edge}_{A \rightarrow C} + \text{Edge}_{C \rightarrow B} \text{ (not the shortest path)}$$
5. There exists a path between the two nodes (which is not the shortest one) with fewer edges than the shortest path. E.g.,

\[ Path_{A \rightarrow B} = Edge_{A \rightarrow C} + Edge_{C \rightarrow B} \] (the shortest path)
\[ Path_{A \rightarrow B} = Edge_{A \rightarrow B} \] (not the shortest path)

6. There exists a path between the two nodes (which is not the shortest one) with fewer edges than the shortest path, and a path with more edges than the shortest path. E.g.,

\[ Path_{A \rightarrow B} = Edge_{A \rightarrow C} + Edge_{C \rightarrow B} \] (the shortest path)
\[ Path_{A \rightarrow B} = Edge_{A \rightarrow B} \] (not the shortest path)
\[ Path_{A \rightarrow B} = Edge_{A \rightarrow C} + Edge_{C \rightarrow D} + Edge_{D \rightarrow B} \] (not the shortest path)

7. There exists a path between the two nodes that has the same cost as the shortest path but that has more edges than the shortest path. E.g.,

\[ Path_{A \rightarrow B} = Edge_{A \rightarrow B} \] (the shortest)
\[ Path_{A \rightarrow B} = Edge_{A \rightarrow C} + Edge_{C \rightarrow B} \] (not the shortest path)

Note: We have defined the shortest path as the path with minimum cost and the minimum number of edges (Section 4.3.1.2)

We devise a final set of equivalence classes by combining the equivalence classes which are devised for each level i.e. L1,L2,L3. For example, the equivalence class 2.1.1 is obtained by combining the partition of L1 which has two vertices (i.e. graphs containing two nodes), the partition of L2 which has one edge (i.e. graphs containing one edge), and the first partition of L3 (i.e. the source and destination nodes are the same). A complete list of the final equivalence classes devised by combining these partitions is provided in Appendix C.4. These equivalence classes represent sets of test cases.

We have devised equivalence classes using the nodes and edges of the graph. This is only one way of devising equivalence classes. However, testers can choose any other criterion they like to devise equivalence class while applying TSC.

Task C: Transform test suites into sets of equivalence classes

We assign an equivalence class to each of the test cases in the test suites. The assignment of equivalence classes is shown in Table 11.

We then transform each test suite into a set of equivalence classes. These sets of equivalence classes are shown in Table 12. \( EQ(TestSuite_1) \), \( EQ(TestSuite_2) \), and...
EQ(TestSuite_3) are the sets of equivalence classes for TestSuite_1, TestSuite_2 and TestSuite_3 respectively.

<table>
<thead>
<tr>
<th>Test Suite 1</th>
<th>Test Suite 2</th>
<th>Test Suite 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case</td>
<td>Equivalence Class</td>
<td>Test Case</td>
</tr>
<tr>
<td>T1</td>
<td>2.1.1</td>
<td>T1</td>
</tr>
<tr>
<td>T2</td>
<td>3.1.1</td>
<td>T2</td>
</tr>
<tr>
<td>T3</td>
<td>3.2.1</td>
<td>T3</td>
</tr>
<tr>
<td>T4</td>
<td>3.1.1</td>
<td>T4</td>
</tr>
<tr>
<td>T5</td>
<td>3.2.1</td>
<td>T5</td>
</tr>
<tr>
<td>T6</td>
<td>3.1.1</td>
<td>T6</td>
</tr>
<tr>
<td>T7</td>
<td>3.2.2</td>
<td>T7</td>
</tr>
<tr>
<td>T8</td>
<td>4.1.1</td>
<td>T8</td>
</tr>
<tr>
<td>T9</td>
<td>4.2.1</td>
<td>T9</td>
</tr>
<tr>
<td>T10</td>
<td>4.3.1</td>
<td>T10</td>
</tr>
<tr>
<td>T11</td>
<td>4.2.4</td>
<td>T11</td>
</tr>
<tr>
<td>T12</td>
<td>4.1.1</td>
<td>T12</td>
</tr>
<tr>
<td>T13</td>
<td>4.2.3</td>
<td>T13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T21</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T27</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T28</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T29</td>
</tr>
</tbody>
</table>

Table 11: Test cases and their equivalence classes for shortest-path case study

<table>
<thead>
<tr>
<th>Test Suite</th>
<th>Equivalence Class Representation</th>
<th>Equivalence Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestSuite_1</td>
<td>EQ(TestSuite_1)</td>
<td>{ 2.1.1, 3.1.1, 3.2.1, 3.2.2, 4.1.1, 4.2.1, 4.2.3, 4.2.4, 4.3.1 }</td>
</tr>
<tr>
<td>TestSuite_2</td>
<td>EQ(TestSuite_2)</td>
<td>{ 2.0.1, 2.1.1, 3.0.2, 3.0.1, 3.1.1, 3.2.2, 4.0.1, 4.0.2, 4.1.1, 4.1.2, 4.2.3, 4.2.4 }</td>
</tr>
<tr>
<td>TestSuite_3</td>
<td>EQ(TestSuite_3)</td>
<td>{ 2.1.1, 3.1.1, 3.1.3, 3.2.1, 4.1.1, 4.1.2, 4.2.1, 4.2.4, 4.3.1 }</td>
</tr>
</tbody>
</table>

Table 12: Equivalence-class representation of test suites for shortest-path case study
Task D: Comparing test suites using their representative sets of equivalence classes

We compare the two sets of equivalence classes to compare the test suites. A comparison of TestSuite_1, TestSuite_2 and TestSuite_3 is shown in Table 13. This table shows the set differences when the sets in the columns are subtracted from the sets in the rows. For example, the third column of the second row shows $EQ(\text{TestSuite}_1) - EQ(\text{TestSuite}_2)$.

The set difference of $EQ(\text{TestSuite}_1)$ and $EQ(\text{TestSuite}_2)$ shows the test cases that are present in TestSuite_1, but missing in TestSuite_2. Similarly, the set difference of $EQ(\text{TestSuite}_2)$ and $EQ(\text{TestSuite}_1)$ shows the test cases that are present in TestSuite_2, but missing in TestSuite_1. The two test suites are equivalent if both the difference sets are empty.

<table>
<thead>
<tr>
<th>Set Difference</th>
<th>EQ (TestSuite_1)</th>
<th>EQ (TestSuite_2)</th>
<th>EQ (TestSuite_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ (TestSuite_1)</td>
<td>$\emptyset$</td>
<td>${3.2.1, 4.2.1, 4.3.1}$</td>
<td>${3.2.2, 4.2.3}$</td>
</tr>
<tr>
<td>EQ (TestSuite_2)</td>
<td>${2.0.1, 3.0.2, 3.0.1,$</td>
<td>$\emptyset$</td>
<td>${2.1.1, 3.1.1, 3.2.2,$</td>
</tr>
<tr>
<td></td>
<td>$4.0.1, 4.0.2, 4.1.2}$</td>
<td></td>
<td>$4.0.1, 4.0.2, 4.2.3}$</td>
</tr>
<tr>
<td>EQ (TestSuite_3)</td>
<td>${3.1.3, 4.1.2}$</td>
<td>${3.1.3, 3.2.1, 4.2.1, 4.3.1}$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

Table 13: Comparison of test suites for shortest-path case study

4.3.1.5. Data interpretation

The results of the comparison (shown in Table 13) show that TSC has differentiated between the test suites. The test suites TestSuite_1, TestSuite_2 and TestSuite_3 have 13, 29 and 22 test cases respectively. Their representative sets of equivalence classes have 9, 12 and 9 equivalence classes. The difference of $EQ(\text{TestSuite}_2)$ and $EQ(\text{TestSuite}_1)$ indicates gaps in TestSuite_2 compared to TestSuite_1. These gaps are due to the test cases of TestSuite_2 missing in TestSuite_1. These gaps show that TestSuite_1 does not have any test case which executes the following scenarios (Appendix C.4):

1. The graph has 2 nodes, and the source and destination nodes are the same (equivalence class 2.0.1).
2. The graph has 3 nodes, and the source and destination nodes are the same (equivalence class 3.0.1).
3. The graph has 3 nodes, and no path exists between the source and destination nodes (equivalence class 3.0.2).
4. The graph has 4 nodes, and the source and destination nodes are the same (The graph class 4.0.1).
5. The graph has 4 nodes, and no path exists between the source and destination nodes (equivalence class 4.0.2).
6. The graph has 4 nodes, the shortest path has one edge, and there is only one path exists between the source and destination nodes (equivalence class 4.1.2).

Similarly, the difference of $EQ(\text{TestSuite}_2)$ and $EQ(\text{TestSuite}_1)$ shows gaps in $\text{TestSuite}_1$ compared to the $\text{TestSuite}_2$. These gaps show $\text{TestSuite}_2$ has no test case which executes the following scenarios:

1. The graph has 3 nodes, the shortest path has 2 edges, and there is only one path from source to destination node (equivalence class 3.2.1).
2. The graph has 4 nodes, the shortest path has 2 edges, and there is only one path from source to destination node (equivalence class 4.2.1).
3. The graph has 4 nodes, the shortest path has 3 edges, and there is only one path from source to destination node (equivalence class 4.3.1).

The gaps identified in the test suites show the ability of $TSC$ to differentiate test suites. Further, these gaps can be used to enrich the test suites.

4.3.1.6. Discussion

$TSC$ can be applied to differentiate between two test suites for the same implementation. To demonstrate its applicability, we have applied it to three test suites that were developed by different testers. $TSC$ has differentiated these test suites (according to the specified criterion) and highlighted gaps in each of these test suites. This information can be used to extend the test adequacy of $\text{TestSuite}_1$ by adding one test case (of $\text{TestSuite}_2$) which belongs to the equivalence class that is present in the set difference of $EQ(\text{TestSuite}_2)$ and $EQ(\text{TestSuite}_1)$.

$TSC$ is general in two aspects: language independency and criteria (for devising equivalence classes) independency. Its language-independent nature supports the comparison of test suites written in any unit testing framework, such as JUnit or CUnit. The method also allows for plugging-in different criteria for devising equivalence classes, e.g., the equivalence classes devised using a black-box approach, white-box approach [148] or a combination of these approaches.
The case study used to determine the applicability of the TSC is small and simple. We shall use a different case study, which is larger and more complex, for validating the effectiveness of the method. The effectiveness of the method can be measured by using coverage analysis or mutation analysis. These analyses will be performed in the next case study in which we shall evaluate the effectiveness of the method.

4.3.1.7. Threats to validity

As our objective was to demonstrate the viability of TSC, we consider the threats to the applicability of TSC to other systems and applications. This case study has the following threats:

1. We have applied TSC to a small number of test suites (i.e. three test suites).
2. We have applied TSC to the test suites which are small in size. Therefore, this case study does not demonstrate the ability of TSC to differentiate large test suites.

4.3.2. Case Study: Evaluating effectiveness of TestSuiteComparator

4.3.2.1. Introduction

The objective of this case study is to determine the effectiveness of TSC using the following two techniques: coverage analysis and mutation analysis. Our hypothesis is that TSC can be effectively applied to compare test suites.

We have used the shortest-path case study to determine applicability of TSC in Section 4.3.1. We could have used the same case study for evaluating the effectiveness of TSC. However, we chose to use a different case study for evaluation of the framework for the following two reasons: i) it does not make sense to use the same shortest-path case study (that was used in the development of a method/tool) to evaluate that same method/tool, and ii) the use of a different case study provides further evidence of the applicability of TSC other than evaluating the effectiveness of TSC.

Coverage analysis is a popular and effective technique to determine test adequacy [149-155]. With coverage analysis, we measure the coverage of each test suite.

Mutation analysis, which was proposed by Hamlet [156] and DeMillo et al. [157-159], is another widely used technique to measure the quality of test suites [32, 64, 160-165]. With mutation analysis, we seed some defects in the implementation of a program to create mutants (faulty implementations), execute test suites over the mutants, and measure the
number of mutants killed by each test suite. As the prime objective of a test suite is to detect
defects, the mutant-killing capability can be used as an indication of the quality of a test
suite.

We then see if the results of these analyses support the results of our comparison, i.e.,
does the test suite that has smaller gaps, give better coverage and kill more mutants?

As our motivation for devising TSC is to evaluate and extend the test adequacy of
component testing, we take effectiveness of TSC as the increase in test adequacy of a test
suite (using the gaps identified by TSC). For coverage analysis, effectiveness refers to the
increase in coverage of the extended test suite. For mutation analysis, effectiveness refers to
the increase in the number of mutants killed by the extended test suite.

4.3.2.2. Object description

In this case study, we use an implementation of the Boyer-Moore pattern-matching algorithm
[166]. It searches for a pattern in a string and returns the index of the first occurrence of the
pattern in the string. The implementation is provided in Appendix C.5.

Pattern matching algorithms address the problem of finding occurrence(s) of a pattern
(a string) within another string. National Institute of Standards and Technology (NIST)
defines a pattern as a finite number of strings that are searched for in another string [167].
Boyer and Moore have devised a fast algorithm for finding patterns in a string.

The interface of the program consists of:

1. Instantiation of the BoyerMoore object using its constructor which takes the
   following two parameters of type String:
      i. The pattern to be found.
      ii. The text in which to search.

2. Invocation of the match method on the BoyerMoore object. This method returns an
   integer as a result.
      i. If the pattern is found (occurs in the string), it returns the index of the first
         occurrence of the pattern in the string.
      ii. If the pattern is not found, it returns -1.

If the string and the pattern are the same, we assume that the pattern is at the beginning of the
string. If the pattern is an empty string, we assume that the pattern does not occur in the
string. If the pattern occurs at multiple places in the string, the implementation of Boyer-Moore’s algorithm finds the first occurrence of the pattern.

The motivations for selecting this case study are the following:
1. The pattern-matching problem is simple to understand, and testers can develop test suites without having to understand the details of the algorithm.
2. The implementation is small in size (98 lines of code) and yet complex enough for testing. Its source code consists of four methods, eight if-else statements and ten loops (for and while loops).

4.3.2.3. Experimental planning

During this case study, we measure the following variables:
1. The number of statements in the code
2. Statements executed by the original test suites
3. Statements executed by the extended test suites
4. The number of defects seeded in the code
5. Mutants killed by the original test suites
6. Mutants killed by the extended test suites

We execute the following steps during the case study:
1. Devise test suites to be compared
   - We devise three test suites to test the implementation of Boyer-Moore’s algorithm. As the implementation is Java-based, these test suites are coded in JUnit. These test suites are provided in Appendix C.6.
   - We chose to devise three test suites because we can perform three pairwise comparisons, and enrich each test suite in three ways for this case study. This data can be used to determine the effectiveness of TSC.
   - These test suites are devised by different testers (postgraduate students) so that they may vary in structure and style in order to imitate a real-world scenario.
   Apply TSC using a black-box approach
2. Extend the test suites using the gaps identified by TSC
   A. Extend TestSuite_1
   B. Extend TestSuite_2
   C. Extend TestSuite_3
3. Validate the effectiveness of TSC using a white-box criterion (statement coverage)
   A. Measure statement coverage of the original test suites
   B. Measure statement coverage of the extended test suites

4. Validate the effectiveness of TSC using a mutation-based criterion
   A. Seed defects in the code to come up with different program mutants
   B. Identify the mutants killed by the original test suites
   C. Identify the mutants killed by the extended test suites

5. Analyse the usefulness of TSC
   A. Determine TSC’s ability to identify gaps in the test suites, and to extend them
   B. Determine the increase in the statement coverage of the extended test suites
   C. Determine the increase in the number of mutants killed by the extended test suites

4.3.2.4. Applying TestSuiteComparator to the case study

The execution of the four tasks of TSC over the three test suites is illustrated along with the intermediate and final outputs.

**Task A: Extract test cases (from the test suites) using tracing**

We extract test cases from the test suites using tracing. For this purpose, we instrument the test suites and execute them to capture their execution traces. These execution traces are shown in Appendix C.7. From these traces, the test cases are extracted which are shown in Table 14.

In this case study, a pattern (which is being searched for) and a string (text in which the pattern is sought) are the inputs, and an integer representing the occurrence of the pattern is the expected result of a test case.

**Task B: Devising equivalence classes**

We devise equivalence classes using a black-box approach. For this purpose, we define a criterion for partitioning of test cases, and devise equivalence classes using this criterion (Sub-Tasks B1 and B2 in Figure 23). We partition the test cases at three levels:

1. Level 1 (L1): The number of characters that the string contains.
2. Level 2 (L2): The number of characters that the pattern contains.
3. Level 3 (L3): The occurrence (and location) of the pattern in the string.
We then devise equivalence classes for these levels. We denote the length of the string and the pattern by $S_{\text{LENGTH}}$ and $P_{\text{LENGTH}}$ respectively.

<table>
<thead>
<tr>
<th>Test Suite_1</th>
<th>Test Suite_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC</td>
<td>String</td>
</tr>
<tr>
<td>T1</td>
<td>&quot;&quot;&quot;&quot;</td>
</tr>
<tr>
<td>T2</td>
<td>&quot;&quot;&quot;&quot;</td>
</tr>
<tr>
<td>T3</td>
<td>A</td>
</tr>
<tr>
<td>T4</td>
<td>A</td>
</tr>
<tr>
<td>T5</td>
<td>A</td>
</tr>
<tr>
<td>T6</td>
<td>Aa</td>
</tr>
<tr>
<td>T7</td>
<td>Aa</td>
</tr>
<tr>
<td>T8</td>
<td>Aa</td>
</tr>
<tr>
<td>T9</td>
<td>Aa</td>
</tr>
<tr>
<td>T10</td>
<td>A</td>
</tr>
<tr>
<td>T11</td>
<td>A&lt;sp&gt;</td>
</tr>
<tr>
<td>T12</td>
<td>&lt;sp&gt;A</td>
</tr>
<tr>
<td>T13</td>
<td>Aaa</td>
</tr>
<tr>
<td>T14</td>
<td>Aaa</td>
</tr>
<tr>
<td>T15</td>
<td>Aaa</td>
</tr>
<tr>
<td>T16</td>
<td>Aaa</td>
</tr>
<tr>
<td>T17</td>
<td>A&lt;sp&gt;</td>
</tr>
<tr>
<td>T18</td>
<td>Aa&lt;sp&gt;</td>
</tr>
<tr>
<td>T19</td>
<td>abcd</td>
</tr>
<tr>
<td>T20</td>
<td>ababababab</td>
</tr>
<tr>
<td>T21</td>
<td>abcd</td>
</tr>
<tr>
<td>T23</td>
<td>abcd</td>
</tr>
<tr>
<td>T25</td>
<td>Abcd</td>
</tr>
</tbody>
</table>

**Legend:** TC, Exp, and <sp> denote Test Case, Expected Result, and a space character respectively.

**Table 14:** Test cases extracted from test suites for pattern-matching case study

In $L_1$, we devise the equivalence classes based on the length of the string. The strings with four or more characters are put into the same category (equivalence class). Therefore, the $L_1$ equivalence classes are the following:

1. Empty string ($S_{\text{LENGTH}} = 0$).
2. Strings that contain one character ($S_{\text{LENGTH}} = 1$).
3. Strings that contain two characters \((S_{\text{LENGTH}} = 2)\).
4. Strings that contain three or more characters \((S_{\text{LENGTH}} = 3)\).
5. Strings that contain three or more characters \((S_{\text{LENGTH}} \geq 4)\).

In L2, we devise the equivalence classes based on the length of the pattern. The patterns with four or more characters are put into the same category (equivalence class). Therefore, the L2 equivalence classes are the following:

1. Pattern that is an empty string \((P_{\text{LENGTH}} = 0)\).
2. Patterns that contain one character \((P_{\text{LENGTH}} = 1)\).
3. Patterns that contain one character \((P_{\text{LENGTH}} = 2)\).
4. Patterns that contain one character \((P_{\text{LENGTH}} = 3)\).
5. Patterns that contain more characters than the string \((P_{\text{LENGTH}} \geq 4)\).

In L3, we devise equivalence classes to group testing scenarios based on the occurrence of the pattern (i.e. beginning, middle and end), case sensitivity of the pattern, etc. The L3 equivalence classes are the following:

1. The pattern occurs neither with actual case nor with a different case. Here, “actual case” and “different case” represent case-sensitive and case-insensitive comparisons respectively. For example, the patterns \(AB\) and \(Ab\) in the string \(ABC\).
2. The pattern does not occur with actual case but occurs with a different case.
3. The pattern occurs at the beginning of the string (i.e. there is no character in the string before the occurrence of the pattern).
4. The pattern occurs at the end of the string (i.e. there is no character in the string after the occurrence of the pattern).
5. The pattern occurs in the middle of the string (i.e. there is at least one character in the string before and after the occurrence of the pattern).
6. The pattern occurs more than once (i.e. multiple occurrences of the pattern).

We devise a final set of equivalence classes by combining the equivalence classes which are devised for each level i.e. \(L1. L2. L3\). For example, the equivalence class \(2.1.3\) is obtained by combining the partition of L1 in which the strings contain two characters, the partition of L2 in which the patterns contain one character, and the third partition of L3 (i.e. the pattern occurs in the beginning of the string). A complete list of the final equivalence classes devised by combining these partitions is provided in Appendix C.8.
Task C: Transform test suites into sets of equivalence classes

We assign an equivalence class to each of the test cases in the test suites. The assignment of equivalence classes is shown in Table 15.

We then transform each test suite into a set of equivalence classes. These sets of equivalence classes are shown in Table 16. As noted before, \( EQ (TS) \) is used to denote a set of equivalence classes for the test suite \( TS \).

<table>
<thead>
<tr>
<th>Test Suite_1</th>
<th>Test Case</th>
<th>Equivalence Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>0.0.1</td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>0.1.1</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>1.0.1</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>1.1.2</td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>1.2.1</td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td>2.0.1</td>
<td></td>
</tr>
<tr>
<td>T7</td>
<td>2.1.4</td>
<td></td>
</tr>
<tr>
<td>T8</td>
<td>2.1.1</td>
<td></td>
</tr>
<tr>
<td>T9</td>
<td>2.2.3</td>
<td></td>
</tr>
<tr>
<td>T10</td>
<td>2.2.1</td>
<td></td>
</tr>
<tr>
<td>T11</td>
<td>2.2.3</td>
<td></td>
</tr>
<tr>
<td>T12</td>
<td>2.2.3</td>
<td></td>
</tr>
<tr>
<td>T13</td>
<td>3.0.1</td>
<td></td>
</tr>
<tr>
<td>T14</td>
<td>3.1.3</td>
<td></td>
</tr>
<tr>
<td>T15</td>
<td>3.2.3</td>
<td></td>
</tr>
<tr>
<td>T16</td>
<td>3.3.2</td>
<td></td>
</tr>
<tr>
<td>T17</td>
<td>3.1.5</td>
<td></td>
</tr>
<tr>
<td>T18</td>
<td>3.1.4</td>
<td></td>
</tr>
<tr>
<td>T19</td>
<td>4.1.1</td>
<td></td>
</tr>
<tr>
<td>T20</td>
<td>4.2.3</td>
<td></td>
</tr>
<tr>
<td>T21</td>
<td>4.4.3</td>
<td></td>
</tr>
<tr>
<td>T22</td>
<td>4.4.4</td>
<td></td>
</tr>
<tr>
<td>T23</td>
<td>4.4.4</td>
<td></td>
</tr>
<tr>
<td>T24</td>
<td>4.4.3</td>
<td></td>
</tr>
<tr>
<td>T25</td>
<td>4.4.1</td>
<td></td>
</tr>
<tr>
<td>T26</td>
<td>4.4.1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Suite_2</th>
<th>Test Case</th>
<th>Equivalence Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>4.3.2</td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>4.4.1</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>4.4.5</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>4.3.5</td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>4.3.1</td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td>4.2.2</td>
<td></td>
</tr>
<tr>
<td>T7</td>
<td>4.4.1</td>
<td></td>
</tr>
<tr>
<td>T8</td>
<td>4.2.1</td>
<td></td>
</tr>
<tr>
<td>T9</td>
<td>4.4.1</td>
<td></td>
</tr>
<tr>
<td>T10</td>
<td>4.3.1</td>
<td></td>
</tr>
<tr>
<td>T11</td>
<td>4.3.5</td>
<td></td>
</tr>
<tr>
<td>T12</td>
<td>4.3.1</td>
<td></td>
</tr>
<tr>
<td>T13</td>
<td>4.4.4</td>
<td></td>
</tr>
<tr>
<td>T14</td>
<td>4.0.1</td>
<td></td>
</tr>
<tr>
<td>T15</td>
<td>4.4.5</td>
<td></td>
</tr>
<tr>
<td>T16</td>
<td>0.0.1</td>
<td></td>
</tr>
<tr>
<td>T17</td>
<td>0.1.1</td>
<td></td>
</tr>
<tr>
<td>T18</td>
<td>0.3.1</td>
<td></td>
</tr>
<tr>
<td>T19</td>
<td>4.0.1</td>
<td></td>
</tr>
<tr>
<td>T20</td>
<td>4.4.3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Suite_3</th>
<th>Test Case</th>
<th>Equivalence Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>0.0.1</td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>4.0.1</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>4.1.5</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>4.3.1</td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>4.3.3</td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td>4.3.2</td>
<td></td>
</tr>
<tr>
<td>T7</td>
<td>4.4.5</td>
<td></td>
</tr>
<tr>
<td>T8</td>
<td>4.4.5</td>
<td></td>
</tr>
<tr>
<td>T9</td>
<td>4.4.1</td>
<td></td>
</tr>
<tr>
<td>T10</td>
<td>4.4.5</td>
<td></td>
</tr>
<tr>
<td>T11</td>
<td>4.4.4</td>
<td></td>
</tr>
<tr>
<td>T12</td>
<td>4.4.2</td>
<td></td>
</tr>
<tr>
<td>T13</td>
<td>4.4.1</td>
<td></td>
</tr>
<tr>
<td>T14</td>
<td>4.4.3</td>
<td></td>
</tr>
<tr>
<td>T15</td>
<td>4.4.3</td>
<td></td>
</tr>
<tr>
<td>T16</td>
<td>4.4.1</td>
<td></td>
</tr>
</tbody>
</table>

Table 15: Test cases and equivalence classes for pattern-matching case study

<table>
<thead>
<tr>
<th>Test Suite</th>
<th>Equivalence Class Representation</th>
<th>Equivalence Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestSuite_1</td>
<td>EQ ( TestSuite_1 )</td>
<td>{ 0.0.1, 0.1.1, 1.0.1, 1.1.2, 1.2.1, 2.0.1, 2.1.1, 2.1.4, 2.2.1, 2.2.3, 3.0.1, 3.1.3, 3.1.4, 3.1.5, 3.2.3, 3.3.2, 4.1.1, 4.2.3, 4.4.1, 4.4.3, 4.4.4 }</td>
</tr>
<tr>
<td>TestSuite_2</td>
<td>EQ ( TestSuite_2 )</td>
<td>{ 0.0.1, 0.1.1, 0.3.1, 4.0.1, 4.2.1, 4.3.1, 4.3.2, 4.3.5, 4.4.1, 4.4.3, 4.4.4, 4.4.5 }</td>
</tr>
</tbody>
</table>
Table 16: Equivalence-class representation of test suites for pattern-matching case study

Task D: Comparing test suites using their representative sets of equivalence classes

The comparison of TestSuite_1, TestSuite_2 and TestSuite_3 is shown in Table 17.

<table>
<thead>
<tr>
<th>Set Difference</th>
<th>EQ (TestSuite_1)</th>
<th>EQ (TestSuite_2)</th>
<th>EQ (TestSuite_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ (TestSuite_1)</td>
<td>Ø</td>
<td>{ 1.0.1, 1.1.2, 1.2.1, 2.0.1, 2.1.1, 2.1.4, 2.2.1, 2.2.3, 3.0.1, 3.1.3, 3.1.4, 3.1.5, 3.2.3, 3.3.2, 4.1.1, 4.2.3 }</td>
<td>{ 0.1.1, 1.0.1, 1.1.2, 1.2.1, 2.0.1, 2.1.1, 2.1.4, 2.2.1, 2.2.3, 3.0.1, 3.1.3, 3.1.4, 3.1.5, 3.2.3, 3.3.2, 4.1.1, 4.2.3 }</td>
</tr>
<tr>
<td>EQ (TestSuite_2)</td>
<td>{ 0.3.1, 4.0.1, 4.2.1, 4.2.2, 4.3.1, 4.3.2, 4.3.3, 4.3.5, 4.4.5 }</td>
<td>Ø</td>
<td>{ 0.1.1, 0.3.1, 4.2.1, 4.3.5 }</td>
</tr>
<tr>
<td>EQ (TestSuite_3)</td>
<td>{ 4.0.1, 4.1.5, 4.3.1, 4.3.2, 4.3.3, 4.4.2, 4.4.5 }</td>
<td>{ 4.1.5, 4.3.3, 4.4.2 }</td>
<td>Ø</td>
</tr>
</tbody>
</table>

Table 17: Comparison of test suites for pattern-matching case study

4.3.2.5. Extending the test suites using the gaps identified by TSC

We use the gaps identified by TSC to extend the test suites (shown in Table 18). As we have performed three pairwise comparisons, we can extend each test suite in three ways. We extend TestSuite_1 in the following ways:

1. Using the gaps identified by TestSuite_2
2. Using the gaps identified by TestSuite_3
3. Using the gaps identified by both TestSuite_2 and TestSuite_3

A subscript will be used to indicate a test suite which has been extended to cover gaps identified by comparison with other test suites, e.g. TestSuite_1_2 represents TestSuite_1 extended using the gaps identified by TestSuite_2. This is explained in detail below.
To extend \textit{TestSuite}_1 using \textit{TestSuite}_2, for each equivalence class in the set difference of $\text{EQ}(\textit{TestSuite}_2)$ and $\text{EQ}(\textit{TestSuite}_1)$, we find a test case in \textit{TestSuite}_2 that belongs to that equivalence class. For instance, we may select test cases $T1$, $T6$, $T8$, $T11$, $T12$, $T14$, $T15$ and $T18$ of \textit{TestSuite}_2 as our representatives of the equivalence classes 4.3.2, 4.2.2, 4.2.1, 4.3.5, 4.3.1, 4.0.1, 4.4.5 and 0.3.1 respectively. We then add these test cases to \textit{TestSuite}_1 to enrich it giving $\textit{TestSuite}_1_{G2}$. 

<table>
<thead>
<tr>
<th>Enriched Test Suites</th>
<th>Test Cases</th>
</tr>
</thead>
</table>
| **TestSuite_1G2**    | TestSuite_1  
                     U { T1, T6, T8, T11, T12, T14, T15, T18 } |
| **TestSuite_1G3**    | TestSuite_1  
                     U { T2, T3, T4, T5, T6, T7, T12 } |
| **TestSuite_1G2G3**  | TestSuite_1  
                     U { T1, T6, T8, T11, T12, T14, T15, T18 }  
                     U { T2, T3, T4, T5, T6, T7, T12 } |
| **TestSuite_2G1**    | TestSuite_2  
                     U { T3, T4, T5, T6, T7, T8, T10, T11, T13, T14, T15, T16, T17, T18, T19, T20 } |
| **TestSuite_2G3**    | TestSuite_2  
                     U { T3, T5, T12 } |
| **TestSuite_2G13**   | TestSuite_2  
                     U { T3, T4, T5, T6, T7, T8, T10, T11, T13, T14, T15, T16, T17, T18, T19, T20 }  
                     U { T3, T5, T12 } |
| **TestSuite_3G1**    | TestSuite_3  
                     U { T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T13, T14, T15, T17, T18, T19, T20 } |
| **TestSuite_3G2**    | TestSuite_3  
                     U { T8, T11, T17, T18 } |
| **TestSuite_3G12**   | TestSuite_3  
                     U { T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T13, T14, T15, T17, T18, T19, T20 }  
                     U { T8, T11, T17, T18 } |

**Table 18:** The enriched test suites for pattern-matching case study
<table>
<thead>
<tr>
<th>Test Suite</th>
<th>Statements Executed (SE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestSuite_1</td>
<td>{ 02 04 05 06 08 09 10 12 13 14 15 16 17 20 22 23 25 26 27 28 29 30 32 33 34 35 36 39 42 43 44 46 47 48 53 54 55 58 60 61 62 63 64 65 69 70 71 72 74 75 76 77 78 80 86 87 88 89 90 93 94 96 }</td>
</tr>
<tr>
<td>TestSuite_2</td>
<td>{ 02 04 05 06 08 09 10 12 13 14 15 16 17 20 22 23 25 26 27 28 29 30 32 33 34 35 36 39 42 43 44 46 47 48 53 54 55 58 60 61 62 63 64 65 69 86 87 88 89 90 91 92 93 94 96 }</td>
</tr>
<tr>
<td>TestSuite_3</td>
<td>{ 02 04 05 06 08 09 10 12 13 14 15 16 17 20 22 23 25 26 27 28 29 30 32 33 34 35 36 39 42 43 44 46 47 48 53 54 55 58 60 61 62 63 64 65 69 70 71 72 74 75 86 87 88 89 90 93 94 96 }</td>
</tr>
<tr>
<td>TestSuite_1G2</td>
<td>SE (TestSuite_1) U { 91 92 }</td>
</tr>
<tr>
<td>TestSuite_1G3</td>
<td>SE (TestSuite_1)</td>
</tr>
<tr>
<td>TestSuite_1G33</td>
<td>SE (TestSuite_1) U { 91 92 }</td>
</tr>
<tr>
<td>TestSuite_2G1</td>
<td>SE (TestSuite_2) U { 70 71 72 74 75 76 77 78 80 }</td>
</tr>
<tr>
<td>TestSuite_2G3</td>
<td>SE (TestSuite_2) U { 70 71 72 74 75 }</td>
</tr>
<tr>
<td>TestSuite_2G13</td>
<td>SE (TestSuite_2) U { 70 71 72 74 75 76 77 78 80 }</td>
</tr>
<tr>
<td>TestSuite_3G1</td>
<td>SE (TestSuite_3) U { 76 77 78 80 }</td>
</tr>
<tr>
<td>TestSuite_3G2</td>
<td>SE (TestSuite_3) U { 91 92 }</td>
</tr>
<tr>
<td>TestSuite_3G12</td>
<td>SE (TestSuite_3) U { 76 77 78 80 91 92 }</td>
</tr>
</tbody>
</table>

**Table 19: Coverage of the original and extended test suites for pattern-matching case study**
<table>
<thead>
<tr>
<th>Test Suite</th>
<th>Statement Coverage (SC)</th>
<th>Percentage SC</th>
<th>Percentage Increase In SC</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestSuite_1</td>
<td>63</td>
<td>96.92</td>
<td>-</td>
</tr>
<tr>
<td>TestSuite_2</td>
<td>56</td>
<td>86.15</td>
<td>-</td>
</tr>
<tr>
<td>TestSuite_3</td>
<td>59</td>
<td>90.77</td>
<td>-</td>
</tr>
<tr>
<td>TestSuite_1g2</td>
<td>65</td>
<td>100.00</td>
<td>3.08</td>
</tr>
<tr>
<td>TestSuite_1g3</td>
<td>63</td>
<td>96.92</td>
<td>0.00</td>
</tr>
<tr>
<td>TestSuite_1g23</td>
<td>65</td>
<td>100.00</td>
<td>3.08</td>
</tr>
<tr>
<td>TestSuite_2g1</td>
<td>65</td>
<td>100.00</td>
<td>13.85</td>
</tr>
<tr>
<td>TestSuite_2g3</td>
<td>61</td>
<td>93.85</td>
<td>7.70</td>
</tr>
<tr>
<td>TestSuite_2g13</td>
<td>65</td>
<td>100.00</td>
<td>13.85</td>
</tr>
<tr>
<td>TestSuite_3g1</td>
<td>63</td>
<td>96.92</td>
<td>6.15</td>
</tr>
<tr>
<td>TestSuite_3g2</td>
<td>61</td>
<td>93.85</td>
<td>3.08</td>
</tr>
<tr>
<td>TestSuite_3g12</td>
<td>65</td>
<td>100.00</td>
<td>9.23</td>
</tr>
</tbody>
</table>

Table 20: Results of coverage analysis for pattern-matching case study

4.3.2.7. Validating the effectiveness of TSC using a mutation-based criterion

To perform mutation analysis, the author seeded some defects in the implementation (Appendix C.5) by using common mistakes which developers might make, e.g., interchanging the variables used in nested loops such as $i$ and $j$, the tightening or weakening of guard conditions (such as changing $i-1$ to $i$ in loops), etc. We identified 16 mutants in this way. We seeded 16 defects which are provided in Appendix C.9. Each defect is seeded separately, i.e. the test suites are executed over 16 mutants of the program.

We execute the original test suites, and the enriched test suites over the 16 mutants of the program to measure the number of mutants killed by each test suite. Table 21 shows the mutants killed by the original and the extended test suites. The results of the mutation analysis are shown in Table 22.
<table>
<thead>
<tr>
<th>Test Suite</th>
<th>Mutants Killed (MK)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestSuite_1</td>
<td>{ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 }</td>
</tr>
<tr>
<td>TestSuite_2</td>
<td>{ 1 2 3 4 5 6 7 10 14 15 16 }</td>
</tr>
<tr>
<td>TestSuite_3</td>
<td>{ 1 2 3 4 5 6 7 8 9 10 14 }</td>
</tr>
<tr>
<td>TestSuite_1G2</td>
<td>MK (TestSuite_1) U { 15, 16 }</td>
</tr>
<tr>
<td>TestSuite_1G3</td>
<td>MK (TestSuite_1)</td>
</tr>
<tr>
<td>TestSuite_1G23</td>
<td>MK (TestSuite_1) U { 15, 16 }</td>
</tr>
<tr>
<td>TestSuite_2G1</td>
<td>MK (TestSuite_2) U { 8, 9, 11, 12, 13 }</td>
</tr>
<tr>
<td>TestSuite_2G3</td>
<td>MK (TestSuite_2) U { 8, 9 }</td>
</tr>
<tr>
<td>TestSuite_2G13</td>
<td>MK (TestSuite_2) U { 8, 9, 11, 12, 13 }</td>
</tr>
<tr>
<td>TestSuite_3G1</td>
<td>MK (TestSuite_3) U { 11, 12, 13 }</td>
</tr>
<tr>
<td>TestSuite_3G2</td>
<td>MK (TestSuite_3) U { 15, 16 }</td>
</tr>
<tr>
<td>TestSuite_3G12</td>
<td>MK (TestSuite_3) U { 11, 12, 13, 15, 16 }</td>
</tr>
</tbody>
</table>

Table 21: Mutants killed by the original and extended test suites for pattern-matching case study

<table>
<thead>
<tr>
<th>Test Suite</th>
<th>Total Mutants Killed (MK)</th>
<th>Percentage MK</th>
<th>Percentage Increase In MK</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestSuite_1</td>
<td>14</td>
<td>87.50</td>
<td>-</td>
</tr>
<tr>
<td>TestSuite_2</td>
<td>11</td>
<td>68.75</td>
<td>-</td>
</tr>
<tr>
<td>TestSuite_3</td>
<td>11</td>
<td>68.75</td>
<td>-</td>
</tr>
<tr>
<td>TestSuite_1G2</td>
<td>16</td>
<td>100.00</td>
<td>12.50</td>
</tr>
<tr>
<td>TestSuite_1G3</td>
<td>14</td>
<td>87.50</td>
<td>0.00</td>
</tr>
<tr>
<td>TestSuite_1G23</td>
<td>16</td>
<td>100.00</td>
<td>12.50</td>
</tr>
<tr>
<td>TestSuite_2G1</td>
<td>16</td>
<td>100.00</td>
<td>31.25</td>
</tr>
<tr>
<td>TestSuite_2G3</td>
<td>13</td>
<td>81.25</td>
<td>12.50</td>
</tr>
<tr>
<td>TestSuite_2G13</td>
<td>16</td>
<td>100.00</td>
<td>31.25</td>
</tr>
<tr>
<td>TestSuite_3G1</td>
<td>14</td>
<td>87.50</td>
<td>18.75</td>
</tr>
<tr>
<td>TestSuite_3G2</td>
<td>13</td>
<td>81.25</td>
<td>12.50</td>
</tr>
<tr>
<td>TestSuite_3G12</td>
<td>16</td>
<td>100.00</td>
<td>31.25</td>
</tr>
</tbody>
</table>

Table 22: Results of the mutation analysis for pattern-matching case study
4.3.2.8. Discussion

The effectiveness of TSC is validated by comparing three JUnit test suites which are created by different developers. Coverage and mutation analyses are performed to determine the effectiveness of the comparison performed by TSC.

The results of these analyses i) support the comparison performed by TSC, and ii) demonstrate the effectiveness of the enriched test suites.

i. TestSuite_1, TestSuite_2 and TestSuite_3 have 26, 20 and 16 test cases respectively. Their representative sets of equivalence classes have 21, 12 and 11 equivalence classes (as shown in Table 12). TSC considers TestSuite_1 the strongest test suite. Further, TSC considers TestSuite_2 stronger than TestSuite_3. This is because it contains more equivalence classes, i.e. it covers more scenarios. Hence it has a potential of detecting more defects than TestSuite_3.

ii. The coverage analysis showed that the stronger test suites have greater coverage (as shown in Table 21). Likewise, the mutation analysis showed that the strongest test suite killed more mutants than the other test suites (as shown in Table 23).

iii. The enriched test suites, which are obtained using the gaps identified during the comparison, showed better coverage and killed more mutants than the original test suites. Further, when a test suite is enriched using the strongest test suite, it showed greater coverage and killed more mutants compared to its enrichment using a less strong test suite. For example, TestSuite_2G1 showed 100% coverage and killed all the 16 mutants but TestSuite_2G3 showed 93.85% coverage and killed 13 mutants.

During coverage analysis, TestSuite_1G2 and TestSuite_1G23, showed a small increase in coverage i.e. 3.08%. This is because TestSuite_1 has already executed 63 statements out of 65, making it difficult to extend its coverage further. However, the coverage of these test suites reached the maximum, i.e. 100%. TestSuite_1G3 did not show any increase in coverage. This is because the test cases of TestSuite_3 that were added to TestSuite_1 did not execute any additional statement, and hence they failed to increase the coverage of TestSuite_1G3.

TestSuite_2G1 and TestSuite_2G13 have shown the greatest increase in coverage which is 13.85%, and the coverage of these test suites reached the maximum. These test suites have executed the statements 70, 71, 72, 74, 75, 76, 77, 78 and 80 that were not executed by TestSuite_2. These statements were executed due to the test cases of TestSuite_1 that were
added to TestSuite_2. The coverage of TestSuite_2G3 increased by 7.70% but did not reach the maximum.

TestSuite_3G12 showed 9.23% increase in coverage and its coverage reached the maximum. The coverage of TestSuite_3G1 and TestSuite_3G2 increased by 6.15% and 3.08% respectively, but did not reach the maximum.

The majority of the enriched test suites (5 out of 9) reached the maximum coverage. The rest of the test suites did not reach the maximum (i.e. 100%) which may be due to the following reasons:

i. The test suite is not enriched using a strong test suite.

We enrich a test suite using the gaps identified by comparing with another test suite (the enriching test suite). Therefore, the extension of the test suite depends on the enriching test suite. If the enriching test suite has some weaknesses, it may not identify all the gaps in the test suite. Eventually, it fails to fully enrich the test suite, and hence the coverage of the enriched test suite did not reach the maximum (i.e. 100%). For example, when we enrich TestSuite_2 using TestSuite_3, TestSuite_3 fails to identify all the gaps in TestSuite_2. This is because TestSuite_3 does not contain all the equivalence classes which are devised for the comparison. Therefore, the coverage of TestSuite_2G3 does not reach the maximum.

ii. Criteria used for identifying gaps and measuring effectiveness are different.

We enrich a test suite using the gaps which are identified using equivalence classes. As these equivalence classes are based on a criterion which is different from the coverage criterion, there is a possibility that the enriched test suite may not show much increase in coverage. This is because the test cases added in the enriched test suite may not execute further statements. For example, when we enrich TestSuite_1 using TestSuite_3, the test cases (T2, T3, T4, T5, T6, T7, and T12) which are added to TestSuite_1 to enrich it (TestSuite_1G3) do not execute any additional statements. Hence, they fail to increase the coverage of TestSuite_1.

If we devise equivalence classes using the same coverage criterion which is used to measure the effectiveness, the extension in size of the test suite and the increase in coverage of the test suite may be proportional. However, this would make the effectiveness analysis biased and defeat its purpose.
iii. Only one test case is selected for each equivalence class in the set difference. We find one test case, from the enriching test suite, for each equivalence class which is present in the difference set (Table 18). However, multiple test cases belong to the equivalence class in the enriching test suite. Therefore, there is a possibility that the test case we pick does not extend the coverage but other test cases which we do not pick do increase the coverage.

During mutation analysis, TestSuite_1 killed 14 mutants out of 16 but TestSuite_1G2, and TestSuite_1G23 killed all the 16 mutants. However, TestSuite_1G3 killed the same number of mutants as TestSuite_1 and did not show any increase in mutation-detection capability. This is because the test cases of TestSuite_3 that were added to TestSuite_1 did not kill any additional mutants.

TestSuite_2G1 and TestSuite_2G13 killed all the mutants. These test suites killed the mutants 8, 9, 11, 12 and 13 that were not killed by TestSuite_2. TestSuite_2G3 killed mutants 8 and 9 which were not killed by TestSuite_2, but it failed to kill all 16 mutants.

TestSuite_3G12 killed all 16 mutants. TestSuite_3G1 and TestSuite_3G2 killed more mutants than TestSuite_3 but it did not kill all of them. These test suites failed to kill all the mutants for the same reasons as we discussed for the coverage analysis.

The results of the coverage and mutation analyses are encouraging for the following reasons:

i. The test suites which are considered stronger by TSC showed greater coverage and killed more mutants than other test suites.

ii. Most of the enriched test suites (8 out of 9) showed an increase in coverage, and killed more mutants than the original test suites.

iii. The majority of the enriched test suites (5 out of 9) showed the maximum coverage and killed all the 16 mutants.

iv. When a test suite is enriched using the two other test suites, it showed 100% coverage and killed all the 16 mutants.

We used coverage analysis and mutation analysis to determine the effectiveness of TSC. These analyses have some issues with them. Though the enriched test suites showed greater statement coverage, the execution of additional statements may not necessarily expose defects which appear under certain executing conditions (or in certain contexts). Similarly,
the mutation analysis has its own issues. In mutation analysis, we seed defects independently (i.e. one at a time). However, if we seed all the defects simultaneously (i.e. all at once), there is a possibility that a defect can mask other defect(s). This is a weakness of mutation analysis.

4.3.2.9. Threats to validity

This case study has the following threats to its validity which affect the application of $TSC$ to other systems:

1. We have applied $TSC$ to small test suites. Therefore, this case study does not demonstrate the ability of $TSC$ to differentiate and enrich large test suites.

2. We have applied $TSC$ to a small number of test suites (i.e. three test suites). Hence, we cannot generalise the viability and effectiveness of $TSC$ based on comparing three test suites.

3. We have applied only one criterion for comparing test suites (black box). Hence, the ability of $TSC$ to use different criteria is not demonstrated by this case study.

4.4. Related Work

The need for comparing test suites emerged as a step in MD-CDCT in which we needed to compare component-level testing against system-level testing to decide on the adequacy of the component testing, and to extend the component testing (if necessary). We are not aware of any work on the comparison of test suites. The work closest to the comparison of test suites is the reduction of test suites. The reduction of test suites can be regarded as a special case of test suite comparison in which the reduced test suite is a subset of the original test suite. Therefore, we discuss the related work in the area of test suite reduction briefly.

Harrold et al. [168] propose to reduce a test suite using test case coverage. In this approach, a representative set of test cases is identified which does not have redundant test cases. The redundant test cases are those test cases that provide the same coverage which is achieved by some other test cases in the test suite. The redundant test cases are eliminated from the test suite which results in a smaller test suite. This approach is similar to ours, in that $TSC$ also transforms a test suite into a set of representative set of test cases. However, this approach is based on specific coverage criterion whereas $TSC$ is general in that it can use any arbitrary criterion to define equivalence classes.
Saif-Ur-Rehman and Nadeem [151] propose a tool (called TestFilter) for reducing test suites which is closest to our work. TestFilter produces a master test suite whose statement-coverage is equivalent to the statement-coverage of the test suite being reduced. It computes the union of non-redundant test cases (with respect to a certain criterion) of the test suite. Therefore, if a critical test case is missing in the test suites to be reduced, it will also be missing in the master test suite produced by TestFilter. In contrast to TestFilter, TSC allows the user to devise a set of equivalence classes (which can be considered as an abstract master test suite), and hence the user can add critical test cases to the master test suite. Furthermore, TestFilter is applied to abstract test suites whereas TSC has been validated on concrete and executable JUnit test suites. The comparison of concrete test suites is more challenging than comparing two abstract test suites. This is because different developers code test suites in different ways.

Jeffrey and Neelam [169] argue that some redundancy of test cases should be permitted while reducing a test suite. They believe that test cases that seem redundant (with respect to a certain criterion) may execute different behaviours of a system, and hence they are not redundant from a testing perspective. Hence, they should not be eliminated. This selective redundancy is necessary because when a test suite is reduced by eliminating the redundant test cases, there is a possibility that the fault-detection capability of that test suite is reduced as the redundant test cases may test different requirements, and may execute different behaviours.

McMaster and Memon [150] present a technique for test suite reduction using a call stack coverage criterion. In this technique, the test cases are executed and the call stacks produced during their execution are recorded. A “call stack” represents active function calls in a stack-based execution environment. These call stacks are used as a coverage criterion to reduce the test suite. However, this approach gives method-level coverage and does not guarantee that all the statements (within a method) are executed. The effectiveness of this method is measured by performing mutation analysis, whereas the effectiveness of TSC is validated using both coverage analysis and mutation analysis.

Da Silva Simao et al. [170] present a technique for reducing the size of test suites for regression testing. In this technique, each test case is summarised into a feature vector. These feature vectors contain information about the software behaviour. Using ART-2A (a self-organising neural network), these feature vectors are grouped into clusters. When a change is
made in the software, the cluster(s) that relate to the functionality being changed are identified. Eventually the test cases that are part of the identified clusters are executed instead of executing the whole test suite for regression testing. TSC can use the grouping of vectors (clusters) to devise equivalence classes for comparing test suites. The test cases that belong to the same cluster can be considered equivalent.

Vaysburg et al. [171] present a technique for reducing requirement-based test suites using Extended Finite State Machine (EFSM) dependency analysis. In this approach, different types of dependencies are identified between elements of the EFSM model of the system. For each requirement being tested by a test case, dependencies are identified among the parts of the model. This dependency information is used to reduce the test suite. Two test cases are considered to be similar if they have the same set of dependencies i.e. the same pattern of interactions between elements of the EFSM model. This can be used in TSC as a criterion for defining equivalence classes, i.e. the test cases that have the same dependency-set are put into the same equivalence class. Korel et al. [172] extend this approach to automatically identify changes in the model. Later, the dependence-based technique was applied to the set of test cases that were identified to execute the modified transitions. Chen et al. [173] extend [172] to handle complex representations of model changes.

Zhang et al. [174] suggest that cost-effective test suite reduction can be achieved through the optimisation of test requirements. The optimisation of testing requirements, such as removing redundancies, may lead to smaller test suites.

Xu [175] devise a technique called as Modified Greedy Algorithm using Weighted Set Covering (WSC) techniques to reduce the size of a test suite. In this technique, test cases are assigned weight and coverage attributes. The test suite reduction becomes the problem of finding a subset of test suite with the minimum weight having the same coverage. Recently, Rout et al. [176] devise a priority-based technique for test suite reduction using WSC. They calculate cost of the test requirement and test cases. Using this cost, they calculate a priority (cost of test case divided by requirement of test case) factor for the test cases. They obtain a reduced test suite by removing the low priority test cases.

Dale et al. [177] propose Interaction-based Test-Suite Minimization (ITSM) which reduces a test suite without impacting its coverage of feature interactions. ITSM selects a subset of a test suite that maintains the coverage (up to a certain level supplied by user) of value combinations as the existing test suite. The value combinations are covered as coverage
targets. In ITSM, user can define coverage targets in different forms, such as a Cartesian product (i.e. every combination), or as explicit sets of value combinations to be covered. The test suite reduction becomes the problem of finding a subset of the test suite maintaining the coverage of value combinations (i.e. feature interactions).

Koochakzadeh et al. [178] propose a criterion for evaluating coverage-based test suite reduction techniques. Using the criterion, they evaluate the following techniques: all-def-use [179], branch [180], predicate-use [181], modified condition / decision coverage [182] and statement mutation [183]. They compare the redundancies, identified using these techniques, with manual redundancy decisions (performed through an inspection by a tester) in the context of JUnit test suites.

4.5. Discussion

Comparison of test suites determines which test suite is stronger amongst the two with respect to a certain criterion. Some advantages of comparing test suites are the following:

1. When there are multiple test suites available, the comparison can assist in choosing the best one for the maintenance phase of software.
2. The gaps identified during the comparison can be used to enrich the test suites to increase their test adequacy.
3. In CSD, the comparison of component testing and system testing can indicate the adequacy of component testing.

TSC has been applied to two case studies that involve the comparison of three test suites. These test suites were developed by different developers to simulate a real-world scenario in which people devise different test suites and write test drivers in different ways. TSC has compared concrete and executable test suites in contrast to many other methods for comparing test cases which were validated on abstract test suites.

TSC transforms a test suite into a set of equivalence classes that reduces the comparison of test suites to a simple set comparison. Apart from the simplicity of this comparison, the method has an overhead of devising the set of equivalence classes, which in essence is an abstract master test suite. The creation of these equivalence classes may limit the usefulness of the comparison because if we already have devised a master test suite, then why do we need to compare the other two test suites instead of using the master test suite, and throwing away (ignoring) the two test suites? These limitations can be justified as follows:
1. The master test suite is an abstract test suite, and it is not a concrete and executable one. It would need to be coded before using (executing) it.

2. The set of equivalence classes can also be constructed on the fly. In this way, we can avoid having a complete pre-fabricated master test suite.

In the context of this thesis, we expect to have a master test suite (CT) which is one of the test suites to be compared. The second test suite (CDCT) then has two characteristics:

1. It is (largely) a subset of the master set.
2. It is has context-dependent test cases having a greater tendency to uncover the defects related to the context of component reuse.

As TSC uses equivalence classes for comparison, to test a component for a particular context of reuse, we can reduce the overhead in this context by devising equivalence classes to target that particular reuse. For example, with a white-box approach, while devising equivalence classes, we can ignore the methods of the component which are not used in the context of reuse.

In the test cases, the effectiveness of TSC is determined by performing coverage and mutation analyses. These analyses support the effectiveness of comparison performed by TSC because the enriched test suites performed better than the original test suites in all cases except one, i.e. they have shown greater coverage and killed more mutants than the original test suites.

Most of the tasks of TSC can be automated. However, the task of devising equivalence classes and assigning them to test cases needs to be performed manually by testers. This is because this task requires judgment and expertise. However, if we use trace-based equivalence classes, this task can also be automated, which is demonstrated in Chapter 5.

Despite the threats to validity of this case study, we believe TSC is applicable to different contexts as it supports the use of different. For example, we can devise equivalence classes using one the following criteria:

1. Black-box criterion
2. White-box criterion
   a. Method coverage
   b. Statement coverage
   c. Branch coverage
   d. Path coverage
e. Any combination of a, b, c and d

To use white-box criterion for devising equivalence classes, we can partition test cases based on the statements (methods, branches, paths etc.) they execute, i.e. all those test cases which execute the same set of statements can be considered equivalent.

3. Gray-box criterion (i.e. combination of black-box and white-box criterion)

4. Mutation-based criterion

To use mutation-based criterion for devising equivalence classes, we can partition test cases based on the mutants (defects) which they kill, i.e. all those test cases which kill the same set of mutants can be considered equivalent.

5. Trace-based criterion

To use trace-based criterion for devising equivalence classes, we can partition test cases based on the traces which they produce, i.e. all those test cases which produce same execution trace can be considered equivalent.

6. Heap-based criterion

The tester can use the state of the heap for test case comparison, such as in OCAT (Object Capture based Automated Testing) [199]. In OCAT, testers can define equivalence classes using object instances captured dynamically from program executions. This technique might be useful in the context of object-oriented systems in which the tester is particularly concerned about the state of the heap.

Furthermore, criteria proposed for test suite reduction (discussed in the related work, Section 4.4) can be used in TSC for defining equivalence classes which demonstrates its generality.

In this case study, the effectiveness of TSC is dependent on the gaps identified in a test suite, and the gaps identified are based on the criterion used for devising equivalence classes. Hence, the effectiveness is dependent on the criterion. Therefore, if we measure the effectiveness using the same criterion that is used to devise equivalence classes, it would be unrealistically high. For example, if we use statement-coverage both for devising equivalence classes to compare test suites, and for measuring the effectiveness, the results will be biased. To counter this, we chose to devise equivalence classes using a black-box approach and measured the effectiveness using coverage analysis and mutation analysis.
CHAPTER 5 - A CASE STUDY

5.1. Introduction

The objective of this case study is to evaluate the viability of MD-CDCT by applying and evaluating it on a realistic system.

We evaluate MD-CDCT by applying it to the Lucene search engine [184]. We consider Lucene as the CB software and Searcher, one of its modules, as the component being used (tested). We model the usage scenarios of Lucene, and devise test cases from these usage scenarios (CDCT). We compare these test cases to the original CT of Searcher to identify weaknesses in the CT, and enhance the CT to address these weaknesses. We then create sequence diagrams for the usage scenarios which are used by MTCG to transform the enhanced CT into a concrete and executable test suite. Finally, we execute the enhanced CT to test Searcher and discuss the defects which were detected by the enhanced CT.

In this chapter, we describe Lucene and Searcher. We apply MD-CDCT to test Searcher. We then discuss the usefulness and limitations of MD-CDCT. Finally, we discuss limitations of this case study and threats to its validity.

5.2. Object Description

Lucene is a search engine library developed by Apache [185]. It performs full-text search, and has been widely used in other applications, tools and web-sites as an underlying search engine [186]. It is written in Java, and its source code consists of 331 Java files and 66,703 lines of code. The source code of Searcher consists of 112 Java files and 18,389 lines of code. Lucene is selected for this case study for the following reasons:

1. It is an open-source project and its source code is available which facilitates instrumentation.
2. A comprehensive test suite for component testing of Searcher (CT) is available as developer certification metadata (Section 2.4.3) which can be compared with the CDCT to evaluate and extend the component testing.
3. Extensive documentation is available in the form of books, online tutorials, etc.
Searcher is selected against other components of Lucene for the following reasons:

1. Searcher has more usage scenarios in Lucene than other components which make it interesting from a testing point of view.

2. The test suite for Searcher (CT) consists of over 23,160 test cases which show that Searcher is already well tested. Hence, this case study poses a real test for MD-CDCT to demonstrate its capability of extending test adequacy, and proving its usefulness.

3. CT is a test suite which contains other test suites developed by different testers. These testers have coded test drivers in different ways which presents a real-world scenario making the comparison of test suites a challenging task.

5.2.1. Lucene queries

Lucene indexes data and performs search over the indexed data stored in files. The text (to be searched) is specified in a query object. It returns the documents which contain the query text. Lucene supports the following types of queries:

1. Term query
2. Range query
3. Boolean query
4. Prefix query
5. Wildcard query
6. Fuzzy query
7. Phrase query
8. Multiphrase query
9. SpanFirst query
10. SpanNear query
11. SpanOr query
12. SpanNot query

A Term query \((Q_1)\) is used to search for a term. A term is a single word such as "hello".

A Range query \((Q_2)\) searches within a range, i.e. searching from a starting term through an ending term. For example, the Range query “year: [2002 TO 2005]” searches for the following strings: 2002, 2003, 2004 or 2005, in the “year” field of documents. The beginning and ending terms can be exclusive or inclusive which is controlled by a Boolean parameter in the API of the Range query.
A Boolean query \((Q_3)\) is used to combine queries using the following logical combinations: AND, OR and NOT. For example, the Boolean query "Jakarta Apache NOT Apache Lucene" searches for the documents which have “Jakarta Apache” in them, but do not contain the string “Apache Lucene”.

A Prefix query \((Q_4)\) searches documents that contain terms starting with a specified string. For example, the Prefix query “/languages” matches terms such as “/languages/Java”.

A Wildcard query \((Q_5)\) allows searching for terms with missing parts. The standard wildcard characters are the ‘*’ and the ‘?’ symbols. For example, the Wildcard query “bal?” matches terms like “ball”, “bale”, etc.

A Fuzzy query \((Q_6)\) searches for terms similar to the specified term. For example, the Fuzzy query “book” would match terms like “look”, “hook”, etc. The level of similarity is specified using a value from 0 to 1. A value closer to 1 means that terms with higher similarity will be matched.

A Phrase query \((Q_7)\) is used to search for phrases. A phrase is a sequence of words (terms) surrounded by double quotes such as "hello world". It can also locate terms that are within a certain distance from each other. The allowable distance between the terms is called slop.

A Multiphrase query \((Q_8)\) is a generalisation of the phrase query. For example, the Multiphrase query “little (barbie doll)” will match documents containing the phrases “little barbie” or “little doll”.

A SpanFirst query \((Q_9)\) allows for matching a span within a certain position from the start of the document. For example, a SpanFirst query can search for the documents in which the term “health” occurs within the first 100 words of the document.

A SpanNear query \((Q_{10})\) matches terms which occur near each other, and it defines the maximum distance between terms. The distance is expressed as a number of words in the document. For example, a SpanNear query can search for documents in which the terms “computers” and “intelligence” occurs within a distance of four words of each other.

A SpanOr query \((Q_{11})\) allows for merging spans from other SpanQueries. For example, a SpanOr query can search for the documents which contain the terms “mining” and “safety” close to each other.
A SpanNot query \((Q_{12})\) removes matches that overlap with another SpanQuery. For example, a SpanNot query can search for the documents in which the term “social sciences” occurs but not close to the term "public".

*Lucene* also provides the facility of caching search results, performing remote search, and searching over multiple indexes (data stores).

### 5.2.2. Components (modules) of Lucene

*Lucene* contains the following modules:

1. Analyser
2. Indexer
3. Query Parser
4. Searcher
5. Store
6. Utils

*Analyser* extracts tokens from the query text. *Indexer* creates indexes for directories to perform efficient search. *Query Parser* parses the text (to be searched) and returns a query object. *Query Parser* is created using JavaCC [187]. *Searcher* performs search (to match the query) over the documents in the index. The documents have to be indexed before performing a search. The *Store* module deals with storing and retrieving persistent data. The *Utils* module contains some useful data structures which are used for implementing the search engine. The component diagram of *Lucene* is shown in Figure 24 in which the symbol shows that the artefact is a component of a system.

*Lucene* is used to search for documents in the following two ways:

1. We can pass the query text to the *QueryParser* object. *QueryParser* uses *Analyser* to analyse the text and generates the *Query* object. Finally, it passes the *Query* object to *Searcher* which performs the search over the *Indexed Data*.
2. We can create a *Query* object and directly pass it to *Searcher* to perform the search.
5.2.3. Tasks performed by Searcher

The Searcher component creates directory, query and searcher objects, and invokes the search method. The directory object points to the directory containing the files to be searched. The query object contains the query text. The searcher object performs search and return a hits object which is a collection of the matched documents. Searcher performs the following tasks in order to perform a search:

1. Creation of the directory object
   The directory object points to the directory on which the search is to be performed. It is created as follows:
   
   ```java
   RAMDirectory directory = new RAMDirectory();
   ```

2. Creation of the query object
   The query object contains the query to be executed. A query object can be created in the following two ways:
   
   i) Using a query constructor:
      
      ```java
      Query query = new TermQuery(field, text);
      ```

      The first parameter is the name of the field of the document within which the search is to be executed. The second parameter is the value which is to be searched.
ii) Using the query parser object:

\[
\text{SimpleAnalyzer analyzer = new SimpleAnalyzer();}
\]
\[
\text{QueryParser queryParser = new QueryParser(field, analyzer);}
\]
\[
\text{Query query = queryParser.parse(text);}
\]

*QueryParser* requires an *Analyzer* to extract tokens from the input stream.

3. Creation of the *IndexedSearcher* object:

The *IndexedSearcher* object is responsible for searching the document.

\[
\text{IndexSearcher searcher = new IndexSearcher(directory);}\]

The parameter *directory* represents the data store on which the search is performed.

4. Invocation of the *search* method:

The *search* method returns a collection of the *documents* matched by the query. It also computes the score of each *document* which is the degree of the relevance of the matched *document* to the query.

\[
\text{Hits hits = searcher.search(query);}\]

### 5.2.4. Overloaded implementations of the search method

The search method is overloaded with the following implementations:

1. **Search all documents using query parameter**

   This implementation returns all the documents that match the query. We refer to this method as *M*\(_1\) and it has the following signature:

   \[
   \text{public Hits search(query)}\]

2. **Search all documents using query and filter parameters**

   This implementation returns all the documents that match the query, and satisfy the filter which is specified as a parameter. In *Lucene*, *Filters* are used to filter the query output, i.e. reduce the scope of search. They support efficient search as they offer a yes/no decision for each document in the index for selection or rejection during the search operation. We refer to this method as *M*\(_2\) and it has the following signature:

   \[
   \text{public Hits search(query, filter)}\]

3. **Search all documents using query, filter and sort parameters**

   This implementation returns all the documents that match the query and satisfy the *Filter* criterion. The documents are returned in the sorted order specified by the *Sort* parameter. The *Sort* parameter represents the name of the field (of a document) on
which the returned documents are sorted. We refer to this method as $M_3$ and it has the following signature:

\[
\text{public Hits search(query, filter, sort)}
\]

4. Search all documents using \textit{query} and \textit{sort} parameters

This implementation returns all the documents that match the query. The documents are returned in the sorted order specified by the \textit{Sort} parameter. We refer to this method as $M_4$ and it has the following signature:

\[
\text{public Hits search(query, sort)}
\]

5. Search top documents using \textit{query} and \textit{int} parameters

This implementation returns the top $N$ documents that match the query. The number $N$ is specified as an integer parameter. We refer to this method as $M_5$ and it has the following signature:

\[
\text{public TopDocs search(query, int)}
\]

6. Search top documents using \textit{query}, \textit{filter} and \textit{int} parameters

This implementation returns the top $N$ documents that match the query and satisfy the \textit{Filter} criterion. We refer to this method as $M_6$ and it has the following signature:

\[
\text{public TopDocs search(query, filter, int)}
\]

7. Search top documents using \textit{query}, \textit{filter}, \textit{int}, and \textit{sort} parameters

This implementation returns the top $N$ documents that match the query and satisfy the filter criterion. The documents are returned in the sorted order specified by the \textit{Sort} parameter. We refer to this method as $M_7$ and it has the following signature:

\[
\text{public TopDocs search(query, filter, int, sort)}
\]

8. Search all documents using \textit{query} and \textit{HitCollector} parameters

This implementation returns the documents that match the query into an object (of type \textit{HitCollector}) which is passed as a parameter. We refer to this method as $M_8$ and it has the following signature:

\[
\text{public void search(query, hitCollector)}
\]

9. Search all documents using \textit{query}, \textit{filter} and \textit{HitCollector} parameters

This implementation returns the documents that match the query, and satisfy the filter criterion, into the object (\textit{HitCollector}) which is passed as a parameter. We refer to this method as $M_9$ and it has the following signature:
public void search(query, filter, HitCollector)

The search method can also perform search over more than one index (multiple indexes), indexes which are located on remote machines (remote indexes), and it can cache the search results for performance purposes.

5.3. Experimental Planning

In Chapter 4, we compared test suites by devising equivalence classes using a black-box approach. In this case study, CT which is provided with Lucene as developer certification consists of 16,387 lines of code and it has over 23 thousand test cases. Hence, devising equivalence classes using a black-box approach is expensive due to the manual effort required for devising equivalence classes and assigning these equivalence classes to the test cases of CT. Therefore, we devise equivalence classes using the execution traces of the test suites being compared.

Searcher executes different types of queries. Further, it has different overloaded implementations of the search method to execute different kinds of searches (Section 5.2.4). Therefore, Searcher has a large number of usage scenarios in Lucene making the second step costly for this case study. To apply MD-CDCT in a cost-effective manner, we defer the generation of executable test cases to the last step which is the execution of the extended test suite. This is because we perform trace-based comparison of test suites and we can predict the traces of a test suite without executing it. By doing this, we avoid the extraction of test cases from a concrete test suite to reduce the cost of the second step. Eventually, in the last step we convert only those test cases into concrete and executable form which were not executed during the component testing, reducing the cost of the framework. This alteration makes the framework cost-effective when the system has a large number of usage scenarios.

Previously, we defined a test case as a sequence of method calls along with their parameter values, expected return values and executing conditions. For this case study, we use trace-based equivalence classes, and our test cases produce the following trace:

1. Method: the search method (overloaded implementation) invoked by the test case.
2. Query type: the type of query (such as Phrase query) executed by the test case.
3. Query data category: the category of the query text contained in the query object.
4. Executing condition: the executing condition in which the test case runs. For this case study, it refers to the documents in the data store on which a test case runs.
We take these traces as the features (attributes) of a test case which will be used for comparing test cases. For this case study, our test case is the search method (being tested) along with parameter (query type which is a parameter type and query data which is the parameter value), the number of documents that the query is expected to match and the number of documents in the index (data store) which are represented by the feature “executing condition” of the trace of a test case.

We execute the following steps in the case study:

1. Modelling usage scenarios of Lucene which use Searcher.
2. Deriving component test cases for CDCT
3. Comparing test suites
   i. Instrumenting CT (to obtain execution traces)
   ii. Capturing execution traces of CT
   iii. Compare CDCT with CT to find gaps in CT
4. Enriching CT
   i. Devising an extending test suite (\(\Delta CT\)) to extend the test adequacy of CT.
      We expand the functionality which is missing in CT. We create test cases for different scenarios of the functionality which was not tested during the component testing.
   ii. Extending CT by adding \(\Delta CT\) to give \(CT'\).
5. Executing the enriched test suite
   i. Transforming \(CT'\) into the concrete test suite \(CT''\).
   ii. Executing \(CT''\) to test Searcher

In this case study, we measure the following variables:

1. The number of test cases in CDCT
2. The number of test cases in CT
3. The number of unique traces of CDCT
   As we compare test suites using traces, we take each unique trace as an equivalence class. All the test cases which produce the same trace are considered equivalent.
4. The number of unique traces of CT
5. The number of test cases in \(\Delta CT\)
6. The number of test cases of \(\Delta CT\) which failed during testing
The scope of this case study is confined to the basic search functionality of Searcher as our objective is to determine the capability of MD-CRCT to identify gaps in testing of Searcher. Therefore, we shall not consider the advanced features of Searcher, e.g., search over multiple indexes, cached searches, remote search, etc.

5.4. Applying the Framework to the Case Study

5.4.1. Modelling usage scenarios of CB software

We model usage scenarios of Lucene which use Searcher (task 1 in Figure 2). Searcher executes twelve types of queries. It has nine overloaded implementations of the search method to allow for filtering, sorting etc. We model each query using each overloaded implementation of the search method for CDCT of Searcher.

We create test cases for these usage scenarios by specifying different executing conditions and data for the query object to cover different testing scenarios. For this purpose, we partition the:

1. Executing conditions of test cases
   We partition executing conditions of a test case (documents in data store on which a test case runs), using the following criteria:
   1) The number of documents in the index (data store) to perform search on.
      a. Zero documents
      b. One document
      c. More than one document
   2) The number of documents that the query returns (matches).
      a. Zero matches
      b. One match
      c. More than one match
   There are six valid combinations of the documents and the matches as shown in Table 23.

2. Test data of test cases
   We partition the query text (the data embedded in the query object) which is passed to the search method. This partitioning for the different types of queries is shown in Table 24.
<table>
<thead>
<tr>
<th>Query Type</th>
<th>Data Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Term</td>
<td>(T_1)</td>
<td>Terms containing zero characters</td>
</tr>
<tr>
<td>2.</td>
<td>(T_2)</td>
<td>Terms containing one character</td>
</tr>
<tr>
<td>3.</td>
<td>(T_3)</td>
<td>Terms containing more than one character</td>
</tr>
<tr>
<td>4. Range</td>
<td>(R_1)</td>
<td>Queries with upper and lower range inclusive</td>
</tr>
<tr>
<td>5.</td>
<td>(R_2)</td>
<td>Queries with upper and lower range exclusive</td>
</tr>
<tr>
<td>6. Boolean</td>
<td>(B_1)</td>
<td>Queries with required clause only</td>
</tr>
<tr>
<td>7.</td>
<td>(B_2)</td>
<td>Queries with prohibited clause only</td>
</tr>
<tr>
<td>8.</td>
<td>(B_3)</td>
<td>Queries with both required and prohibited clause</td>
</tr>
<tr>
<td>9.</td>
<td>(B_4)</td>
<td>Queries with neither required nor prohibited clause</td>
</tr>
<tr>
<td>10. Prefix</td>
<td>(P_1)</td>
<td>Terms containing zero characters</td>
</tr>
<tr>
<td>11.</td>
<td>(P_2)</td>
<td>Terms containing one character</td>
</tr>
<tr>
<td>12.</td>
<td>(P_3)</td>
<td>Terms containing more than one character</td>
</tr>
<tr>
<td>13. Wildcard</td>
<td>(W_1)</td>
<td>Queries that contain the '?' wildcard</td>
</tr>
<tr>
<td>14.</td>
<td>(W_2)</td>
<td>Queries that contain the '*' wildcard</td>
</tr>
<tr>
<td>15.</td>
<td>(W_3)</td>
<td>Queries that contain no wildcard</td>
</tr>
<tr>
<td>16. Fuzzy</td>
<td>(F_1)</td>
<td>Queries with high fuzziness ( (&gt;= 0) similarity &lt; 0.5)</td>
</tr>
<tr>
<td>17.</td>
<td>(F_2)</td>
<td>Queries with low fuzziness ( (&gt;=0.5) similarity &lt;= 1.0)</td>
</tr>
<tr>
<td>18. Phrase</td>
<td>(Ph_1)</td>
<td>Queries with zero slops</td>
</tr>
<tr>
<td>19.</td>
<td>(Ph_2)</td>
<td>Queries with one slop</td>
</tr>
<tr>
<td>20.</td>
<td>(Ph_3)</td>
<td>Queries with more than one slop</td>
</tr>
<tr>
<td>21. Multiphrase</td>
<td>(Mp_1)</td>
<td>Queries with zero phrases</td>
</tr>
<tr>
<td>22.</td>
<td>(Mp_2)</td>
<td>Queries with one phrase</td>
</tr>
<tr>
<td>23.</td>
<td>(Mp_3)</td>
<td>Queries with more than one phrase</td>
</tr>
<tr>
<td>24. SpanFirst</td>
<td>(Sf_1)</td>
<td>Spans within one word of the document</td>
</tr>
<tr>
<td>25.</td>
<td>(Sf_2)</td>
<td>Spans within more than one words of the document</td>
</tr>
<tr>
<td>26. SpanNear</td>
<td>(Sn_1)</td>
<td>Match terms within a span of one word</td>
</tr>
<tr>
<td>27.</td>
<td>(Sn_2)</td>
<td>Match terms within a span of more than one word</td>
</tr>
<tr>
<td>28. SpanNot</td>
<td>(Snot_1)</td>
<td>Span term and one remove term</td>
</tr>
<tr>
<td>29.</td>
<td>(Snot_2)</td>
<td>Span terms and more than one remove terms</td>
</tr>
<tr>
<td>30. SpanOr</td>
<td>(Sor_1)</td>
<td>One clause</td>
</tr>
<tr>
<td>31.</td>
<td>(Sor_2)</td>
<td>More than one clause</td>
</tr>
</tbody>
</table>

Table 23: Executing conditions of a test case for the Lucene case study

Table 24: Data-partitioning of query text (content of query object) for the Lucene case study

In this case study, we compare test suites using their traces. In the context of trace-based comparison, a test case is represented by the trace it produces. Therefore, we need a
criterion for devising equivalence classes using traces of test cases. A trace of a test case has the following information (features / attributes): i) executing condition, ii) the search method executed, iii) query type, and iv) query data category. We explore some combinatorial techniques in which coverage targets are defined using value combinations such as a Cartesian product (i.e. every combination) [177]. These combinatorial techniques are popular among testers and a lot of literature exists on applying these techniques [188-192]. We explore the following value combinations of features:

1. All features (feature criterion)
2. All combinations of features (combination criterion)
3. All pairs of features (pairwise criterion)

By feature criterion, we mean the test suite covers all the features of a test case. It requires that a test suite should have at least one test case for every feature. By combination criterion, we mean the test suite covers all combinations of the features. It requires that a test suite should have at least one test case for every combination of features. By pairwise criterion, we mean the test suite covers all pairs of the features. It requires that a test suite should have at least one test case for every pair of input parameters. These criteria are often called feature coverage, combination coverage, and pairwise coverage [193, 194].

We need a criterion that sits between the minimum (exercising each feature at least once) which is not effective, and full combinatorial combination of features which is too large (9 methods x 6 executing conditions x 12 query types x 31 query data category = 20,088 combinations) and hopelessly inefficient. Therefore, we chose the pairwise criterion. It considers the pairing of interesting features, e.g. behaviour of a method under certain executing conditions (which may represent a context). Therefore, we devise CDCT such that it satisfies the pairwise criterion.

To apply pairwise criterion in CDCT, we create test cases for the following pairs of features:

1. Search method and Executing condition \((M, E)\)
2. Search method and Query type \((M, Q)\)
3. Query type and Executing condition \((Q, E)\)
4. Query type and Query data category \((Q, D)\)

We omit the pairing of Query data category \((D)\) with search method \((M)\) and executing condition \((E)\), i.e. \((M, D)\) and \((E, D)\), because the Query data category \((D)\) is specific to a
particular Query type. It partitions different values of a particular Query type. Hence, its pairing is limited to Query type.

After executing a test case, we can perform different types of output checking of the return value. The output checking is different from the expected output value. Expected output value refers to the value(s) which a program should return whereas output checking means what types of checking are performed on the expected output to determine the success of the test case. We perform the following output checks:

1. Check whether the correct number of documents are returned \((O_1)\)
2. Check whether the score of the documents is computed correctly \((O_2)\)

5.4.2. Deriving component test cases for CDCT

For this case study, we chose to perform a trace-based comparison in which a test case is represented by its trace. We derive traces of the component test cases for CDCT from the usage model of Searcher instead of deriving the component test cases (task 2 in Figure 2). Moreover, the system is a shell around the component, and the component is not obscured as it is not embedded deep inside the system which is quite unusual. Taking advantage of this, we can derive the traces straight away without using the tool (MTCG). By doing this, we avoid the following overhead for performing CDCT which saves time:

1. Transforming CDCT into executable form
2. Instrumenting CDCT to log execution traces

The traces of CDCT \((CDCT\text{\_UNIQUE\_TRACES})\) are provided in Appendix D.1.

5.4.3. Comparing test suites

We compare CT and CDCT to identify gaps in component testing (task 3 in Figure 2). As noted before, we shall compare these test suites by comparing their traces.

5.4.3.1. Instrumenting CT

We instrument CT to capture its execution traces. During instrumentation, we log the following information for each test case:

1. The search method which is executed (i.e. which overloaded method is invoked).
2. The query type that is passed as a parameter to the search method.
3. The value of the query object.
4. The executing condition (documents and expected matches) on which the test runs.
5.4.3.2. Capturing execution traces of CT

We execute the instrumented CT and capture the execution traces. An example trace of a test case is shown in Figure 25. CT produced 23,160 traces, one for each test case.

<table>
<thead>
<tr>
<th>Example Trace</th>
<th>M1.Term.T1.E1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td></td>
</tr>
<tr>
<td>M1:</td>
<td>the method “public Hits search(Query)”</td>
</tr>
<tr>
<td>Term:</td>
<td>the type of query object passed as a parameter</td>
</tr>
<tr>
<td>T1:</td>
<td>the data-category of the term query (Table 24)</td>
</tr>
<tr>
<td>E1:</td>
<td>the environment of the test case (Table 23)</td>
</tr>
</tbody>
</table>

Figure 25: Example trace and its description for the Lucene case study

As we devise equivalence classes using traces, we take each unique trace as an equivalence class. CT has 151 unique traces. These unique traces (CT\text{UNIQUE TRACES}) are shown in Appendix D.2.

During instrumentation, we observed that CT performs different types of output checking. It performs the following output checking after executing the search method:

1. Checks whether the correct number of documents are returned (O1).
2. Checks whether the score of matching documents is within acceptable range (O2).
3. Checks whether the actual search text exists in the retrieved documents (O3).
4. Checks whether the documents are returned in the correct order (O4).
5. Checks whether the score of the documents is computed correctly (O5).

In the earlier case studies, the behaviour checking was straightforward. In this case study, the behaviour checking emerged as an interesting problem. Different types of output checking performed by a test case shows that two test cases executing the same method with the same input under the same executing conditions should be considered different if they perform different output checking. It shows that TSC can be improved to consider the output checking while comparing test cases, which may increase its ability to differentiate two test cases. For this case study, we deal with the output checking at the test suite level rather than considering it as an attribute of a test case. We capture the output checks performed by CT and CDCT using instrumentation and log the traces for output checking in a separate file. We then perform a simple set comparison to identify weaknesses (shortcomings) in terms of
output checking. For example, output checking of CT would be considered adequate if it performs all types of output checking that are performed by CDCT.

5.4.3.4. Comparing test suites

We identify gaps in CT using CDCT by comparing their unique traces (which act as equivalence classes). For this purpose, we compare $CT_{UNIQUE_TRACES}$ and $CDCT_{UNIQUE_TRACES}$.

Applying the pairwise criterion, we identify the pairs of features which are covered in CDCT but not in CT. $CDCT_{UNIQUE_TRACES}$ has 265 pairs which are shown in Appendix D.3. $CT_{UNIQUE_TRACES}$ has 132 pairs which are shown in Appendix D.4. $CDCT_{UNIQUE_TRACES}$ has 133 pairs which are not present in $CT_{UNIQUE_TRACES}$ (i.e. $CDCT_{UNIQUE_TRACES} - CT_{UNIQUE_TRACES}$) as shown in Table 25.

We consider the output checking of CT adequate if it performs all types of output checking that are performed by CDCT. For this purpose, we perform a simple set comparison. CT performs the following output checking: $O_1$, $O_2$, $O_3$, $O_4$, and $O_5$. CDCT performs the following output checking: $O_1$ and $O_2$. Hence, there is no gap in CT regarding output checking and it has performed adequate output checking.
| M_1 : E_1 | M_1 : Boolean | M_4 : Prefix | Fuzzy : E_1 | Fuzzy : F_1 |
| M_1 : E_2 | M_1 : Fuzzy | M_4 : Range | Fuzzy : E_2 | Multiphrase : M_p_1 |
| M_1 : E_3 | M_1 : Multiphrase | M_4 : SpanFirst | Fuzzy : E_3 | Prefix : P_1 |
| M_1 : E_4 | M_1 : Phrase | M_4 : SpanNot | Multiphrase : E_1 | Prefix : P_1 |
| M_1 : E_5 | M_1 : Prefix | M_4 : SpanOr | Multiphrase : E_2 | Multiphrase : M_p_1 |
| M_1 : E_6 | M_1 : Range | M_4 : Term | Multiphrase : E_2 | Multiphrase : M_p_1 |
| M_2 : E_1 | M_2 : SpanFirst | M_4 : Wildcard | Phrase : E_1 | SpanOr : S_o_r_1 |
| M_2 : E_2 | M_2 : SpanNear | M_4 : SpanFirst | Phrase : E_2 | Term : T_1 |
| M_2 : E_3 | M_2 : SpanNot | M_4 : SpanNear | Term : T_1 | Term : T_1 |
| M_2 : E_4 | M_2 : SpanNot | M_4 : Prefix | Range : E_1 | Range : E_1 |
| M_2 : E_5 | M_2 : SpanNot | M_4 : Range | Range : E_1 | Multiphrase : M_p_1 |
| M_2 : E_6 | M_2 : SpanNot | M_4 : Fuzzy | Multiphrase : M_p_1 | Multiphrase : M_p_1 |
| M_3 : E_1 | M_3 : Boolean | M_5 : Prefix | Prefix : E_1 | Multiphrase : M_p_1 |
| M_3 : E_2 | M_3 : Fuzzy | M_5 : Prefix | Prefix : E_2 | Prefix : E_2 |
| M_3 : E_3 | M_3 : Multiphrase | M_5 : SpanNot | Prefix : E_3 | Prefix : E_3 |
| M_3 : E_4 | M_3 : Phrase | M_5 : SpanOr | Multiphrase : E_1 | Multiphrase : M_p_1 |
| M_3 : E_5 | M_3 : Prefix | M_5 : SpanOr | Multiphrase : E_2 | Multiphrase : M_p_1 |
| M_3 : E_6 | M_3 : Range | M_5 : Term | Multiphrase : E_2 | Multiphrase : M_p_1 |
| M_4 : E_1 | M_4 : SpanFirst | M_5 : Wildcard | Fuzzy : E_1 | Multiphrase : M_p_1 |
| M_4 : E_2 | M_4 : SpanNear | M_5 : SpanFirst | Fuzzy : E_2 | Multiphrase : M_p_1 |
| M_4 : E_3 | M_4 : SpanNot | M_5 : SpanNear | Fuzzy : E_3 | Multiphrase : M_p_1 |
| M_4 : E_4 | M_4 : SpanNot | M_5 : Prefix | Fuzzy : E_4 | Multiphrase : M_p_1 |
| M_4 : E_5 | M_4 : SpanNot | M_5 : Range | Fuzzy : E_4 | Multiphrase : M_p_1 |
| M_4 : E_6 | M_4 : SpanNot | M_5 : Fuzzy | Fuzzy : E_4 | Multiphrase : M_p_1 |
| M_5 : E_1 | M_5 : Wildcard | M_6 : Phrase | SpanNear : E_1 | SpanNear : E_1 |
| M_5 : E_2 | M_5 : Wildcard | M_6 : Phrase | SpanNear : E_2 | SpanNear : E_2 |
| M_5 : E_3 | M_5 : Wildcard | M_6 : Phrase | SpanNear : E_3 | SpanNear : E_3 |
| M_5 : E_4 | M_5 : Wildcard | M_6 : Fuzzy | Multiphrase : M_p_1 | Multiphrase : M_p_1 |
| M_5 : E_5 | M_5 : Wildcard | M_6 : Fuzzy | Multiphrase : M_p_1 | Multiphrase : M_p_1 |
| M_5 : E_6 | M_5 : Wildcard | M_6 : Fuzzy | Multiphrase : M_p_1 | Multiphrase : M_p_1 |
| M_6 : E_1 | M_6 : Wildcard | M_7 : Phrase | SpanNear : E_1 | SpanNear : E_1 |
| M_6 : E_2 | M_6 : Wildcard | M_7 : Phrase | SpanNear : E_2 | SpanNear : E_2 |
| M_6 : E_4 | M_6 : Wildcard | M_7 : Fuzzy | Multiphrase : M_p_1 | Multiphrase : M_p_1 |
| M_6 : E_5 | M_6 : Wildcard | M_7 : Fuzzy | Multiphrase : M_p_1 | Multiphrase : M_p_1 |
| M_6 : E_6 | M_6 : Wildcard | M_7 : Fuzzy | Multiphrase : M_p_1 | Multiphrase : M_p_1 |

Table 25: Pairs missing in CT
5.4.4. Enriching CT

We enrich CT to target the gaps identified in component testing (task 4 in Figure 2). We use the gaps identified by the pairwise criterion for extending CT. We devise an extending test suite ($\Delta CT$) for testing the functionality of Searcher which was not tested well or at all by CT. To devise this test suite, we analyse the pairs missing in CT (Table 25). These gaps indicate discrepancies in CT which can be at pair-level or feature-level.

By pair-level discrepancy, we mean a feature is tested but it is not tested when paired with some other feature. For example, the method $M_1$ is tested in CT but it is not tested under the executing condition $E_2$.

By feature-level discrepancy, we mean a feature is not tested at all. For example, the method $M_1$ is not tested with any of the executing conditions ($E_1$-$E_6$). As a method is invoked by a test case, and the test case always runs under one of the executing conditions ($E_1$-$E_6$), the absence of pairing of $M_1$ with all six executing conditions indicates that the method is not executed by any of the test cases in CT. This implies that the method is not tested at all by CT. The feature-level discrepancy identifies an area which is not tested.

As noted in Section 1.3, testers can decide how to target the gap depending on the nature of the gap. We deal with pair-level discrepancies by simply adding a test case to CDCT which contributes (produces) the missing feature pair (e.g. $M_1$: $E_1$). To deal with feature-level discrepancies, we devise multiple test cases to extensively test the area of Searcher which is not tested instead of just filling the gap (which we did with pair-level discrepancies).

We devise an extending test suite ($\Delta CT$) to deal with these discrepancies which is provided in Appendix D.5. We then extend CT to $CT'$ by adding $\Delta CT$ to it.

5.4.5. Executing enriched test suite

We execute the enriched $CT'$ to test Searcher (task 5 in Figure 2). $\Delta CT$ is an extension to CDCT containing the test cases for the functionality of the component which was not tested by CT. Therefore, $\Delta CT$ is likely to expose defects. However, for the CB software in which the context of reuse of a component changes significantly, the execution of CDCT can also expose defects. We chose to execute the $\Delta CT$ instead of executing $CT'$. 
We generate test cases for $\Delta CT$ (by modelling corresponding usage scenarios of *Lucene*) to reduce the application cost of *MD-CDCT* as noted before (in Section 5.3). We transform $\Delta CT$ into an executable form using MTCG and execute it to test *Searcher*.

### 5.4.5.1. Transforming the extending test suite into an executable test suite using MTCG

We create SMCs for the usage scenarios of *Lucene* which correspond to the test cases in $\Delta CT$. These SMCs are used by *MTCG* to generate executable test cases and they are provided in Appendix D.7. The sequence diagrams from which the SMCs are derived are provided in Appendix D.6.

*QueryParser* can generate a query object for the following seven types of queries: Term, Boolean, Range, Fuzzy, Wildcard, Phrase and Prefix. However, *QueryParser* cannot parse Multiphrase, SpanFirst, SpanNear, SpanOr and SpanNot queries. Therefore, we have to model these queries with each of the *search* methods that execute these queries in $\Delta CDCT$. We create SMC models for each type of executing environment (i.e. $E_1$- $E_6$) which are: smc1, smc2, smc3, smc4, smc5 and smc6 for $E_1, E_2, E_3, E_4, E_5$ and $E_6$ respectively.

The SMC model, test data and xUnit model for smc1 are provided in Appendices D.7, D.8 and D.9 respectively. The SMC models, test data and xUnit for the remaining are provided online [195].

We transform $\Delta CT$ into a concrete and executable test suite using *MTCG*. The concrete and executable test suite is provided in Appendix D.10.

This step has highlighted some limitations of *MTCGPrototype*. We dealt with these limitations by either finding a work around or, if that was not possible, modifying transformation rules of *MTCGPrototype* to generate concrete and executable test cases. These limitations are discussed in Section 5.6.2.

### 5.4.5.2. Executing the enriched test suite

Finally, we execute $\Delta CT'$ for more extensive testing of *Searcher*. The execution results of $\Delta CT'$ are shown in Table 26.
### Table 26: Execution results of ΔCT/

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Result</th>
<th>Test Case</th>
<th>Result</th>
<th>Test Case</th>
<th>Result</th>
<th>Test Case</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>√</td>
<td>20</td>
<td>√</td>
<td>39</td>
<td>√</td>
<td>58</td>
<td>√</td>
</tr>
<tr>
<td>2</td>
<td>√</td>
<td>21</td>
<td>√</td>
<td>40</td>
<td>√</td>
<td>59</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>√</td>
<td>22</td>
<td>√</td>
<td>41</td>
<td>√</td>
<td>60</td>
<td>√</td>
</tr>
<tr>
<td>4</td>
<td>√</td>
<td>23</td>
<td>√</td>
<td>42</td>
<td>√</td>
<td>61</td>
<td>√</td>
</tr>
<tr>
<td>5</td>
<td>√</td>
<td>24</td>
<td>√</td>
<td>43</td>
<td>√</td>
<td>62</td>
<td>√</td>
</tr>
<tr>
<td>6</td>
<td>√</td>
<td>25</td>
<td>√</td>
<td>44</td>
<td>√</td>
<td>63</td>
<td>√</td>
</tr>
<tr>
<td>7</td>
<td>√</td>
<td>26</td>
<td>√</td>
<td>45</td>
<td>√</td>
<td>64</td>
<td>√</td>
</tr>
<tr>
<td>8</td>
<td>√</td>
<td>27</td>
<td>√</td>
<td>46</td>
<td>X</td>
<td>65</td>
<td>√</td>
</tr>
<tr>
<td>9</td>
<td>√</td>
<td>28</td>
<td>√</td>
<td>47</td>
<td>X</td>
<td>66</td>
<td>√</td>
</tr>
<tr>
<td>10</td>
<td>X</td>
<td>29</td>
<td>√</td>
<td>48</td>
<td>X</td>
<td>67</td>
<td>√</td>
</tr>
<tr>
<td>11</td>
<td>X</td>
<td>30</td>
<td>√</td>
<td>49</td>
<td>√</td>
<td>68</td>
<td>√</td>
</tr>
<tr>
<td>12</td>
<td>√</td>
<td>31</td>
<td>√</td>
<td>50</td>
<td>√</td>
<td>69</td>
<td>√</td>
</tr>
<tr>
<td>13</td>
<td>√</td>
<td>32</td>
<td>√</td>
<td>51</td>
<td>√</td>
<td>70</td>
<td>√</td>
</tr>
<tr>
<td>14</td>
<td>√</td>
<td>33</td>
<td>√</td>
<td>52</td>
<td>√</td>
<td>71</td>
<td>√</td>
</tr>
<tr>
<td>15</td>
<td>X</td>
<td>34</td>
<td>√</td>
<td>53</td>
<td>√</td>
<td>72</td>
<td>√</td>
</tr>
<tr>
<td>16</td>
<td>√</td>
<td>35</td>
<td>√</td>
<td>54</td>
<td>√</td>
<td>73</td>
<td>√</td>
</tr>
<tr>
<td>17</td>
<td>√</td>
<td>36</td>
<td>√</td>
<td>55</td>
<td>√</td>
<td>74</td>
<td>√</td>
</tr>
<tr>
<td>18</td>
<td>√</td>
<td>37</td>
<td>X</td>
<td>56</td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>√</td>
<td>38</td>
<td>√</td>
<td>57</td>
<td>√</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend: √ means passed, X means failed

### 5.5. Data Interpretation

The presence of pairs $M_3:E_1$, $M_3:E_2$, $M_3:E_3$, $M_3:E_4$, $M_3:E_5$ and $M_3:E_6$ in the trace difference ($CDCT_{TRACE}$ - $CT_{TRACE}$) shows that the method $M_3$ is not tested with any of the executing condition ($E_1$ - $E_6$). That is, $M_3$ is missing in $CT$ and this is an area of $Searcher$ which is not tested at all. As this is a feature-level discrepancy, we expand this gap by devising multiple test cases to target $M_3$ instead of just filling the gap by simply adding a test case which produces the pair $M_3:E_1$. Similarly, the method $M_4$ is not tested by $CT$ at all.

The following test cases failed during testing (as shown in Table 26): 10, 11, 15, 37, 46, 47, 48, and 59.

Test cases 10 and 11 threw a run-time exception. Further investigation revealed that whenever $M_3$ and $M_4$ are invoked on an empty index (zero documents), they generate an error while sorting the search results.
Test case 15 failed as the method $M_7$ threw a run-time exception. This error is generated when no document is matched by the query and an attempt is made to sort the search results.

Test cases 37, 46 and 47 failed because the Prefix query did not work properly with the attribute `Field.Index.ANALYSED` of the field object. However, it works fine with the attribute `Field.Index.NOT_ANALYSED`.

Test case 48 failed because the SpanNot query did not return the right number of documents. The investigation revealed that the SpanNot query converts the text in documents to lower case implicitly, which fails when the exclude clause of the SpanNot query has a word in upper case. For example, when a SpanNot query containing “OCL” in its exclude clause is run on a document containing the text “to automate testing, OCL can be used”, it fails. However, if we specify the exclude clause using lowercase (i.e. “ocl”), it works fine.

The methods $M_3$ and $M_4$ were not tested at all. These methods sort the search results before returning them. We created a test case in $\Delta CT$ (test case 59) to target the sorting feature of `Searcher`. We tried to sort the search results on a field containing more than one word (e.g. the body field). This test case generated the following exception: “java.lang.RuntimeException: there are more terms than documents in field”. This shows that the sorting field cannot have a multi-word value. This can be interpreted as a defect in the implementation of sorting functionality or a limitation of `Searcher`.

5.6. Discussion

This case study has demonstrated the viability and usefulness of MD-CDCT. The framework has identified some weaknesses in the testing of `Searcher (CT)`. These weaknesses (gaps) are used to enrich CT which eventually discovered some defects that were not detected by CT even though it has thousands of test cases. Code coverage would not have necessarily detected these errors just by executing the program statements which were not executed before. Instead, these errors are detected by running certain types of queries, containing certain types of data under certain executing conditions, using certain methods which were identified by MD-CDCT.

Further, this exercise showed that MTCG can generate test cases for complex applications. Creation of sequence diagrams and instrumentation of the test suite was done
manually. The use of tool support would make the application of MD-CDCT faster and more user-friendly.

*MTCG* creates the expected objects for comparison with the objects returned by the methods of components which interact with the system. For this purpose, it requires either an API for creating these objects or the source code of the component. COTS components that do not provide an API or access to their source code may be difficult to test using MD-CDCT.

This case study has i) highlighted some challenging aspects of test suite comparison, ii) exposed limitations of the version of *MTCG* which was used to generate test cases for an ATM simulation system in Chapter 3, and iii) suggested an improvement to *TSC* for comparing test suites.

### 5.6.1. Challenges in comparing test suites

Test suite comparison emerged as a challenging aspect of this case study for the following reasons:

1. **The semantics of comparing a system test suite against a component test suite needs to be investigated because a system test suite consists of interactions with system (or components) whereas a component test suite contains method calls which are transparent to the system. We have ignored the test cases of CT which test internal methods of Searcher (sub-unit level test cases). We consider only those test cases which test the interface of Searcher.**

2. **The extraction of abstract test cases (for comparison purposes) from a concrete test suite is yet another challenging issue for the following reasons:**
   
a. **The definition of a test case is not standard. A JUnit test case is a test driver which executes the test cases contained in it. The term test case may refer to just test data in one context and can refer to a concrete and executable test case in another context. Generally, a test case consists of a method (to be tested), execution conditions (environment), input and expected output [38]. For components, a test case is a sequence of method calls along with their parameter values, executing conditions and expected output.**

   b. **The criterion for determining the adequacy of output checking of a test case is another challenge. For example, after executing the search method, should we just check the number of documents returned or should we also ensure that the**
documents are retrieved in the correct order. Further, some output checking is implicit, e.g. if we process the output of a test case in a loop, we are implicitly counting the number of documents.

**CT** contains a number of test suites developed by different testers. These testers have coded test drivers in different ways, i.e., they used different algorithms to generate test cases in a test driver. Further, parameter values and assertions are coded in complex ways, as follows:

i. A parameter of a method call is itself a method call (e.g. snippet 1 in Figure 26). Further, the attribute `scoreDocs` of the object (which is returned by the search method) is accessed. However, the prototype implementation of **MTCG** does not support accessing attributes of objects.

ii. The object that is passed as the second parameter (to the search method) is defined inside the method call (e.g. snippet 2 in Figure 26). The variation in coding style and the complexity of code makes the extraction of test cases (from an executable test suite) a non-trivial task, especially for this case study.

| Snippet 1: | `result = searcher.search( csrq("data","1","6",T,T), null,1000 ).scoreDocs;` |
| Snippet 2: | `searcher.search( pq, `new` HitCollector( ) { `public final void collect( int doc, float score ) {` `assertTrue( score == 2.0f );` `}` `);` |
| Snippet 3: | `Filter filter1 = `new` Filter() { `public` DocIdSet getDocIdSet(IndexReader reader) {` `BitSetbitset = `new` BitSet(1);` `bitset.set(0);` `return `new` DocIdBitSet(bitset);` `}` `};` |
| Snippet 4: | `Hits hits = searcher.search(query1, `new` Filter() {`public` DocIdSet getDocIdSet( IndexReader reader) {` `BitSetbitset = `new` BitSet(1);` `bitset.set(0);` `return `new` DocIdBitSet(bitset);` `});` `});` |
| Snippet 5: | `searcher.search( query1, `new` HitCollector() { `public` void collect( int doc, float score) {` `vector.add( `new` Integer(doc));` `});` |

Figure 26: Code snippets of test suite for Searcher
5.6.2. Limitations of MTCG

This case study served as a real challenge for the applicability of MTCG to a large and complex system. The test cases generated by MTCG in the ATM case study (in Chapter 3) were simple as they only executed a method and compared the return value (variables or objects) with the expected value using an assertion. However, the test cases of Searcher are harder to generate than the ones generated for the ATM system because they:

1. Require type-casting.
2. Compare collections of objects instead of objects. Moreover, different overloaded implementations of the search method return different types of collections, such as the Hits and TopDocs. Further, these collections contain different types of objects, such as the HitDoc, FieldDoc, and ScoreDoc.
3. Require defining a method within another method which was not considered when MTCGPrototype was developed.
4. Use access modifiers.
5. Execute span queries which require declaring an array.

We found a work around for some of the limitations. For the remaining limitations, we modified MTCGPrototype to generate these test cases. These workarounds and modifications are discussed below.

MTCGPrototype does not provide a facility for type-casting. However, the test cases of Searcher use type-casting to set the score of a document, e.g. `float score = (float) 1.0;`. We worked around this problem by using a suffix ‘f’ to specify the float value in the source model, e.g. `float score = 1.0f;`.

The search methods return a Hits object or TopDocs object containing the collection of matching documents. To retrieve these documents (to compare with the expected documents), we added the methods getHitDocs and getScoreDocs to the classes Hits.java and TopDocs.java respectively. Similarly, for comparing the expected object with the actual object (returned by the search method), we need a method to compare these objects. Further, this method should also ensure that the documents are returned in the correct order. Therefore, we created a class HitsVector.java with a method equals to check for the equality of the expected object and the returned object (which contains the matching documents), i.e. `assertTrue ( expectedHits.equals( actualHits ) )`. This method is shown in Appendix D.11.
We encountered another problem while generating the test cases which pass Filter as a parameter to the search method (i.e. $M_2$, $M_3$, $M_6$ and $M_7$). These Filters define a method inside the body of their constructor (e.g. snippet 3 in Figure 26). $MTCG_{Prototype}$ cannot define a method inside the body of another method (or constructor). This problem was dealt with using the freeParameterText attribute of method’s parameter in the source model. The contents of freeParameterText are transferred from the source model to the generated test case. This attribute is dedicated to deal with the complex code which is hard to generate by means of transformation rules. On the one hand, it makes $MTCG$ flexible by leveraging its ability to generate code in complex scenarios. On the other hand, the code that is generated is poorly formatted as the code (typed in the freeParameterText) is appended as one-line text in the generated test case (e.g. snippet 4 in Figure 26). Similarly, the freeParameterText is used for passing the second parameter to the methods m8 and m9, which decreases the readability of the generated test case further (e.g. snippet 5 in Figure 26).

To support the specification of access modifiers, we modified Rule 6 and Rule 7 for the M2M and M2T transformations respectively. These modifications are shown in Figure 27. However, this modification supports the declaration of access modifiers only for the expected object. We can modify this rule further to provide this facility throughout the scope of the generated test case. When $MTCG_{Prototype}$ was developed, we had simple applications in mind whose test cases did not require access modifiers. Therefore, these transformation rules did not support the specification of access modifiers.

The test cases for Searcher which execute span queries (SpanFirst, SpanNear, SpanNot and SpanOr) require the declaration of an array, e.g.

```java
SpanQuery[] spanQuery1 = new SpanQuery[] {query1, query2};
```

We achieve this by changing Rule 3 for M2M and Rule 4 for M2T transformations. These modifications are shown in Figure 28.

In addition to the above limitations, this case study has suggested some improvements in $MTCG_{Prototype}$ to make it more user-friendly.

1. $MTCG$ should be enhanced to support access to attributes (properties) of objects, e.g. hits.scoreDocs.
2. $MTCG$ should be enhanced to add comments and formatting to increase the readability of the generated test cases.
To create multiple objects of the same type, we have to specify these objects manually in the source model. However, *MTCG* should be enhanced to provide a looping construct to specify the creation of multiple objects.

<table>
<thead>
<tr>
<th>Tefkat Rule 6 (Appendix B.1)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rule</strong> ExpectedValue_2_ExpectedValue(model, testSuite, interaction, testCase, message, assertion, method, e1, e2)</td>
</tr>
<tr>
<td><strong>Extends</strong> Message_2_Assertion(model, testSuite, interaction, testCase, message, assertion, method)</td>
</tr>
<tr>
<td><strong>Forall</strong> ::smc::ExpectedValue e1</td>
</tr>
<tr>
<td><strong>Where</strong> e1.owner = message</td>
</tr>
<tr>
<td><strong>Make</strong> ::xUnit::ExpectedValue e2</td>
</tr>
<tr>
<td><strong>Set</strong> e2.name = e1.name, e2.type = e1.type, e2.setter = e1.setter, [e2.accessModifier = e1.accessModifier, assertion.expectedValue = e2];</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MOFScript Rule 7 (Appendix B.2.2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>model.ExpectedValue::mapExpectedValue()</td>
</tr>
<tr>
<td>........................................</td>
</tr>
<tr>
<td>........................................</td>
</tr>
<tr>
<td>if (self.simpleAttribute.isDirectory() &amp; self.complexAttribute.isDirectory()) { // Simple Expected PDT</td>
</tr>
<tr>
<td>........................................</td>
</tr>
<tr>
<td>........................................</td>
</tr>
<tr>
<td>if (self.accessModifier.equals(&quot;&quot;))</td>
</tr>
<tr>
<td>........................................</td>
</tr>
<tr>
<td>........................................</td>
</tr>
<tr>
<td>text = text + &quot;\n\t&quot; + expectedType + &quot; + self.name + &quot; = new &quot; + expectedType + &quot;();&quot;</td>
</tr>
<tr>
<td>........................................</td>
</tr>
<tr>
<td>........................................</td>
</tr>
<tr>
<td>paramHashtable.put(&quot;Text&quot;, text)</td>
</tr>
<tr>
<td>java (&quot;com.m2t.CreateTemplate&quot;, &quot;writeText&quot;, paramHashtable, &quot;C:/eclipse/project/xunit2text&quot;)</td>
</tr>
<tr>
<td>........................................</td>
</tr>
</tbody>
</table>

**Figure 27: Changing M2M and M2T rules for access modifiers**
### Tefkat Rule 3 (Appendix B.1)

**RULE** Message\_2\_Assertion(model, testSuite, interaction, testCase, message, assertion, method)

**EXTENDS** Interaction\_2\_TestCase(model, testSuite, interaction, testCase)

FORALL Message message

WHERE message.owner = interaction

MAKE Assertion assertion, Method method

SET assertion.name = append( append(interaction.name, " "), message.name),

method.name = message.name,

method.static = message.static,

**method.array** = message.array,

method.freeParameterText = message.freeParameterText,

assertion.order = message.order,

assertion.assertionType = message.assertionType,

assertion.method = method,

testCase.assertion= assertion,

testSuite.testCase = testCase

### MOFScript Rule 4 (Appendix B.2.2)

```mofo
model.Method::mapMethod() {
  
  ................
  ................
  isArray = self.array.equalsIgnoreCase("yes")
  
  ................
  ................
  if isStatic == false {
    text = text + "\n\nrt classInstance + " + className + " new " + className + "+ cParameterText + ");";
  } else {
    if isArray == true {
      text = text + "\n\nrt classInstance + " + className + "+ [ ]{" + className + " [ ]{" + cParameterText + ");";
    
  } else {
    if isArray == false {
      text = text + "\n\nrt classInstance + " + className + "[ ]" + className + "[ ]{" + className + "+ cParameterText + ");";

  
  ................
  ................
  isArray = false
  }
```

Figure 28: Changing M2M and M2T rules for declaring arrays

### 5.6.3. An improvement in TSC

This case study has suggested an improvement to TSC. TSC does not consider the types of output checking performed by test cases while comparing test suites. To work around this issue, we capture the output checks performed by CT and CDCT. We then performed a simple set comparison to identify the weaknesses (shortcomings) in output checking. For example, output checking of CT is adequate if it performs all types of output checking that are performed by CDCT. However, we can re-define a test case to include the output checking as well, and refine TSC to compare test suites according to the new definition of a test case.

### 5.6.4. Limitations of the case study

This case study has the following limitations:
1. We created sequence diagrams manually for generating concrete test cases (using \textit{MTCG}). However, some Eclipse plug-ins are available (e.g. MaintainJ, Flowchart4j) that can automatically generate sequence diagrams from Java code.

2. We have instrumented the \textit{Searcher}’s test suite (CT) manually. However, we can investigate whether the following instrumentation tools can instrument the test suites to serve our purpose:
   i) The Java Instrumentation Engine (JIE) which is a source code processor which inserts instrumentation code in Java source code files [196].
   ii) The Java Instrument Package (”\texttt{java.lang.instrument}”) which came with Java 5.0. It modifies class files to insert additional byte-code.
   iii) SOOT [197] which is available as an Eclipse plug-in.

3. We tested Searcher using $\Delta CT$ instead of $CT'$. By doing this, we omitted the execution of the test cases of $CT$ (i.e. \textit{Searcher} was already tested).

4. We have ignored the test cases related to the advanced features of \textit{Searcher} (such as caching of search results) to keep this case study manageable. Further, \textit{MD-CDCT} is devised for CB software, we focused on the interface methods of \textit{Searcher} ignoring the test cases related to its internal methods. If we had included the advanced features and internal methods of \textit{Searcher}, \textit{MD-CDCT} would have detected at least the same defects for basic search functionality if not more.

\textbf{5.7. Threats to Validity}

This case study has the following threats to its validity which affect the application of \textit{MD-CDCT} to other systems:

1. We have taken \textit{Lucene} as the system and \textit{Searcher} as the component. The ideal CB software for this case study should be an application that uses a general-purpose component which is used in other applications as well. However, this case study makes the following compromises:
   i) \textit{Lucene} is a library instead of an application.
   ii) \textit{Searcher} is a component specifically designed for \textit{Lucene}. It has dependencies on the \texttt{Analyzer} and the \texttt{Query Parser} modules.

Though \textit{Lucene} is a library and not an application, it is independently deployable. Testers can safely take \textit{Lucene} as an application to write test drivers for testing
purposes. The interface of Searcher with Lucene models a system-component interaction (relationship) in CB software.

2. We cannot generalise the effectiveness of MD-CDCT based on one case study.

3. We have devised MD-CDCT and evaluated it ourselves. Ideally, the evaluation should be done by someone else to eliminate any bias.
CHAPTER 6 - CONCLUSION

6.1. Summary

This research was aimed at developing a framework to determine the adequacy of component testing in the context of Component-based Software Development (CSD) by extending the existing research in the following areas. Firstly, we devised a model-driven framework for performing context-dependent testing of software components. Secondly, we devised a technique for evaluating and extending the adequacy of testing at the time of a component’s reuse. Thirdly, we extended automation of software testing by making use of model transformation technology. Fourthly, we devised a technique for comparing test suites.

Although CSD brought some advantages to software development, it has complicated testing. The testing performed by the component provider is independent of the context, and the reuse of a component may require re-testing in the new context. To address this issue, we proposed a model-driven framework (MD-CDCT) to test the component for the context of its reuse.

Determining the adequacy of the re-testing of the component at the time of its reuse is another problem. To address this issue, we proposed a technique for evaluating and extending the test adequacy of component testing, and devised a method (TSC) and tool support. TSC identifies gaps in component testing which are used to extend the test adequacy of the context-dependent testing of the component.

Model-Driven Architecture (MDA) has emerged as a new software development paradigm which automates software development activities, reducing development time and cost. However, software testing has not fully benefited from MDA. This research makes another contribution in the area of MDA. Leveraging the use of MDA, a model-driven method (MTCG) was developed to generate concrete and executable test cases from software models by applying horizontal and vertical transformations. Prototype tool support for this method (MTCG_prototype) was provided and evaluated for automated generation of test cases from sequences of method calls (SMCs).

The case studies are used to evaluate the practicability of the contributions. Briefly, the Vending Machine example illustrates the application of MD-CDCT. The ATM case study illustrates how MDA can be used for generating test cases from software models using
The shortest-path and pattern-matching case studies demonstrate the application of TSC for comparing test suites. The Lucene case study evaluates MD-CDCT. It demonstrates the viability and usefulness of MD-CDCT, and its prototype tool support. In this case study, we identified weaknesses in the component testing of Searcher (a component of Lucene) for the context of reuse and enriched it. The enriched test suite detected some defects in Searcher which were not detected during component testing. This shows that i) MD-CDCT has the potential to uncover context-dependent defects of a component which were not detected during component testing, and ii) MDA tools can automate the generation of concrete and executable test cases from platform-independent models of software applications.

Finally, the case studies show that we can benefit from the component testing provided as component metadata to i) determine the reliability of the component for the context of reuse, and ii) extend the test adequacy of CDCT of a component for the new context.

6.2. Discussion

6.2.1. MD-CDCT

MD-CDCT provides a solution to the problem of determining the test adequacy of components at the time of reuse in a new context. The existing component testing techniques, which include component-metadata, component certification, built-in testing and mutation-based testing, do not specify any criterion for adequacy of the component testing for their reuse in a new context. MD-CDCT evaluates the adequacy of component testing for the context of its reuse, and enriches the context-dependent testing of components using the component testing performed at the time of component development.

MD-CDCT is based on software models (i.e. usage scenarios and sequence diagrams) which are created at an early stage of the software development. Therefore, testing activities (e.g. generating concrete and executable test cases) can commence before the implementation of CB software. A limitation of MD-CDCT is that it requires a component’s certification metadata in the form of developer or third-party testing, which is used for evaluating and extending test adequacy of CDCT.

Another advantage of MD-CDCT is that it can be applied to COTS components as it does not require the source code of the component. However, a COTS component which returns a collection object but does not provide any method (API) to extract objects from the
collection object (for comparing them with expected objects), cannot benefit from \textit{MD-CDCT} fully. For these components, the application of \textit{MD-CDCT} is limited to identifying weaknesses of component testing for the context of reuse and extending test adequacy of CDCT of the component. The generation of executable test cases can be worked around by providing an implementation of a method which extracts objects from the collection and compares them with the expected objects.

\textit{MD-CDCT} uses \textit{MTCG} for generating executable test cases and \textit{TSC} for comparing test suites. This work has highlighted certain complexities involved in generating executable test cases using our MDA-based approach. Some of these are: type-casting of objects, comparing collections of objects instead of objects, and the use of access modifiers. Similarly, the comparison of test suites using \textit{TSC} emerged as another challenging task. The extraction of test cases from a concrete test suite is a non-trivial task because i) the definition of a test case is not standard, ii) the criterion for determining the adequacy of output checking of a test case needs to be addressed, and iii) different testers code test drivers in different ways making the extraction of a test case from a concrete test suite a non-trivial task.

\textbf{6.2.2. Evaluation of MD-CDCT}

The \textit{Lucene} case study demonstrates that \textit{MD-CDCT} can be applied to large and complex applications, and it can detect the defects of components which appear in a certain context. The results of this case study are encouraging because \textit{MD-CDCT} has i) generated concrete and executable test cases, and ii) detected some real defects in \textit{Searcher} which was tested previously using thousands of test cases during component testing.

The evaluation of \textit{MD-CDCT} exposed some limitations of the prototype tool support. \textit{MTCG}_{Prototype} was modified to overcome some of the limitations. The remaining limitations were worked around.

The application of \textit{MTCG}_{Prototype} is expensive for large and complex systems. Therefore, \textit{MTCG}_{Prototype} should be enhanced with the improvements suggested in Sections 3.7 and 5.6.2 to make it cost-effective. Some of these improvements include i) automated extractions of SMC models from the graphical representations of sequence diagrams, ii) automated comparison of the actual trace with the sequence diagram, and iii) provision for using access modifiers, type-casting and comparing collections of objects. These enhancements would make \textit{MTCG}_{Prototype} more user-friendly. \textit{MTCG}_{Prototype} generates test
cases from SMC models instead of UML models, and hence it requires tool support to incorporate UML models when applied to large and complex systems. $MTCG_{prototype}$ generates test cases from SMCs. $MTCG$ itself is quite general in that it can be applied to generate test cases from other UML diagrams such as statecharts.

The evaluation of $TSC$ highlighted a potential improvement. $TSC$ does not consider the output checking which is performed as part of comparing test cases. This may affect the capability of $TSC$ to differentiate test suites. Further, the extraction of abstract test cases from a concrete test suite proved to be a non-trivial task. Its application to systems having a large test suite is expensive due to the manual assignment of equivalence classes to a large number of test cases of the test suite. However, the trace-based approach (Section 4.5) reduces the application cost of $TSC$ by facilitating the task of devising and assigning equivalence classes to the test cases. Further, the trace-based approach makes it possible to apply $TSC$ to large systems.

6.3. Future Work

$TSC$ can be enhanced to incorporate output checking which is performed by test cases to differentiate them while comparing test suites. This can improve its effectiveness in terms of identifying gaps in test suites.

We can explore different criteria for comparing test suites, and investigate i) which criterion identifies more gaps in the test suites and ii) which criterion identifies more context related defects, when the context of reuse changes significantly as below:

i. The component is developed on one platform but reused on another. The new platform may have a different way of data processing giving different interpretations of the information exchanged between the CB software and the component [198]. Further, in the case of concurrent software, platforms may also have different thread scheduling mechanisms which drive the program through different thread interleavings. As such, there is a possibility that concurrent software can work without failure on one platform, but always fail on another.

ii. The component is developed for standalone (desktop) systems but reused for web-based systems.

iii. The component is developed for one type of system (e.g. financial systems) but reused for a different type of system (e.g. geographical systems).
The $MTCG_{prototype}$ is based on sequence diagrams. However, we can explore the automated generation of test cases from other modelling diagrams (such as statecharts) to determine which modelling diagram is the best for generating test cases, in terms of cost and effectiveness.

The prototype implementation of $MTCG$ can be enhanced to facilitate its application and extend its usability. It can be refined to:

i. Generate test cases from UML 2.0 sequence diagrams instead of an SMC model.
ii. Access the attributes of objects and provide a facility for type-casting of objects.
iii. Add documentation comments to the generated test cases for increased readability.
iv. Add formatting tags in the code. This would allow generating readable test cases.
v. Provide a looping-construct while defining multiple objects of the same type in the source instead of specifying them one by one. This would simplify the creation of SMC model using $MTCG$. 
REFERENCES


Development Methods - Countering or Integrating the eXtremists, Toronto, Canada, 2001, pp. 43-55.


JavaCC: [https://javacc.dev.java.net/](https://javacc.dev.java.net/), accessed on 21/11/2009.


A.1: Test Suite for Dispenser (CT)

```java
import junit.framework.*;
public class CT_Dispenser extends TestCase {

    int[][] data1, data2, data3; // Test data for i) Test Case 1-15, ii) Test Cases 16-30 and iii) Test Cases 31-45

    protected void setUp() {
        data1 = new int[][] {
            {0, 0, -3}, {0, 3, -3}, {0, 4, -3}, {0, 1, -2}, {0, 5, -1},
            {1, 0, -3}, {1, 3, -3}, {1, 4, -3}, {1, 1, -2}, {1, 5, -1},
            {2, 0, 2}, {2, 3, 2}, {2, 4, 2}, {2, 1, -2}, {2, 5, -1}
        };

        data2 = new int[][] {
            {0, 0, 0, -3, -3}, {0, 4, 4, -3, -3}, {0, 1, 0, -2, -2}, {0, 5, 5, -1, -1},
            {101, -3, -3}, {1, 4, 4, -3, -3}, {1, 1, 1, -2, -2}, {1, 5, 5, -1, -1},
            {2, 0, 0, 2, 2}, {2, 3, 3, 2, 2}, {2, 4, 4, 2, 2}, {2, 1, 1, -2, -2}, {2, 5, 5, -1, -1}
        };

        data3 = new int[][] {
            {0, 0, 0, 0, -3, -3}, {0, 4, 4, -3, -3}, {0, 1, 0, -2, -2}, {0, 5, 5, -1, -1},
            {101, -3, -3}, {1, 4, 4, -3, -3}, {1, 1, 1, -2, -2}, {1, 5, 5, -1, -1},
            {2, 0, 2, 0, 2, 2}, {2, 3, 3, 2, 2}, {2, 4, 4, 2, 2}, {2, 1, 1, -2, -2}, {2, 5, 2, 5, -1, -1}
        };
    }

    public void testDispenser() {
        try {
            Dispenser dispenser;
            int expected_value, return_value;

            for (int i=0; i<15; i++) {
                dispenser = new Dispenser( );
                dispenser.setCredit( data1[i][0] );
                return_value = (int) dispenser.dispense( data1[i][1] );
                expected_value = data1[i][2];
                assertTrue( return_value == expected_value );
            }

            for (int i=0; i<15; i++) {
                dispenser = new Dispenser( );
                dispenser.setCredit( data2[i][0] );
                return_value = (int) dispenser.dispense( data2[i][1] );
                expected_value = data2[i][3];
                assertTrue( return_value == expected_value );
                return_value = (int) dispenser.dispense( data2[i][2] );
                expected_value = data2[i][4];
                assertTrue( return_value == expected_value );
            }

            for (int i=0; i<15; i++) {
                dispenser = new Dispenser( );
                dispenser.setCredit( data3[i][0] );
                return_value = (int) dispenser.dispense( data3[i][1] );
                expected_value = data3[i][4];
                assertTrue( return_value == expected_value );
                dispenser.setCredit( data3[i][2] );
                return_value = (int) dispenser.dispense( data3[i][3] );
                expected_value = data3[i][5];
                assertTrue( return_value == expected_value );
            }
        } catch (Exception exp) { fail("Exception in test case execution"); }
    }

    // End of Method
    // End of Class
}
```
A.2: SMC-Model for Vending Machine (CDCT)

Model VM_Model

Interaction vm1

- Message SETUP_VendingMachine
  - Owner Class VendingMachine
- Message SETUP_VendingMachineGUI
  - Owner Class VendingMachine
- Message SETUP_Parameter1
  - Owner Class Integer
  - Constructor Parameter item1
- Message vend
  - Parameter item1
  - Expected Value message1
  - Owner Class VendingMachine
- Message reset
  - Owner Class VendingMachine
- Message insert
  - Owner Class VendingMachine
- Message vend
- Message reset
- Message insert
- Message insert
- Message vend
- Message reset
- Message insert
- Message insert
- Message vend
- Message insert
- Message insert
- Message insert
- Message insert
- Message insert
- Message reset
- Message insert
- Message insert
- Message insert
- Message insert
- Message insert
- Message insert
- Message insert
- Message insert
- Message reset
- Message insert
- Message insert
- Message insert
- Message insert
- Message insert
- Message reset
- Message insert
- Message insert
- Message insert
- Message insert
- Message insert
- Message insert
- Message insert
- Message insert
A.3: Test data for Vending Machine (CDCT)

<?xml version="1.0" encoding="UTF-8"?>
<SequenceDiagram name="vm1">
    <Message name="Integer::SETUP_Parameter1">
        <ConstructorParameter name="item1" value="3"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message1" value="Insufficient credit"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message2" value="Insufficient credit"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message3" value="Take your item"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message4" value="Take your item:Take your change"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message5" value="Insufficient credit"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message6" value="Insufficient credit"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message7" value="Take your item"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message8" value="Insufficient credit"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message9a" value="Take your item"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message9b" value="Take your item"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message10a" value="Take your item:Take your change"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message10b" value="Insufficient credit"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message11a" value="Take your item"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message11b" value="Insufficient credit"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message12a" value="Insufficient credit"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message12b" value="Insufficient credit"/>
    </Message>
</SequenceDiagram>
<?xml version="1.0" encoding="UTF-8"?>
<SequenceDiagram name="vm2">
    <Message name="Integer::SETUP_Parameter1">
        <ConstructorParameter name="item1" value="2"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message1" value="Item unavailable"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message2" value="Item unavailable"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message3" value="Item unavailable"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message4" value="Item unavailable"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message5" value="Item unavailable"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message6" value="Item unavailable"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message7" value="Item unavailable"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message8" value="Item unavailable"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message9a" value="Item unavailable"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message9b" value="Item unavailable"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message10a" value="Item unavailable"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message10b" value="Item unavailable"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message11a" value="Item unavailable"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message11b" value="Item unavailable"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message12a" value="Item unavailable"/>
    </Message>
    <Message name="VendingMachine::vend">
        <ExpectedValue name="message12b" value="Item unavailable"/>
    </Message>
</SequenceDiagram>
<?xml version="1.0" encoding="UTF-8"?>
<SequenceDiagram name="vm3">
  <Message name="Integer::SETUP_Parameter1">
    <ConstructorParameter name="item1" value="35"/>
  </Message>
  <Message name="VendingMachine::vend">
    <ExpectedValue name="message1" value="Invalid selection"/>
  </Message>
  <Message name="VendingMachine::vend">
    <ExpectedValue name="message2" value="Invalid selection"/>
  </Message>
  <Message name="VendingMachine::vend">
    <ExpectedValue name="message3" value="Invalid selection"/>
  </Message>
  <Message name="VendingMachine::vend">
    <ExpectedValue name="message4" value="Invalid selection"/>
  </Message>
  <Message name="VendingMachine::vend">
    <ExpectedValue name="message5" value="Invalid selection"/>
  </Message>
  <Message name="VendingMachine::vend">
    <ExpectedValue name="message6" value="Invalid selection"/>
  </Message>
  <Message name="VendingMachine::vend">
    <ExpectedValue name="message7" value="Invalid selection"/>
  </Message>
  <Message name="VendingMachine::vend">
    <ExpectedValue name="message8" value="Invalid selection"/>
  </Message>
  <Message name="VendingMachine::vend">
    <ExpectedValue name="message9a" value="Invalid selection"/>
  </Message>
  <Message name="VendingMachine::vend">
    <ExpectedValue name="message9b" value="Invalid selection"/>
  </Message>
  <Message name="VendingMachine::vend">
    <ExpectedValue name="message10a" value="Invalid selection"/>
  </Message>
  <Message name="VendingMachine::vend">
    <ExpectedValue name="message10b" value="Invalid selection"/>
  </Message>
  <Message name="VendingMachine::vend">
    <ExpectedValue name="message11a" value="Invalid selection"/>
  </Message>
  <Message name="VendingMachine::vend">
    <ExpectedValue name="message11b" value="Invalid selection"/>
  </Message>
  <Message name="VendingMachine::vend">
    <ExpectedValue name="message12a" value="Invalid selection"/>
  </Message>
  <Message name="VendingMachine::vend">
    <ExpectedValue name="message12b" value="Invalid selection"/>
  </Message>
</SequenceDiagram>
A.4: xUnit-Model for Vending Machine (CDCT)

- Test Suite VM_Model
  - Test Case vm1
    - Assertion vm1_SETUP_VendingMachine
      - Method SETUP_VendingMachine
        - Constructor Parameter vendingMachine
    - Assertion vm1_SETUP_Parameter1
      - Method SETUP_Parameter1
        - Constructor Parameter item1
      - Constructor Parameter item1
    - Assertion vm1_vend
      - Expected Value message1
      - Method vend
        - Parameter item1
      - Owner Class VendingMachine
    - Assertion vm1_reset
      - Method reset
        - Owner Class VendingMachine
    - Assertion vm1_insert
      - Method insert
        - Owner Class VendingMachine
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
      - Assertion vm1_vend
      - Assertion vm1_reset
      - Assertion vm1_insert
      - Assertion vm1_insert
A.5: Test suite for Vending Machine (CDCT)

```java
import junit.framework.*;

class TestSuite_vm1 extends TestCase {

    public static void main(String args[]) {
        TestSuite testSuite = new TestSuite(TestSuite_vm1.class);
        testSuite.run(new TestResult());
    }

    public void test_vm1() {
        try {
            VendingMachine vendingMachine = new VendingMachine();
            int item1 = 3;
            Integer integer1 = new Integer(item1);
            String expected_message1 = "Insufficient credit";
            String message1 = (String) vendingMachine.vend(item1);
            assertTrue(message1.equals(expected_message1));
            vendingMachine.reset();
            vendingMachine.insert();
            String expected_message2 = "Insufficient credit";
            String message2 = (String) vendingMachine.vend(item1);
            assertTrue(message2.equals(expected_message2));
            vendingMachine.reset();
            vendingMachine.insert();
            String expected_message3 = "Take your item";
            String message3 = (String) vendingMachine.vend(item1);
            assertTrue(message3.equals(expected_message3));
            vendingMachine.reset();
            vendingMachine.insert();
            String cancel1 = (String) vendingMachine.cancel();
            String expected_message5 = "Insufficient credit";
            String message5 = (String) vendingMachine.vend(item1);
            assertTrue(message5.equals(expected_message5));
            vendin
assertTrue( message8.equals( expected_message8 ) );

vendingMachine.reset();
vendingMachine.insert();
vendingMachine.insert();
String expected_message9a = "Take your item";
String message9a = (String) vendingMachine.vend( item1 );
assertTrue( message9a.equals( expected_message9a ) );

vendingMachine.insert();
vendingMachine.insert();
String expected_message9b = "Take your item";
String message9b = (String) vendingMachine.vend( item1 );
assertTrue( message9b.equals( expected_message9b ) );

vendingMachine.reset();
vendingMachine.insert();
vendingMachine.insert();
vendingMachine.insert();
String expected_message10a = "Take your item:Take your change"
String message10a = (String) vendingMachine.vend( item1 );
assertTrue( message10a.equals( expected_message10a ) );
String expected_message10b = "Insufficient credit"
String message10b = (String) vendingMachine.vend( item1 );
assertTrue( message10b.equals( expected_message10b ) );

vendingMachine.reset();
vendingMachine.insert();
vendingMachine.insert();
String expected_message11a = "Take your item"
String message11a = (String) vendingMachine.vend( item1 );
assertTrue( message11a.equals( expected_message11a ) );
String expected_message11b = "Insufficient credit"
String message11b = (String) vendingMachine.vend( item1 );
assertTrue( message11b.equals( expected_message11b ) );

vendingMachine.reset();
vendingMachine.insert();
String expected_message12a = "Insufficient credit"
String message12a = (String) vendingMachine.vend( item1 );
assertTrue( message12a.equals( expected_message12a ) );

vendingMachine.insert();
String expected_message12b = "Insufficient credit"
String message12b = (String) vendingMachine.vend( item1 );
assertTrue( message12b.equals( expected_message12b ) );

} catch ( Exception exp ) {
    System.out.println( exp.toString() );
    fail("Exception occured during test case execution");
}

} //End of Method
} //End of Class
import junit.framework.*;

public class TestSuite_vm2 extends TestCase {

    public static void main( String args[] ) {
        TestSuite testSuite = new TestSuite(TestSuite_vm2.class);
        testSuite.run( new TestResult( ) );
    }

    public void test_vm2( ) {
        try {
            VendingMachine vendingMachine = new VendingMachine(  );
            int item1 = 2;
            Integer integer1 = new Integer( item1 );
            String expected_message1 = "Item unavailable";
            String message1 = (String) vendingMachine.vend( item1 );
            assertTrue( message1.equals( expected_message1 ) );
            vendingMachine.reset();
            vendingMachine.insert();
            String expected_message2 = "Item unavailable";
            String message2 = (String) vendingMachine.vend( item1 );
            assertTrue( message2.equals( expected_message2 ) );
            vendingMachine.reset();
            vendingMachine.insert();
            String expected_message3 = "Item unavailable";
            String message3 = (String) vendingMachine.vend( item1 );
            assertTrue( message3.equals( expected_message3 ) );
            vendingMachine.reset();
            vendingMachine.insert();
            String expected_message4 = "Item unavailable";
            String message4 = (String) vendingMachine.vend( item1 );
            assertTrue( message4.equals( expected_message4 ) );
            vendingMachine.reset();
            vendingMachine.insert();
            String cancel1 = (String) vendingMachine.cancel(  );
            String expected_message5 = "Item unavailable";
            String message5 = (String) vendingMachine.vend( item1 );
            assertTrue( message5.equals( expected_message5 ) );
            vendingMachine.reset();
            vendingMachine.insert();
            String cancel2 = (String) vendingMachine.cancel(  );
            String expected_message6 = "Item unavailable";
            String message6 = (String) vendingMachine.vend( item1 );
            assertTrue( message6.equals( expected_message6 ) );
            vendingMachine.reset();
            vendingMachine.insert();
            String cancel3 = (String) vendingMachine.cancel(  );
            String expected_message7 = "Item unavailable";
            String message7 = (String) vendingMachine.vend( item1 );
            assertTrue( message7.equals( expected_message7 ) );
            vendingMachine.reset();
            vendingMachine.insert();
            String cancel4 = (String) vendingMachine.cancel(  );
            String expected_message8 = "Item unavailable";
            String message8 = (String) vendingMachine.vend( item1 );
            assertTrue( message8.equals( expected_message8 ) );
        }
    }
}
vendingMachine.reset();
vendingMachine.insert();
vendingMachine.insert();
String expected_message9a = "Item unavailable";
String message9a = (String) vendingMachine.vend( item1 );
assertTrue( message9a.equals( expected_message9a ) );

vendingMachine.insert();
vendingMachine.insert();
String expected_message9b = "Item unavailable";
String message9b = (String) vendingMachine.vend( item1 );
assertTrue( message9b.equals( expected_message9b ) );

vendingMachine.reset();
vendingMachine.insert();
vendingMachine.insert();
vendingMachine.insert();
String expected_message10a = "Item unavailable";
String message10a = (String) vendingMachine.vend( item1 );
assertTrue( message10a.equals( expected_message10a ) );
String expected_message10b = "Item unavailable";
String message10b = (String) vendingMachine.vend( item1 );
assertTrue( message10b.equals( expected_message10b ) );

vendingMachine.reset();
vendingMachine.insert();
String expected_message11a = "Item unavailable";
String message11a = (String) vendingMachine.vend( item1 );
assertTrue( message11a.equals( expected_message11a ) );
String expected_message11b = "Item unavailable";
String message11b = (String) vendingMachine.vend( item1 );
assertTrue( message11b.equals( expected_message11b ) );

vendingMachine.reset();
vendingMachine.insert();
String expected_message12a = "Item unavailable";
String message12a = (String) vendingMachine.vend( item1 );
assertTrue( message12a.equals( expected_message12a ) );

vendingMachine.insert();
String expected_message12b = "Item unavailable";
String message12b = (String) vendingMachine.vend( item1 );
assertTrue( message12b.equals( expected_message12b ) );
}

} catch ( Exception exp ) {
    System.out.println( exp.toString() );
    fail("Exception occured during test case execution");
}

} //End of Method

} //End of Class
import junit.framework.*;

public class TestSuite_vm3 extends TestCase {

    public static void main(String args[]) {
        TestSuite testSuite = new TestSuite(TestSuite_vm3.class);
        testSuite.run(new TestResult());
    }

    public void test_vm3() {
        try {
            VendingMachine vendingMachine = new VendingMachine();
            int item1 = 35;
            Integer integer1 = new Integer(item1);
            String expected_message1 = "Invalid selection";
            String message1 = (String) vendingMachine.vend(item1);
            assertTrue(message1.equals(expected_message1));

            vendingMachine.reset();
            vendingMachine.insert();
            String expected_message2 = "Invalid selection";
            String message2 = (String) vendingMachine.vend(item1);
            assertTrue(message2.equals(expected_message2));

            vendingMachine.reset();
            vendingMachine.insert();
            String expected_message3 = "Invalid selection";
            String message3 = (String) vendingMachine.vend(item1);
            assertTrue(message3.equals(expected_message3));

            vendingMachine.reset();
            vendingMachine.insert();
            vendingMachine.insert();
            String expected_message4 = "Invalid selection";
            String message4 = (String) vendingMachine.vend(item1);
            assertTrue(message4.equals(expected_message4));

            vendingMachine.reset();
            vendingMachine.insert();
            String cancel1 = (String) vendingMachine.cancel();
            String expected_message5 = "Invalid selection";
            String message5 = (String) vendingMachine.vend(item1);
            assertTrue(message5.equals(expected_message5));

            vendingMachine.reset();
            vendingMachine.insert();
            String cancel2 = (String) vendingMachine.cancel();
            String expected_message6 = "Invalid selection";
            String message6 = (String) vendingMachine.vend(item1);
            assertTrue(message6.equals(expected_message6));

            vendingMachine.reset();
            vendingMachine.insert();
            String cancel3 = (String) vendingMachine.cancel();
            String expected_message7 = "Invalid selection";
            String message7 = (String) vendingMachine.vend(item1);
            assertTrue(message7.equals(expected_message7));

            vendingMachine.reset();
            vendingMachine.insert();
            String cancel4 = (String) vendingMachine.cancel();
            String expected_message8 = "Invalid selection";
            String message8 = (String) vendingMachine.vend(item1);
            assertTrue(message8.equals(expected_message8));
        }
    }
}
vendingMachine.reset();
vendingMachine.insert();
vendingMachine.insert();
String expected_message9a = "Invalid selection";
String message9a = (String) vendingMachine.vend( item1 );
assertTrue( message9a.equals( expected_message9a ) );

vendingMachine.insert();
vendingMachine.insert();
String expected_message9b = "Invalid selection";
String message9b = (String) vendingMachine.vend( item1 );
assertTrue( message9b.equals( expected_message9b ) );

vendingMachine.reset();
vendingMachine.insert();
vendingMachine.insert();
vendingMachine.insert();
String expected_message10a = "Invalid selection";
String message10a = (String) vendingMachine.vend( item1 );
assertTrue( message10a.equals( expected_message10a ) );
String expected_message10b = "Invalid selection";
String message10b = (String) vendingMachine.vend( item1 );
assertTrue( message10b.equals( expected_message10b ) );

vendingMachine.reset();
vendingMachine.insert();
vendingMachine.insert();
String expected_message11a = "Invalid selection";
String message11a = (String) vendingMachine.vend( item1 );
assertTrue( message11a.equals( expected_message11a ) );
String expected_message11b = "Invalid selection";
String message11b = (String) vendingMachine.vend( item1 );
assertTrue( message11b.equals( expected_message11b ) );

vendingMachine.reset();
vendingMachine.insert();
String expected_message12a = "Invalid selection";
String message12a = (String) vendingMachine.vend( item1 );
assertTrue( message12a.equals( expected_message12a ) );

vendingMachine.insert();
String expected_message12b = "Invalid selection";
String message12b = (String) vendingMachine.vend( item1 );
assertTrue( message12b.equals( expected_message12b ) );
}
} catch ( Exception exp ) {
    System.out.println( exp.toString() );
    fail("Exception occured during test case execution");
}
} //End of Method

} //End of Class
### A.6: Execution Traces of test suite for Vending Machine (CDCT)

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>1i</td>
<td>setCredit(0) dispense(3) output=-3</td>
</tr>
<tr>
<td>1ii</td>
<td>setCredit(0) dispense(2) output=-3</td>
</tr>
<tr>
<td>1iii</td>
<td>setCredit(0) dispense(35) output=-3</td>
</tr>
<tr>
<td>2i</td>
<td>setCredit(1) dispense(3) output=-3</td>
</tr>
<tr>
<td>2ii</td>
<td>setCredit(1) dispense(2) output=-3</td>
</tr>
<tr>
<td>2iii</td>
<td>setCredit(1) dispense(35) output=-3</td>
</tr>
<tr>
<td>3i</td>
<td>setCredit(2) dispense(3) output=2</td>
</tr>
<tr>
<td>3ii</td>
<td>setCredit(2) dispense(2) output=-2</td>
</tr>
<tr>
<td>3iii</td>
<td>setCredit(2) dispense(35) output=-1</td>
</tr>
<tr>
<td>4i</td>
<td>setCredit(3) dispense(3) output=2</td>
</tr>
<tr>
<td>4ii</td>
<td>setCredit(3) dispense(2) output=-2</td>
</tr>
<tr>
<td>4iii</td>
<td>setCredit(3) dispense(35) output=-1</td>
</tr>
<tr>
<td>5i</td>
<td>setCredit(0) dispense(3) output=-2</td>
</tr>
<tr>
<td>5ii</td>
<td>setCredit(0) dispense(2) output=-2</td>
</tr>
<tr>
<td>5iii</td>
<td>setCredit(0) dispense(35) output=-2</td>
</tr>
<tr>
<td>6i</td>
<td>setCredit(1) dispense(3) output=2</td>
</tr>
<tr>
<td>6ii</td>
<td>setCredit(1) dispense(2) output=-2</td>
</tr>
<tr>
<td>6iii</td>
<td>setCredit(1) dispense(35) output=-1</td>
</tr>
<tr>
<td>7i</td>
<td>setCredit(2) dispense(3) output=2</td>
</tr>
<tr>
<td>7ii</td>
<td>setCredit(2) dispense(2) output=-2</td>
</tr>
<tr>
<td>7iii</td>
<td>setCredit(2) dispense(35) output=-1</td>
</tr>
<tr>
<td>8i</td>
<td>setCredit(1) dispense(3) output=2</td>
</tr>
<tr>
<td>8ii</td>
<td>setCredit(1) dispense(2) output=-2</td>
</tr>
<tr>
<td>8iii</td>
<td>setCredit(1) dispense(35) output=-1</td>
</tr>
<tr>
<td>9i</td>
<td>setCredit(2) dispense(3) output=2 setCredit(2) dispense(3) output=2</td>
</tr>
<tr>
<td>9ii</td>
<td>setCredit(2) dispense(2) output=-2 setCredit(2) dispense(2) output=-2</td>
</tr>
<tr>
<td>9iii</td>
<td>setCredit(2) dispense(35) output=-1 setCredit(2) dispense(35) output=-1</td>
</tr>
<tr>
<td>10i</td>
<td>setCredit(4) dispense(3) output=2 setCredit(0) dispense(3) output=-3</td>
</tr>
<tr>
<td>10ii</td>
<td>setCredit(4) dispense(2) output=-2 setCredit(0) dispense(2) output=-2</td>
</tr>
<tr>
<td>10iii</td>
<td>setCredit(4) dispense(35) output=-1 setCredit(0) dispense(35) output=-1</td>
</tr>
<tr>
<td>11i</td>
<td>setCredit(2) dispense(3) output=2 setCredit(0) dispense(3) output=-3</td>
</tr>
<tr>
<td>11ii</td>
<td>setCredit(2) dispense(2) output=-2 setCredit(0) dispense(2) output=-2</td>
</tr>
<tr>
<td>11iii</td>
<td>setCredit(2) dispense(35) output=-1 setCredit(0) dispense(35) output=-1</td>
</tr>
<tr>
<td>12i</td>
<td>setCredit(1) dispense(3) output=-3 setCredit(1) dispense(3) output=-3</td>
</tr>
<tr>
<td>12ii</td>
<td>setCredit(1) dispense(2) output=-2 setCredit(1) dispense(2) output=-2</td>
</tr>
<tr>
<td>12iii</td>
<td>setCredit(1) dispense(35) output=-1 setCredit(1) dispense(35) output=-1</td>
</tr>
</tbody>
</table>
A.7: Test Suite for \( \Delta CT \)

```java
import junit.framework.*;
public class CT_Dispenser_Delta extends TestCase {

    int[][] data1, data2, data3; // Test data for i) Test Cases 1-6, and ii) Test Cases 7-12

    protected void setUp() {
        data1 = new int[][] {
            {3, 3, 2}, {3, 2, -2}, {3, 35, -1},
            {5, 3, 2}, {5, 2, -2}, {5, 35, -1}
        };
        data2 = new int[][] {
            {4, 3, 0, 3, 2, -3}, {4, 2, 0, 2, -2, -2}, {4, 35, 0, 35, -1, -1},
            {5, 3, 0, 3, 2, -3}, {5, 2, 0, 2, -2, -2}, {5, 35, 0, 35, -1, -1}
        };
    }

    public void testDispenser() {
        try {
            Dispenser dispenser;
            int expected_value, return_value;

            for (int i=0; i<6; i++) {
                dispenser = new Dispenser();
                dispenser.setCredit(data1[i][0]);
                return_value = (int) dispenser.dispense(data1[i][1]);
                expected_value = data1[i][2];
                assertTrue(return_value == expected_value);
            }

            for (int i=0; i<6; i++) {
                dispenser = new Dispenser();
                dispenser.setCredit(data2[i][0]);
                return_value = (int) dispenser.dispense(data2[i][1]);
                expected_value = data2[i][4];
                assertTrue(return_value == expected_value);
                dispenser.setCredit(data2[i][2]);
                return_value = (int) dispenser.dispense(data2[i][3]);
                expected_value = data2[i][5];
                assertTrue(return_value == expected_value);
            }
        } catch (Exception exp) { fail("Exception in test case execution"); }
    }

    //End of Method
}
//End of Class
```
APPENDIX B: MODEL-DRIVEN TEST CASE GENERATION

B.1: Tefkat rules (horizontal transformation)

TRANSFORMATION smc2xUnit : smc->xUnit

IMPORT http://smc
IMPORT http://xUnit

// Rule: 1
RULE Model_2_TestSuite(model, testSuite)
  FORALL Model model
  MAKE TestSuite testSuite
  SET testSuite.name = model.name
;

// Rule: 2
RULE Interaction_2_TestCase(model, testSuite, interaction, testCase)
  EXTENDS Model_2_TestSuite(model, testSuite)
  FORALL Interaction interaction
  WHERE interaction.owner = model
  MAKE TestCase testCase
  SET testCase.name = interaction.name,
  testSuite.testCase = testCase
;

// Rule: 3
RULE Message_2_Assertion(model, testSuite, interaction, testCase, message, assertion, method)
  EXTENDS Interaction_2_TestCase(model, testSuite, interaction, testCase)
  FORALL Message message
  WHERE message.owner = interaction
  MAKE Assertion assertion,
  Method method
  SET assertion.name = append( append(interaction.name, "_"), message.name),
  method.name = message.name,
  method.static = message.static,
  method.accessModifier = message.accessModifier,
  method.freeParameterText = message.freeParameterText,
  assertion.order = message.order,
  assertion.assertionType = message.assertionType,
  assertion.method = method,
  testCase.assertion = assertion,
  testSuite.testCase = testCase
;

// Rule: 4
RULE OwnerClass_2_OwnerClass(model, testSuite, interaction, testCase, message, assertion, method, c1, c2)
  EXTENDS Message_2_Assertion(model, testSuite, interaction, testCase, message, assertion, method)
  FORALL ::smc::OwnerClass c1
  WHERE c1.owner = message
  MAKE ::xUnit::OwnerClass c2
  SET c2.name = c1.name,
  c2.objectName = c1.objectName,
  c2.array = c1.array,
  method.ownerClass = c2
;

// Rule: 5
RULE Parameter_2_Parameter(model, testSuite, interaction, testCase, message, assertion, method, p1, p2)
  EXTENDS Message_2_Assertion(model, testSuite, interaction, testCase, message, assertion, method)
  FORALL ::smc::Parameter p1
  WHERE p1.owner = message
  MAKE ::xUnit::Parameter p2
  SET p2.name = p1.name,
  p2.type = p1.type,
  p2.setter = p1.setter,
  method.parameter = p2
;
// Rule: 6
RULE ExpectedValue_2_ExpectedValue(model, testSuite, interaction, testCase, message, assertion, method, e1, e2)
EXTENDS Message_2_Assertion(model, testSuite, interaction, testCase, message, assertion, method)
FORALL ::smc::ExpectedValue e1
  WHERE e1.owner = message
  MAKE ::xUnit::ExpectedValue e2
  e2.name = e1.name,
  e2.type = e1.type,
  e2.setter = e1.setter,
  e2.accessModifier = e1.accessModifier,
  assertion.expectedValue = e2;

// Rule: 7
RULE SimpleAttribute_2_Parameter(model, testSuite, interaction, testCase, message, assertion, method, p1, p2, sp1, sp2)
EXTENDS Parameter_2_Parameter(model, testSuite, interaction, testCase, message, assertion, method, p1, p2)
FORALL ::smc::SimpleAttribute sp1
  WHERE sp1.owner = p1
  MAKE ::xUnit::SimpleAttribute sp2
  sp2.name = sp1.name,
  sp2.type = sp1.type,
  sp2.setter = sp1.setter,
  p2.simpleAttribute = sp2;

// Rule: 8
RULE ComplexAttribute_2_Parameter(model, testSuite, interaction, testCase, message, assertion, method, p1, p2, cp1, cp2)
EXTENDS Parameter_2_Parameter(model, testSuite, interaction, testCase, message, assertion, method, p1, p2)
FORALL ::smc::ComplexAttribute cp1
  WHERE cp1.owner = p1
  MAKE ::xUnit::ComplexAttribute cp2
  cp2.name = cp1.name,
  cp2.type = cp1.type,
  cp2.setter = cp1.setter,
  p2.complexAttribute = cp2;

// Rule: 9
RULE SimpleAttribute_2_ComplexAttribute_Parameter(model, testSuite, interaction, testCase, message, assertion, method, p1, p2, cp1, cp2, scp1, scp2)
EXTENDS ComplexAttribute_2_Parameter(model, testSuite, interaction, testCase, message, assertion, method, p1, p2, cp1, cp2)
FORALL ::smc::SimpleAttribute scp1
  WHERE scp1.owner = cp1
  MAKE ::xUnit::SimpleAttribute scp2
  scp2.name = scp1.name,
  scp2.type = scp1.type,
  scp2.setter = scp1.setter,
  cp2.simpleAttribute = scp2;

// Rule: 10
RULE ComplexAttribute_2_ComplexAttribute_Parameter(model, testSuite, interaction, testCase, message, assertion, method, p1, p2, cp1, cp2, ccp1, ccp2)
EXTENDS ComplexAttribute_2_Parameter(model, testSuite, interaction, testCase, message, assertion, method, p1, p2, cp1, cp2)
FORALL ::smc::ComplexAttribute ccp1
  WHERE ccp1.owner = cp1
  MAKE ::xUnit::ComplexAttribute ccp2
  ccp2.name = ccp1.name,
  ccp2.type = ccp1.type,
  ccp2.setter = ccp1.setter,
  cp2.complexAttribute = ccp2;

// Rule: 11
RULE SimpleAttribute_2_ComplexComplex_Parameter(model, testSuite, interaction, testCase, message, assertion, method, p1, p2, cp1, cp2, scp1, scp2, s1, s2)
EXTENDS ComplexAttribute_2_ComplexAttribute_Parameter(model, testSuite, interaction, testCase, message, assertion, method, p1, p2, cp1, cp2, ccp1, ccp2)
FORALL ::smc::SimpleAttribute s1
WHERE s1.owner = ccp1
MAKE ::xUnit::SimpleAttribute s2
SET s2.name = s1.name,
    s2.type = s1.type,
    s2.setter = s1.setter,
    ccp2.simpleAttribute = s2;

// Rule: 12
RULE SimpleAttribute_2_ExpectedValue(model, testSuite, interaction, testCase, message, assertion, method, e1, e2, se1, se2)
EXTENDS ExpectedValue_2_ExpectedValue(model, testSuite, interaction, testCase, message, assertion, method, e1, e2)
FORALL ::smc::SimpleAttribute se1
WHERE se1.owner = e1
MAKE ::xUnit::SimpleAttribute se2
SET se2.name = se1.name,
    se2.type = se1.type,
    se2.setter = se1.setter,
    e2.simpleAttribute = se2;

// Rule: 13
RULE ComplexAttribute_2_ExpectedValue(model, testSuite, interaction, testCase, message, assertion, method, e1, e2, ce1, ce2)
EXTENDS ExpectedValue_2_ExpectedValue(model, testSuite, interaction, testCase, message, assertion, method, e1, e2)
FORALL ::smc::ComplexAttribute ce1
WHERE ce1.owner = e1
MAKE ::xUnit::ComplexAttribute ce2
SET ce2.name = ce1.name,
    ce2.type = ce1.type,
    ce2.setter = ce1.setter,
    e2.complexAttribute = ce2;

// Rule: 14
RULE SimpleAttribute_2_ComplexAttribute_ExpectedValue(model, testSuite, interaction, testCase, message, assertion, method, e1, e2, ce1, ce2, sce1, sce2)
EXTENDS ComplexAttribute_2_ExpectedValue(model, testSuite, interaction, testCase, message, assertion, method, e1, e2, ce1, ce2)
FORALL ::smc::SimpleAttribute sce1
WHERE sce1.owner = ce1
MAKE ::xUnit::SimpleAttribute sce2
SET sce2.name = sce1.name,
    sce2.type = sce1.type,
    sce2.setter = sce1.setter,
    ce2.simpleAttribute = sce2;

// Rule: 15
RULE ComplexAttribute_2_ComplexAttribute_ExpectedValue(model, testSuite, interaction, testCase, message, assertion, method, e1, e2, ce1, ce2, cce1, cce2)
EXTENDS ComplexAttribute_2_ExpectedValue(model, testSuite, interaction, testCase, message, assertion, method, e1, e2, ce1, ce2)
FORALL ::smc::ComplexAttribute cce1
WHERE cce1.owner = ce1
MAKE ::xUnit::ComplexAttribute cce2
SET cce2.name = cce1.name,
    cce2.type = cce1.type,
    cce2.setter = cce1.setter,
    ce2.complexAttribute = cce2;

// Rule: 16
RULE SimpleAttribute_2_ComplexComplex_ExpectedValue(model, testSuite, interaction, testCase, message, assertion, method, e1, e2, ce1, ce2, cce1, cce2, s1, s2)
EXTENDS ComplexAttribute_2_ComplexAttribute_ExpectedValue(model, testSuite, interaction, testCase, message, assertion, method, e1, e2, ce1, ce2, cce1, cce2)
FORALL ::smc::SimpleAttribute s1
WHERE s1.owner = cce1

MAKE ::xUnit::SimpleAttribute s2
SET s2.name = s1.name,
    s2.type = s1.type,
    s2.setter = s1.setter,
    cce2.simpleAttribute = s2;

RULE ConstructorParameter_2_ConstructorParameter(model, testSuite, interaction, testCase, message, 
    assertion, method, c1, c2, cp1, cp2)
EXTENDS OwnerClass_2_OwnerClass(model, testSuite, interaction, testCase, message, assertion, method, 
    c1, c2)
FORALL ::smc::ConstructorParameter cp1
    WHERE cp1.owner = c1
MAKE ::xUnit::ConstructorParameter cp2
    cp2.name = cp1.name,
    cp2.type = cp1.type,
    cp2.setter = cp1.setter,
    c2.constructorParameter = cp2;

RULE SimpleAttribute_2_ConstructorParameter(model, testSuite, interaction, testCase, message, 
    assertion, method, c1, c2, cp1, cp2, se1, se2)
EXTENDS ConstructorParameter_2_ConstructorParameter(model, testSuite, interaction, testCase, message, 
    assertion, method, c1, c2, cp1, cp2)
FORALL ::smc::SimpleAttribute se1
    WHERE se1.owner = cp1
MAKE ::xUnit::SimpleAttribute se2
    se2.name = se1.name,
    se2.type = se1.type,
    se2.setter = se1.setter,
    cp2.simpleAttribute = se2;

RULE ComplexAttribute_2_ConstructorParameter(model, testSuite, interaction, testCase, message, 
    assertion, method, c1, c2, cp1, cp2, ce1, ce2)
EXTENDS ConstructorParameter_2_ConstructorParameter(model, testSuite, interaction, testCase, message, 
    assertion, method, c1, c2, cp1, cp2)
FORALL ::smc::ComplexAttribute ce1
    WHERE ce1.owner = cp1
MAKE ::xUnit::ComplexAttribute ce2
    ce2.name = ce1.name,
    ce2.type = ce1.type,
    ce2.setter = ce1.setter,
    cp2.complexAttribute = ce2;

RULE SimpleAttribute_2_ComplexAttribute_ConstructorParameter(model, testSuite, interaction, testCase, message, 
    assertion, method, c1, c2, cp1, cp2, ce1, ce2, sce1, sce2)
EXTENDS ComplexAttribute_2_ConstructorParameter(model, testSuite, interaction, testCase, message, 
    assertion, method, c1, c2, cp1, cp2, ce1, ce2)
FORALL ::smc::SimpleAttribute sce1
    WHERE sce1.owner = ce1
MAKE ::xUnit::SimpleAttribute sce2
    sce2.name = sce1.name,
    sce2.type = sce1.type,
    sce2.setter = sce1.setter,
    ce2.simpleAttribute = sce2;
B.2: MOFScript rules (vertical transformation)

B.2.1: Rules for xUnit to JUnit

/* MOFScript: version 1.1.4
To deal with limitations of this version of MOFScript, the following three java methods are used in these rules:
1. writeHeader(): it simply copies the "CodeHeader" file to the top of the output file (the test case).
2. writeText(): it simply writes the test to the output file (the test case).
3. writeTestData(): it reads data from the "TestData" file and appends it in the output file (the test case).
*/
texttransformation testM2T (in model:xUnit)

testimonial testM2T (in model:xUnit)

/***************************************************************************/
_model.TestSuite::main() {
    paramHashtable.put("OutputFilePath", outputFilePath)
    paramHashtable.put("OutputFileName", "TestSuite_" + self.testCase.first().name)
    paramHashtable.put("OutputFileExtension", outputFileExtension)
    paramHashtable.put("TestDataFilePath", testDataFilePath)
    paramHashtable.put("TestDataFileName", testDataFileName + "_" + self.testCase.first().name)
    paramHashtable.put("TestDataFileExtension", testDataFileExtension)
    paramHashtable.put("HeaderFilePath", headerFilePath)
    paramHashtable.put("HeaderFileName", headerFileName)
    paramHashtable.put("HeaderFileExtension", headerFileExtension)
    // Copy Code_Header to the Test Case
    java ("Writer", "writeHeader", paramHashtable, "/UQ/project3/model2text")
    text = text + "package TestSuite_; "+ self.testCase.first().name + " extends TestCase {\n\n"
    text = text + "public static void main( String args[] ) { \n\n"
    text = text + "\tTestSuite testSuite = new TestSuite(" + self.testCase.first().name + ".class);\n"
    text = text + "\ttestSuite.run( new TestResult( ) );\n\n"
    paramHashtable.put("Text", text)
    java ("Writer", "writeText", paramHashtable, "/UQ/project3/model2text")
}
// Generate code for each test case
self.testCase.forEach(tc: model.TestCase) {
    tc.mapTestCase()
} text = "

// End of Class
";
paramHashtable.put("Text", text)
java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")

***************************************************************************
Rule 2: mapTestCase()
Description: This generates code for a test case.
Invoked from: main()
***************************************************************************
model.TestCase::mapTestCase() {
    variableMap = ""
    variableSuffix = 0
    testCaseCounter = 1;
    text = "\n\tpublic void test_" + self.name + "() {
\n\ttry {
    paramHashtable.put("Text", text)
    java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")
    text = ""
    paramHashtable.put("Interaction", self.name)
    counter = self.assertion.size()
    counter.forEach(c) {
        self.assertion.forEach(a: model.Assertion) {
            if(a.order.trim().equals(c)) {
                a.mapAssertion()
            }
        }
    }
\n\tcatch ( Exception exp ) {
\n\tSystem.out.println( exp.toString() );
\n\tfail("Exception occured during test case execution");
\n\t} //End of Method"
    paramHashtable.put("Text", text)
    java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")
} text = text + "\n\tcatch ( Exception exp ) {
\n\tSystem.out.println( exp.toString() );
\n\tfail("Exception occured during test case execution");
\n\t} //End of Method"
    paramHashtable.put("Text", text)
    java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")

***************************************************************************
Rule 3: mapAssertion()
Description: This rules creates text for assertion in the test case.
Invoked from: mapTestCase()
model.Assertion::mapAssertion() {
    assertionType = self.assertionType.trim()
    var accessModifier:String = self.method.first().accessModifier
    isEmptyExpected = self.expectedValue.isEmpty()

    self.method->forEach( m:model.Method | m = self.method.first()) {
        returnType = self.expectedValue.first().type.trim()
        if ( self.expectedValue.isEmpty() || self.expectedValue.first().name.equals("")) {
            returnVariable = returnType.firstToLower() + "_" + variableSuffix
            variableSuffix = variableSuffix + 1
        } else {
            returnVariable = self.expectedValue.first().name.trim()
        }
        expectedVariable = "expected_" + self.expectedValue.first().name.trim()
        typeCast = "(" + returnType + ")"
        assertion = ""
        isSetup = m.name.startsWith("SETUP_")
        isStatic = m.static.equalsIgnoreCase("yes")

        if (isStatic) {
            classReference = m.ownerClass.first().name.trim()
        } else if (m.ownerClass.first().objectName.trim().equals("")) {
            classReference = m.ownerClass.first().name.trim() + "_" + variableSuffix
        } else {
            classReference = m.ownerClass.first().objectName.trim()
        }

        // map method
        m.mapMethod()

        if ( not(isEmptyExpected || assertionType.equals("")) ) {
            self.expectedValue->forEach(p:model.ExpectedValue) {
                p.mapExpectedValue()
            }
        }

        // Store the returnVariable in the map for existing variables
        variableMap.put(returnVariable.trim(), returnVariable.trim())
    }
    if(not isSetup) {
        if (isEmptyExpected) {
            returnType = ""
            returnVariable = ""
        } else {
            returnType = self.expectedValue.first().type.trim()
            returnVariable = returnType.firstToLower() + "_" + variableSuffix
        }
    }
}
typeCast = ''
assertion = ''
}
text = '''
if (accessModifier.equals("final")) {
    text = '\n\t' + accessModifier + '' + returnType + '' + returnVariable + ' = new ' + returnType + '(');
}

if (returnVariable.equals('') || (not accessModifier.equals(''))) {
    text = text + '\n\t' + returnType + '' + returnVariable + '' + typeCast + classReference + '.' + self.method.first().name.trim() + '(' + parameterText + ');''
} else {
    text = text + '\n\t' + returnType + '' + returnVariable + '' + typeCast + classReference + '.' + self.method.first().name.trim() + '(' + parameterText + ');''
}

if (isStatic || assertionType.equals('')) {
    assertion = ''
} else if (returnType.equalsIgnoreCase("int") || returnType.equalsIgnoreCase("long") || returnType.equalsIgnoreCase("float") || returnType.equalsIgnoreCase("double") || returnType.equalsIgnoreCase("char") ) {
    typeCast = ''
    assertion = "assertTrue( " + returnVariable + " == " + expectedVariable + ");''
    text = text + '\n\t' + assertion + ' ' + testCaseCounter + '
    testCaseCounter = testCaseCounter + 1
} else {
    assertion = "assertTrue( " + returnVariable + " .equals( " + expectedVariable + ");''
    text = text + '\n\t' + assertion + ' ' + testCaseCounter + '
    testCaseCounter = testCaseCounter + 1
}
}
paramHashtable.put("Text", text)
java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")

text = '''
}
paramHashtable.put("Message", self.ownerClass.first().name.trim() + ":" + self.name.trim())

storedClassReference = ""
key = self.ownerClass.first().objectName

if not key.equals(""") {
    storedClassReference = variableMap.get(key.trim())
}

if storedClassReference.equals(""") {
    self.ownerClass->forEach(cn: model.OwnerClass | cn = self.ownerClass.first()) {
        cn.mapOwnerClass()
    }
    if (isStatic == false) {
        if (isArray) {
            text = text + "\t\t" + returnType + "[]" + classReference + " = new " + returnType + "[] { " + cParameterText + " };"
        } else {
            text = text + "\t\t" + returnType + "" + classReference + " = new " + returnType + "{" + cParameterText + " };"
        }
        cParameterVariable = self.name + "" + variableSuffix
    }
    paramHashtable.put("Text", text)
    java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/"
    variableMap.put(classReference.trim(), classReference.trim())
}

text = ""
parameterText = ""
parameterNumber = 1

if not isSetup) {
    self.parameter->forEach(p: model.Parameter) {
        p.mapParameter( parameterNumber )
        parameterNumber = parameterNumber + 1
    }
}

if not self.freeParameterText.trim().equals("") parameterText = self.freeParameterText
Rule 5: mapOwnerClass()
Description: This rules creates an object on which the method is invoked.
Invoked from: mapMethod()
***************************************************************************/
model.OwnerClass:mapOwnerClass() {
  cParameterText = ""
  cParameterNumber = 1
  self.constructorParameter->forEach(cp:model.ConstructorParameter) {
    cp.mapConstructorParameter(cParameterNumber)
    cParameterNumber = cParameterNumber + 1
  }
}
***************************************************************************/
Rule 6: mapConstructorParameter(cParameterNumberStr: String)
Description: This rules creates constructor parameter.
Invoked from: mapMethod()
***************************************************************************/
model.ConstructorParameter:mapConstructorParameter(cParameterNumberStr: String) {
  cParameterSeperator = ", "
  if (cParameterNumber == 1) cParameterSeperator = ""
  cParameterType = self.type
  cParameterVariable = self.name
  storedParamVariable = variableMap.get(cParameterVariable.trim())

  // read previous parameter
  if (storedParamVariable.trim().equals("")) {
    cParameterVariable = self.name
    variableMap.put(cParameterVariable.trim(), cParameterVariable.trim())
  } else {
    cParameterVariable = storedParamVariable
  }

  cParameterText = cParameterText + cParameterSeperator + self.name

  if (storedParamVariable.trim().equals("")) {
    if (self.simpleAttribute.isEmpty() && self.complexAttribute.isEmpty()) {
      text = text + "\n\n" + cParameterType + " " + cParameterVariable + " = "
      if (self.type.equalsIgnoreCase("String")) text = text + "\n"
      paramHashtable.put("Text", text)
java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")
text = "\n"

paramHashtable.put("Type", "ConstructorParameter")
paramHashtable.put("Name", self.name.trim())
java ("Writer", "writeTestData", paramHashtable, "E:/UQ/project3/model2text/")
text = ";"
if (self.type.equalsIgnoreCase("String")) text = "\";"
paramHashtable.put("Text", text)
java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")
text = ""}
else {
    text = text + "\n\n\n" + cParameterType + " = new " + cParameterType + "();"
paramHashtable.put("Text", text)
java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")
text = "\n"
}

complexAttributeCounter = 1
self.complexAttribute->forEach (ca:model.ComplexAttribute) {
    ca.mapComplexAttribute(cParameterNumberStr, cParameterVariable, complexAttributeCounter)
    complexAttributeCounter = complexAttributeCounter + 1
}

simpleAttributeCounter = 1
self.simpleAttribute->forEach (sa:model.SimpleAttribute) {
    sa.mapSimpleAttribute(cParameterNumberStr, cParameterVariable, simpleAttributeCounter)
    simpleAttributeCounter = simpleAttributeCounter + 1
}
}

/**********************************************************
Rule 7: mapExpectedValue()
Description: This rules creates expected value for a method call.
Invoked from mapExpectedValue()
**********************************************************/
model.ExpectedValue::mapExpectedValue() {
    expectedType = self.type
    expectedVariable = "expected_" + self.name
    if (self.name.equals("")) {
        returnVariable = expectedType.toLowerCase() + "=" + variableSuffix
    } else {
        returnVariable = expectedType + "=" + variableSuffix
    }
}
returnVariable = self.name
}
variableMap.put( returnVariable.trim(), returnVariable.trim() )
paramHashtable.put("Type", "ExpectedValue")

if ( self.simpleAttribute.isEmpty() && self.complexAttribute.isEmpty()) {
    text = text + "\n\n" + expectedType + " + " + expectedVariable + " = "
    if (self.type.equalsIgnoreCase("String"))
        paramHashtable.put("Text", text)
        java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")
        paramHashtable.put("Name", "expected. " + self.name.trim())
        java ("Writer", "writeTestData", paramHashtable, "E:/UQ/project3/model2text/")
    text = ";"
    if (self.type.equalsIgnoreCase("String"))
        paramHashtable.put("Text", text)
        java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")
} else {
    text = text + "\n\n" + expectedType + " + " + expectedVariable + " = new " + expectedType + "();"
    paramHashtable.put("Text", text)
    java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")
}
complexAttributeCounter = 1
self.complexAttribute->forEach( ca:model.ComplexAttribute ) {
    ca.mapComplexAttribute("1", expectedVariable, complexAttributeCounter)
    complexAttributeCounter = complexAttributeCounter + 1
}
simpleAttributeCounter = 1
self.simpleAttribute->forEach( sa:model.SimpleAttribute ) {
    sa.mapSimpleAttribute("1", expectedVariable, simpleAttributeCounter)
    simpleAttributeCounter = simpleAttributeCounter + 1
}
parameterSeparator = "," 
if (parameterNumber == 1) parameterSeparator = ""

storedParamVariable = "" 
parameterVariable = ""

parameterType = self.type 
parameterVariable = "expected_" + self.name.trim() 

storedParamVariable = variableMap.get(self.name.trim()) 
if (storedParamVariable.equals("")) { 
    storedParamVariable = variableMap.get(parameterVariable.trim())  
    variableMap.put(parameterVariable.trim(), parameterVariable.trim())
} else {
    parameterVariable = storedParamVariable
}

parameterText = parameterText + parameterSeparator + parameterVariable

if (storedParamVariable.trim().equals("")) {
    text = ""
    if (self.simpleAttribute.isEmpty() && self.complexAttribute.isEmpty()) {
        text = text + "n\n\n\n" + parameterType + " " + parameterVariable + " = "
        if (self.type.equalsIgnoreCase("String")) text = text + "\"
        paramHashtable.put("Text", text)
        java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")
        text = "\n"
        paramHashtable.put("Type", "Parameter") 
        paramHashtable.put("Name", self.name.trim())
        java ("Writer", "writeTestData", paramHashtable, "E:/UQ/project3/model2test/")
        text = ":\n"
        if (self.type.equalsIgnoreCase("String")) text = "\":
        paramHashtable.put("Text", text)
        java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")
        text = ""
    } else {
        text = text + "n\n\n\n" + parameterType + " " + parameterVariable + " = new " + parameterType + "();"
        paramHashtable.put("Text", text)
        java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2test/")
        text = "\n"
    }
} else { 
    text = text + "n\n\n\n" + parameterType + " " + parameterVariable + " = new " + parameterType + "];
    paramHashtable.put("Text", text)
    java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2test/")
    text = "\n"}
complexAttributeCounter = 1
self.complexAttribute->forEach( ca:model.ComplexAttribute ) {
    ca.mapComplexAttribute(parameterNumberStr, parameterVariable, complexAttributeCounter)
    complexAttributeCounter = complexAttributeCounter + 1
}

simpleAttributeCounter = 1
self.simpleAttribute->forEach( sa:model.SimpleAttribute ) {
    sa.mapSimpleAttribute(parameterNumberStr, parameterVariable, simpleAttributeCounter)
    simpleAttributeCounter = simpleAttributeCounter + 1
}
}

/** ************************************************************************************
Rule 9: mapComplexAttribute()
Description: This rule creates complex objects.
Invoked from: mapParameter(), mapExpectedValue(), mapConstructorParameter()
*******************************************************************************/
model.ComplexAttribute::mapComplexAttribute( parameterNumberStr:String, parentVariable:String, complexAttributeCounterStr:String) {
    
    complexAttributeType = self.type
    complexAttributeVariable = parentVariable + "_" + complexAttributeType.firstToLower() + complexAttributeCounterStr.firstToLower()
    storedParamVariable = variableMap.get( self.name )
    if ( storedParamVariable.equals("") ) {
        text = text + 
        complexAttributeType + "" + complexAttributeVariable + " = new " + complexAttributeType + "();"
        paramHashtable.put("Text", text)
        java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")
        complexAttributeCounter2 = 1
        self.complexAttribute->forEach( sa:model.ComplexAttribute ) {
            sa.mapComplexAttribute(parameterNumberStr, parameterVariable + "_" + self.name.trim(), complexAttributeCounter2)
            complexAttributeCounter2 = complexAttributeCounter2 + 1
        }
    }
    else {
        complexAttributeVariable = self.name
    }
}

200
if (self.setter==null) {
    setter = "set" + self.name.firstToUpper()
} else {
    setter = self.setter
}

text = "\n\n" + parentVariable + "," + setter + "( " + complexAttributeVariable + ");" + 
paramHashtable.put("Text", text)
java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")
text = ""

/*******************************************************************************
Rule 10: mapSimpleAttribute()  
Description: This rules creates simple objects. 
Invoked from: mapParameter(), mapExpectedValue(), mapConstructorParameter(), mapComplexParameter() 
*******************************************************************************/

model.SimpleAttribute::mapSimpleAttribute(parameterNumberStr:String, parentVariable:String, simpleAttributeCounterStr:String) {

    text = ""
    simpleAttributeType = self.type
    simpleAttributeVariable = parentVariable + "," + simpleAttributeType.firstToUpper() + simpleAttributeCounterStr

    text = text + "\n\n"+ simpleAttributeType + "," + simpleAttributeVariable + ":" + 

    if (self.type.equalsIgnoreCase("String")) text = text + "\n"
    paramHashtable.put("Text", text)
    java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")

    paramHashtable.put("Name", parentVariable + "," + self.name.trim())
    java ("Writer", "writeTestData", paramHashtable, "E:/UQ/project3/model2text/")

    text = ";" 
    if (self.type.equalsIgnoreCase("String")) text = ";;"
    paramHashtable.put("Text", text)
    java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")

        text = ""
    else {
        setter = self.setter
        
    text = "\n\n" + parentVariable + "," + setter + "(" + simpleAttributeVariable + ");" + 
    paramHashtable.put("Text", text)
    java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")
}
B.2.2: Rules for xUnit to SUnit

/*
MOFScript: version 1.1.4
To deal with limitations of this version of MOFScript, following three java methods are used in these rules:
1. writeHeader(): it simply copies the "CodeHeader" file to the top of the output file (the test case).
2. writeText(): it simply writes the test to the output file (the test case).
3. writeTestData(): it reads data from the "TestData" file and appends it in the output file (generated test case).
*/
texttransformation testM2T (in model:xUnit)

/***************************************************************************
Rule 1:
Description: This initiates the model-to-text transformation.
Invoked from: none
***************************************************************************/
model.TestSuite::main() {
    paramHashtable.put("OutputFilePath", outputFilePath)
    paramHashtable.put("OutputFileName", "TestSuite_"+ self.testCase.first().name)
    paramHashtable.put("OutputFileExtension", outputFileExtension)
    paramHashtable.put("TestDataFilePath", testDataFilePath)
    paramHashtable.put("TestDataFileName", testDataFileName + "." + self.testCase.first().name)
    paramHashtable.put("TestDataFileExtension", testDataFileExtension)
    paramHashtable.put("HeaderFilePath", headerFilePath)
    paramHashtable.put("HeaderFileName", headerFileName)
    paramHashtable.put("HeaderFileExtension", headerFileExtension)

    // Copy Code_Header to the Test Case
    java ("Writer", "writeHeader", paramHashtable, "E:/UQ/project3/model2text/")

    // Generate code for each test case
    self.testCase->forEach(tc:model.TestCase) {
        text = "\n\ntClass: Test_" + tc.name.trim()
        text = text + "\n\tsuperclass: TestCase"
        paramHashtable.put("Text", text)
        java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")
    }
}
text = ""

varMap = ""

varSuffix = 0

testCaseCounter = 1

testCaseName = self.name.trim()

paramHashtable.put("Interaction", self.name)

counter = self.assertion.size()

counter-1

forall(c) {
  self.assertion-1
  if(a.order.trim().equals(c) {
    a.mapAssertion()
  }
}

testCaseName = self.name.trim()
Rule 3: mapAssertion()

Description: This rule creates text for assertions in the test case.

Invoked from: mapTestCase()

***************************************************************************/

model.Assertion::mapAssertion() {

    assertionType = self.assertionType.trim()
    var accessModifier:String = self.method.first().accessModifier
    isEmptyExpected = self.expectedValue.isEmpty()

    self.method->forEach( m:model.Method | m = self.method.first()) {

        returnType = self.expectedValue.first().type.trim()
        if ( self.expectedValue.isEmpty() || self.expectedValue.first().name.equals("") ) {
            returnVariable = returnType.firstToLower() + "_" + variableSuffix
            variableSuffix = variableSuffix + 1
        } else {
            returnVariable = self.expectedValue.first().name.trim()
        }

        expectedVariable = "expected_" + self.expectedValue.first().name.trim()
        typeCast = "(" + returnType + ")"
        assertion = ""

        isSetup = m.name.startsWith("SETUP_")
        isStatic = m.static.equalsIgnoreCase("yes")

        if (isStatic) {
            classReference = m.ownerClass.first().name.trim()
        } else if (m.ownerClass.first().objectName.trim().equals("") ) {
            classReference = m.ownerClass.first().name.trim() + "_" + variableSuffix
        } else {
            classReference = m.ownerClass.first().objectName.trim()
        }

        // map method
        m.mapMethod()

        if ( not(isEmptyExpected || assertionType.equals("")) ) {
            self.expectedValue->forEach(p:model.ExpectedValue) {
                p.mapExpectedValue()
            }
        }
    }

    variableMap.put(returnVariable.trim(), returnVariable.trim())

    if(not isSetup) {
        if (isEmptyExpected) {

        } else {

        }

    }

}
returnType = ""
returnVariable = ""
typeCast = ""
assertion = ""
}
text = ""
if (accessModifier.equals("final")) {
    text = "\t" + accessModifier + " returnVariable + := " + returnType + " new.";
}
if (not parameterText.equals("")) parameterText = " " + parameterText
if (returnVariable.equals("")) {
    text = text + "\t" + returnVariable + " = " + returnType + " new.";
}
else {
    text = text + "\t" + returnVariable + " = " + returnType + " new.";
}
if (isStatic || assertionType.equals("")) {
    assertion = ""
} else if (returnType.equalsIgnoreCase("int") || returnType.equalsIgnoreCase("long") || returnType.equalsIgnoreCase("float") || returnType.equalsIgnoreCase("double") || returnType.equalsIgnoreCase("char")) {
    typeCast = ""
    text = text + "\t" + returnVariable + " = " + returnType + " new.";
    testCaseCounter = testCaseCounter + 1
} else {
    typeCast = ""
    text = text + "\t" + returnVariable + " = " + returnType + " new.";
    testCaseCounter = testCaseCounter + 1
}
paramHashtable.put("Text", text)
java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")
}

/***************************************************************************
Rule 4:  mapMethod()
Description:  This rules creates text for method invocation.
Invoked from:  mapAssertion()
***************************************************************************/
model.Method::mapMethod() {
    var returnType: String = self.ownerClass.first().name
}
var isArray: boolean = self.ownerClass().array.equals("yes")

paramHashtable.put("Message", self.ownerClass().name().trim() + ":" + self.name().trim())

storedClassReference = ""
key = self.ownerClass().objectName

if( not key.equals("")) {
    storedClassReference = variableMap.get(key.trim())
}

if( storedClassReference.equals("")) {
    self.ownerClass().forEach(cn: model.OwnerClass | cn = self.ownerClass().first()) {
        cn.mapOwnerClass()
    }
    if( not cParameterText.equals("")) cParameterText = " " + cParameterText
    if( isStatic == false ) {
        text = text + 
            "\n\n" + classReference + ":" + returnType + " new" + cParameterText + ","
        cParameterVariable = self.name + "," + variableSuffix
    }
    paramHashtable.put("Text", text)
    java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/"
    variableMap.put(classReference.trim(), classReference.trim())
}

text = ""
parameterText = ""
parameterNumber = 1

if( not isSetup ) {
    if( not assertionType.equals("")) {
        text = "\n\n" + testCaseName + " >>test_" + self.name
        paramHashtable.put("Text", text)
        java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/"
        testCasesToAdd = testCasesToAdd + ( 
            "tsuite addTestCase:" + testCaseName + ": selector: #" + "test" + self.name + ":"
        )
        text = ""
    }
    self.parameter().forEach(p: model.Parameter) {
        p.mapParameter( parameterNumber )
        parameterNumber = parameterNumber + 1
    }
}
if( not self.freeParameterText.trim().equals("") ) parameterText = self.freeParameterText

/***************************************************************************
Rule 5: mapOwnerClass()
Description: This rules creates an object on which the method is invoked.
Invoked from: mapMethod()
***************************************************************************/
model.OwnerClass::mapOwnerClass() {
  cParameterText = ""
  cParameterNumber = 1
  self.constructorParameter->forEach(cp:model.ConstructorParameter) {
    cp.mapConstructorParameter( cParameterNumber )
    cParameterNumber = cParameterNumber + 1
  }
}

/***************************************************************************
Rule 6: mapConstructorParameter()
Description: This rules creates constructor parameter.
Invoked from: mapMethod()
***************************************************************************/
model.ConstructorParameter::mapConstructorParameter( cParameterNumberStr: String ) {
  cParameterSeparator = " with "
  if( cParameterNumber == 1 ) cParameterSeparator = ""

  cParameterType = self.type
  cParameterVariable = self.name

  storedParamVariable = variableMap.get( cParameterVariable.trim() )

  if( storedParamVariable.trim().equals("") ) {
    cParameterVariable = self.name
    variableMap.put( cParameterVariable.trim(), cParameterVariable.trim() )
  } else {
    cParameterVariable = storedParamVariable
  }

  cParameterText = cParameterText + cParameterSeparator + self.name

  if( storedParamVariable.trim().equals("") ) {
    if( self.simpleAttribute.isEmpty() && self.complexAttribute.isEmpty() ) {
      text = text + "\n\n\n" + cParameterVariable + " := " + cParameterType + " new." 
      text = text + "\n\n\n" + cParameterVariable + " := "
    }
  }

if (self.type.equalsIgnoreCase("String")) text = text + "\n"
paramHashtable.put("Text", text)
java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")
paramHashtable.put("Type", "ConstructorParameter")
paramHashtable.put("Name", self.name.trim())
java ("Writer", "writeTestData", paramHashtable, "E:/UQ/project3/model2text/")

\n
}\nelse {
  text = text + "\n\nnew."
paramHashtable.put("Text", text)
java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")
}

}\n
complexAttributeCounter = 1
self.complexAttribute->forEach( ca:model.ComplexAttribute ) {
  ca.mapComplexAttribute(cParameterNumberStr, cParameterVariable, complexAttributeCounter)
  complexAttributeCounter = complexAttributeCounter + 1
}

simpleAttributeCounter = 1
self.simpleAttribute->forEach( sa:model.SimpleAttribute ) {
  sa.mapSimpleAttribute(cParameterNumberStr, cParameterVariable, simpleAttributeCounter)
  simpleAttributeCounter = simpleAttributeCounter + 1
}

}
) else {
    returnVariable = self.name
}

variableMap.put( returnVariable.trim(), returnVariable.trim() )

paramHashtable.put("Type", "ExpectedValue")

if ( self.simpleAttribute.isEmpty() && self.complexAttribute.isEmpty() ) {
    text = text + "\n\n" + expectedVariable + ":= " + expectedType + " new."  
    text = text + "\n\n"
    paramHashtable.put("Text", text)
    java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/"

    paramHashtable.put("Type", "ExpectedValue")
    paramHashtable.put("Name", "expected_" + self.name.trim())
    java ("Writer", "writeTestData", paramHashtable, "E:/UQ/project3/model2text/"

    text = "."
    if (self.type.equalsIgnoreCase("String")) text = "."
    paramHashtable.put("Text", text)
    java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/"

}) else {
    text = text + "\n\n" + expectedVariable + ":= " + expectedVariable + " new."  
    paramHashtable.put("Text", text)
    java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/"

} else {
    text = text + "\n\n" + expectedVariable + ":= " + expectedVariable + " new."  
    paramHashtable.put("Text", text)
    java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/"

complexAttributeCounter = 1
self.complexAttribute->forEach( ca:model.ComplexAttribute ){
    ca.mapComplexAttribute("1", expectedVariable, complexAttributeCounter)
    complexAttributeCounter = complexAttributeCounter + 1
}

simpleAttributeCounter = 1
self.simpleAttribute->forEach( sa:model.SimpleAttribute ){
    sa.mapSimpleAttribute("1", expectedVariable, simpleAttributeCounter)
    simpleAttributeCounter = simpleAttributeCounter + 1
}
/***************************************************************************
Rule 8: mapParameter()
Description: This rules creates parameter for a method call.
Invoked from: mapMethod()
***************************************************************************

model.Parameter::mapParameter( parameterNumberStr:String ) {
    parameterSeparator = " with "
    if( parameterNumber == 1 ) parameterSeparator = ""

    storedParamVariable = "" 
    parameterVariable = ""
    parameterType = self.type 
    parameterVariable = "expected_" + self.name.trim()

    storedParamVariable = variableMap.get( self.name.trim() )
    if( storedParamVariable.equals( "" ) ) { 
        storedParamVariable = variableMap.get( parameterVariable.trim() )
        parameterVariable = self.name 
        variableMap.put( parameterVariable.trim(), parameterVariable.trim() )
    } else {
        parameterVariable = storedParamVariable
    }

    parameterText = parameterText + parameterSeparator + parameterVariable

    if( storedParamVariable.trim().equals( "" ) ) {
        text = ""
        if( self.simpleAttribute.isEmpty() && self.complexAttribute.isEmpty() ) {
            text = text + "\"}" + parameterVariable + ":= " + parameterType + " new." 
            text = text + "\"}" + parameterVariable + ":= " 
            if( self.type.equalsIgnoreCase("String") ) text = text + "\""
            paramHashtable.put("Text", text)
            java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")
            text = "\n"
            paramHashtable.put("Type", "Parameter")
            paramHashtable.put("Name", self.name.trim())
            java ("Writer", "writeTestData", paramHashtable, "E:/UQ/project3/model2text/")
            text = "."
            if( self.type.equalsIgnoreCase("String") ) text = "\"."
            paramHashtable.put("Text", text)
            java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")
        }
    } else { 
    }
}
text = ""

} else {
    text = text + \n\t+ parameterVariable + ":= " + parameterType + " new."
    paramHashtable.put("Text", text)
    java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/"
    text = "\n"
}

complexTypeCounter = 1
self.complexAttribute->forEach { ca:model.ComplexAttribute } {
    ca.mapComplexAttribute(parameterNumberStr, parameterVariable, complexAttributeCounter)
    complexAttributeCounter = complexAttributeCounter + 1
}

simpleAttributeCounter = 1
self.simpleAttribute->forEach { sa:model.SimpleAttribute } {
    sa.mapSimpleAttribute(parameterNumberStr, parameterVariable, simpleAttributeCounter)
    simpleAttributeCounter = simpleAttributeCounter + 1
}

/
***************************************************************************************/

Rule 9: mapComplexAttribute()
Description: This rules creates complex objects.
Invoked from: mapParameter(), mapExpectedValue(), mapConstructorParameter()
***************************************************************************************/

model.ComplexAttribute::mapComplexAttribute( parameterNumberStr: String, parentVariable: String, complexAttributeCounterStr: String ) {
    complexAttributeType = self.type
    complexAttributeVariable = parentVariable + "_" + complexAttributeType.firstToLower() + complexAttributeCounterStr.firstToLower()
    storedParamVariable = variableMap.get( self.name )
    if ( storedParamVariable.equals("" ) ) {
        text = text + \n\t+ complexAttributeVariable + ":= " + complexAttributeType + " new."
        paramHashtable.put("Text", text)
        java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/"
        complexAttributeCounter2 = 1
        self.complexAttribute->forEach { sa:model.ComplexAttribute } {
            sa.mapComplexAttribute(parameterNumberStr, parentVariable + "_" + self.name.trim(), complexAttributeCounter2)
            complexAttributeCounter2 = complexAttributeCounter2 + 1
        }
    }
}
simpleAttributeCounter = 1

self.simpleAttribute.forEach(sa:model.SimpleAttribute) {
    sa.mapSimpleAttribute(parameterNumberStr, parentVariable + "+" + self.name.trim(), simpleAttributeCounter)
    simpleAttributeCounter = simpleAttributeCounter + 1
}

else {
    complexAttributeVariable = self.name
}

if (self.setter == null) {
    setter = "set" + self.name.firstToUpper()
} else {
    setter = self.setter
}

text = "\n\n\n\n" + parentVariable + " " + setter + " " + complexAttributeVariable + ""

paramHashtable.put("Text", text)
java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")

text = ""

model.SimpleAttribute::mapSimpleAttribute(parameterNumberStr:String, parentVariable:String, simpleAttributeCounterStr:String) {
    text = ""
    simpleAttributeType = self.type
    simpleAttributeVariable = parentVariable + "+" + simpleAttributeType.firstToUpper() + simpleAttributeCounterStr
    text = text + "\n\n\n\n" + simpleAttributeVariable + ":=" + simpleAttributeType + " new."
    text = text + "\n\n\n\n" + simpleAttributeVariable + ":="

    if (self.type.equalsIgnoreCase("String")) text = text + "\n\n\n\n"
    paramHashtable.put("Text", text)
    java ("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/")

    paramHashtable.put("Name", parentVariable + "+" + self.name.trim())
    java ("Writer", "writeTestData", paramHashtable, "E:/UQ/project3/model2text/")

    text = "\n\n\n\n"
    if (self.type.equalsIgnoreCase("String")) text = "\n\n\n\n"
paramHashtable.put("Text", text)
java("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/"

if (self.setter==null) {
    setter = "set" + self.name.firstToUpper()
} else {
    setter = self.setter
}

text = "\n\n" + parentVariable + " " + setter + " " + simpleAttributeVariable + ".
" paramHashtable.put("Text", text)
java("Writer", "writeText", paramHashtable, "E:/UQ/project3/model2text/"
text = ""
}
B.3: JUnit test cases generated by MTCG

B.3.1: JUnit test case for withdrawal operation

```java
import atm.*;
import atm.physical.*;
import atm.transaction.*;
import banking.*;
import simulation.*;
import junit.framework.*;
import java.awt.*;
import java.awt.event.*;
import java.net.InetAddress;

public class TestSuite_withdrawal extends TestCase {

    public static void main( String args[] ) {
        TestSuite testSuite = new TestSuite(TestSuite_withdrawal.class);
        testSuite.run( new TestResult() );
    }

    public void test_withdrawal() {
        try {
            int id = 41;
            String place = "Gorden College";
            String name = "National Bank";
            InetAddress address = null;
            ATM aTM = new ATM( id, place, name, address );
            Session session = new Session( aTM);
            Simulation simulation = new Simulation( aTM);
            Card expected_card = new Card();
            int expected_card_Int1 = 1;
            expected_card.setNumber( expected_card_Int1 );
            Card card = (Card) simulation.readCard(  );
            assertTrue( card.equals( expected_card ) );
            // Test Case # 1

            CustomerConsole customerConsole = new CustomerConsole(  );
            String prompt = "Please enter PIN."
            int expected_pin = 42;
            int pin = (int) customerConsole.readPIN( prompt );
            assertTrue( pin == expected_pin );
            // Test Case # 2

            Transaction.makeTransaction(aTM, session, card, pin);
            Withdrawal withdrawal = new Withdrawal( aTM, session, card, pin );
            withdrawal.performTransaction();
            simulation.ejectCard();
        }
        catch ( Exception exp ) {
            System.out.println( exp.toString() );
            fail("Exception occured during test case execution");
        }
    }

    } //End of Method

} //End of Class
```

B.3.2: JUnit test case for deposit operation

```java
import atm.*;
import atm.physical.*;
import atm.transaction.*;
import banking.*;
import simulation.*;
import junit.framework.*;
import java.awt.*;
import java.awt.event.*;
import java.net.InetAddress;

public class TestSuite_deposit extends TestCase {

    public static void main( String args[] ) {
        TestSuite testSuite = new TestSuite(TestSuite_deposit.class);
        testSuite.run( new TestResult( ) );
    }

    public void test_deposit( ) {
        try {
            int id = 41;
            String place = "Gorden College";
            String name = "National Bank";
            InetAddress address = null;
            ATM aTM = new ATM( id, place, name, address );
            Session session = new Session( aTM );
            Simulation simulation = new Simulation( aTM );
            Card expected_card = new Card();
            int expected_card_Int1 = 1;
            expected_card.setNumber( expected_card_Int1 );
            Card card = (Card) simulation.readCard( );
            assertTrue( card.equals( expected_card ) );
            // Test Case # 1

            CustomerConsole customerConsole = new CustomerConsole( );
            String prompt = "Please enter PIN.";
            int expected_pin = 42;
            int pin = (int) customerConsole.readPIN( prompt );
            assertTrue( pin == expected_pin );
            // Test Case # 2

            Transaction.makeTransaction(aTM, session, card, pin);
            Deposit deposit = new Deposit( aTM, session, card, pin );
            deposit.performTransaction();
            simulation.ejectCard();
        } catch ( Exception exp ) {
            System.out.println( exp.toString() );
            fail("Exception occurred during test case execution");
        }
    }
}
```

//End of Class
B.3.3: JUnit test case for transfer operation

```java
import atm.*;
import atm.physical.*;
import atm.transaction.*;
import banking.*;
import simulation.*;
import junit.framework.*;
import java.awt.*;
import java.awt.event.*;
import java.net.InetAddress;

public class TestSuite_transfer extends TestCase {

    public static void main( String args[] ) {
        TestSuite testSuite = new TestSuite(TestSuite_transfer.class);
        testSuite.run( new TestResult( ) );
    }

    public void test_transfer() {
        try {
            int id = 41;
            String place = "Gorden College";
            String name = "National Bank";
            InetAddress address = null;
            ATM aTM = new ATM( id, place, name, address );
            Session session = new Session( aTM );
            Simulation simulation = new Simulation( aTM );
            Card expected_card = new Card();
            int expected_card_Int1 = 1;
            expected_card.setNumber(expected_card_Int1);
            Card card = (Card) simulation.readCard();
            assertTrue( card.equals(expected_card) ); // Test Case # 1

            CustomerConsole customerConsole = new CustomerConsole();
            String prompt = "Please enter PIN."
            int expected_pin = 42;
            int pin = (int) customerConsole.readPIN( prompt );
            assertTrue( pin == expected_pin ); // Test Case # 2

            Transaction.makeTransaction(aTM, session, card, pin);
            Transfer transfer = new Transfer(aTM, session, card, pin );
            transfer.performTransaction();
            simulation.ejectCard();
        } catch ( Exception exp ) {
            System.out.println( exp.toString() );
            fail("Exception occured during test case execution");
        }
    }

} //End of Method

} //End of Class
```

Model-Driven Framework for Context Dependent Testing of Components 216
B.3.4: JUnit test case for balance inquiry operation

```java
import atm.*;
import atm.physical.*;
import atm.transaction.*;
import banking.*;
import simulation.*;
import junit.framework.*;
import java.awt.*;
import java.awt.event.*;
import java.net.InetAddress;

public class TestSuite_inquiry extends TestCase {

    public static void main( String args[] ) {
        TestSuite testSuite = new TestSuite(TestSuite_inquiry.class);
        testSuite.run( new TestResult() );
    }

    public void test_inquiry() {
        try {
            int id = 41;
            String place = "Gorden College";
            String name = "National Bank";
            InetAddress address = null;
            ATM aTM = new ATM( id, place, name, address );
            Session session = new Session( aTM );
            Simulation simulation = new Simulation( aTM );
            Card expected_card = new Card();
            int expected_card_Int1 = 1;
            expected_card.setNumber( expected_card_Int1 );
            Card card = (Card) simulation.readCard();
            assertTrue( card.equals( expected_card ) ); // Test Case # 1
            CustomerConsole customerConsole = new CustomerConsole( );
            String prompt = "Please enter PIN."
            int expected_pin = 42;
            int pin = (int) customerConsole.readPIN( prompt );
            assertTrue( pin == expected_pin ); // Test Case # 2
            Transaction.makeTransaction(aTM, session, card, pin);
            Inquiry inquiry = new Inquiry( aTM, session, card, pin );
            inquiry.performTransaction();
            simulation.ejectCard();
        } catch( Exception exp ) {
            System.out.println( exp.toString() );
            fail("Exception occured during test case execution");
        }
    } //End of Method
}
//End of Class
```
B.4: SUnit test cases generated by MTCG

B.4.1: SUnit test case for withdrawal operation

```java
import atm.*;
import atm.physical.*;
import atm.transaction.*;
import banking.*;
import simulation.*;
import junit.framework.*;
import java.awt.*;
import java.awt.event.*;
import java.net.InetAddress;

Class: Test_withdrawal
    superclass: TestCase
    id := int new.
    id := 41.
    place := String new.
    place := "Gorden College".
    name := String new.
    name := "National Bank".
    address := InetAddress new.
    address := null.
    aTM := ATM new id with place with name with address.
    session := Session new aTM.
    simulation := Simulation new aTM.

    Test_withdrawal>>test_readCard
        expected_card:= expected_card new.
        expected_card_Int1:=int new.
        expected_card_Int1:=1.
        expected_card setNumber expected_card_Int1.
        card = (Card) simulation readCard.
        self assert: (card equals expected_card ).

    customerConsole := CustomerConsole new.

    Test_withdrawal>>test_readPIN
        prompt:= String new.
        prompt:= "Please enter PIN."
        expected_pin:= int new.
        expected_pin := 42.
        pin = (int) customerConsole readPIN prompt.
        self assert: (pin = expected_pin ).

    Transaction makeTransaction aTM with session with card with pin.
    withdrawal := Withdrawal new aTM with session with card with pin.
    withdrawal performTransaction.
    simulation ejectCard.

    |suite|
    suite := TestSuite named: 'withdrawal Tests'.
    suite addTestCase:(Test_withdrawal selector: #testreadCard).
    suite addTestCase:(Test_withdrawal selector: #testreadPIN).
    ^suite
```
B.4.2: SUnit test case for deposit operation

```java
import atm.*;
import atm.physical.*;
import atm.transaction.*;
import banking.*;
import simulation.*;
import junit.framework.*;
import java.awt.*;
import java.awt.event.*;
import java.net.InetAddress;

Class: Test_deposit
    superclass: TestCase
    id := int new
    id := 41.
    place := String new
    place := "Gorden College".
    name := String new
    name := "National Bank".
    address := InetAddress new.
    address := null.
    aTM := ATM new id with place with name with address.
    session := Session new aTM.
    simulation := Simulation new aTM.

    Test_deposit>>test_readCard
    expected_card:= expected_card new.
    expected_card_Int1:=int new.
    expected_card_Int1:=1.
    expected_card setNumber expected_card_Int1.
    card = (Card) simulation readCard.
    self assert:( card equals expected_card ).
            " Test Case # 1"

    customerConsole := CustomerConsole new.

    Test_deposit>>test_readPIN
    prompt:= String new.
    prompt:= "Please enter PIN.".
    expected_pin:= int new.
    expected_pin := 42.
    pin = (int) customerConsole readPIN prompt.
    self assert:( pin = expected_pin ).
            " Test Case # 2"

    Transaction makeTransaction aTM with session with card with pin.
    deposit := Deposit new aTM with session with card with pin.
    deposit performTransaction.
    simulation ejectCard.
```

```java
|suite|
suite := TestSuite named: 'deposit Tests'.
suite addTestCase:(Test_deposit selector: #testreadCard).
suite addTestCase:(Test_deposit selector: #testreadPIN).
^suite
```
B.4.3: SUnit test case for transfer operation

```java
import atm.*;
import atm.physical.*;
import atm.transaction.*;
import banking.*;
import simulation.*;
import junit.framework.*;
import java.awt.*;
import java.awt.event.*;
import java.net.InetAddress;

Class: Test_transfer
  superclass: TestCase
  id := int new.
  id := 41.
  place := String new.
  place := "Gorden College".
  name := String new.
  name := "National Bank".
  address := InetAddress new.
  address := null.
  aTM := ATM new id with place with name with address.
  session := Session new aTM.
  simulation := Simulation new aTM.

Test_transfer>>test_readCard
  expected_card:= expected_card new.
  expected_card_Int1:=int new.
  expected_card_Int1:=1.
  expected_card setNumber expected_card_Int1.
  card = (Card) simulation readCard.
  self assert:(card equals expected_card).   "Test Case # 1"

customerConsole := CustomerConsole new.

Test_transfer>>test_readPIN
  prompt:= String new.
  prompt:= "Please enter PIN."
  expected_pin:= int new.
  expected_pin := 42.
  pin = (int) customerConsole readPIN prompt.
  self assert:(pin = expected_pin).   "Test Case # 2"

Transaction makeTransaction aTM with session with card with pin.
transfer := Transfer new aTM with session with card with pin.
transfer performTransaction.
simulation ejectCard.

|suite|
suite := TestSuite named: 'transfer Tests'.
suite addTestCase:(Test_transfer selector: #testreadCard).
suite addTestCase:(Test_transfer selector: #testreadPIN).
^suite
```

Model-Driven Framework for Context Dependent Testing of Components
B.4.4: SUnit test case for balance inquiry operation

```java
import atm.*;
import atm.physical.*;
import atm.transaction.*;
import banking.*;
import simulation.*;
import junit.framework.*;
import java.awt.*;
import java.awt.event.*;
import java.net.InetAddress;

Class: Test_inquiry
                superclass: TestCase
                id := int new.
                id := 41.
                place := String new.
                place := "Gorden College".
                name := String new.
                name := "National Bank".
                address := InetAddress new.
                address := null.
                aTM := ATM new id with place with name with address.
                session := Session new aTM.
                simulation := Simulation new aTM.

Test_inquiry>>test_readCard
                expected_card:= expected_card new.
                expected_card_Int1:=int new.
                expected_card_Int1:=1.
                expected_card setNumber expected_card_Int1.
                card = (Card) simulation readCard.
                self assert:( card equals expected_card ). " Test Case # 1"

customerConsole := CustomerConsole new.

Test_inquiry>>test_readPIN
                prompt:= String new.
                prompt:= "Please enter PIN."
                expected_pin:= int new.
                expected_pin := 42.
                pin = (int) customerConsole readPIN prompt.
                self assert:( pin = expected_pin ). " Test Case # 2"

Transaction makeTransaction aTM with session with card with pin.
inquiry := Inquiry new aTM with session with card with pin.
inquiry performTransaction.
simulation ejectCard.

|suite|
suite := TestSuite named: 'inquiry Tests'.
suite addTestCase:(Test_inquiry selector: #testreadCard).
suite addTestCase:(Test_inquiry selector: #testreadPIN).
^suite
```
APPENDIX C: COMPARISON OF TEST SUITES

C.1: Implementation of Dijkstra’s shortest-path algorithm

```java
class GraphException extends RuntimeException {
    public GraphException(String name) {
        super(name);
    }
}
class Edge {
    public Vertex dest;
    public double cost;
    public Edge(Vertex d, double c) {
        dest = d;
        cost = c;
    }
}
class Path implements Comparable {
    public Vertex dest;
    public double cost;
    public Path(Vertex d, double c) {
        dest = d;
        cost = c;
    }
    public int compareTo(Object rhs) {
        double otherCost = ((Path)rhs).cost;
        return cost < otherCost ? -1 : cost > otherCost ? 1 : 0;
    }
}
class Vertex {
    public String name; // Vertex name
    public List adj; // Adjacent vertices
    public double dist; // Cost
    public Vertex prev; // Previous vertex on shortest path
    public int scratch;
    public Vertex(String nm) { name = nm; adj = new LinkedList(); reset(); }
    public void reset() { dist = Graph.INFINITY; prev = null; pos = null; scratch = 0; }
}
```
public class Graph {
    public static final double INFINITY = Double.MAX_VALUE;
    private Map vertexMap = new HashMap();

    public void addEdge(String sourceName, String destName, double cost) {
        Vertex v = getVertex(sourceName);
        Vertex w = getVertex(destName);
        v.adj.add(new Edge(w, cost));
    }

    public void printPath(String destName) {
        Vertex w = (Vertex) vertexMap.get(destName);
        if (w == null)
            throw new NoSuchElementException("Destination vertex not found");
        else if (w.dist == INFINITY)
            System.out.println(destName + " is unreachable");
        else {
            System.out.print(("Cost is: " + w.dist + ") "));
            printPath(w);
            System.out.println();
        }
    }

    private Vertex getVertex(String vertexName) {
        Vertex v = (Vertex) vertexMap.get(vertexName);
        if (v == null)
            { v = new Vertex(vertexName);
                vertexMap.put(vertexName, v);
            }
        return v;
    }

    private void printPath(Vertex dest) {
        if (dest.prev != null)
        { printPath(dest.prev); System.out.print( " to ");
            System.out.println(dest.name);
        }
    }

    private void clearAll() {
        for (Iterator itr = vertexMap.values().iterator(); itr.hasNext(); )
            (Vertex)itr.next().reset();
    }

    public String getPath(String destName) {
        String path = "";
        Vertex w = (Vertex) vertexMap.get(destName);
        if (w == null) |
throw new NoSuchElementException("Destination vertex not found");
} else if (w.dist == INFINITY) {
    path = "-1";
} else {
    path = getPath(w);
}
return path;

private String getPath(Vertex dest) {
    String path = "";
    if (dest.prev == null) {
        path = dest.name;
    } else {
        path = getPath(dest.prev) + " " + dest.name;
    }
    return(path);
}

public void dijkstra(String startName) {
    PriorityQueue pq = new BinaryHeap();
    Vertex start = (Vertex) vertexMap.get(startName);
    if (start == null) throw new NoSuchElementException("Start vertex not found");
    clearAll();
    pq.insert(new Path(start, 0));
    start.dist = 0;
    int nodesSeen = 0;
    while (!pq.isEmpty() && nodesSeen < vertexMap.size()) {
        Path vrec = (Path) pq.deleteMin();
        Vertex v = vrec.dest;
        if (v.scratch != 0) continue;
        v.scratch = 1;
        nodesSeen++;
        for (Iterator itr = v.adj.iterator(); itr.hasNext(); )
        {
            Edge e = (Edge) itr.next();
            Vertex w = e.dest;
            double cvw = e.cost;
            if (cvw < 0)
                throw new GraphException("Graph has negative edges");
            if (w.dist > v.dist + cvw)
            {
                w.dist = v.dist + cvw;
                w.prev = v;
                pq.insert(new Path(w, w.dist));
            }
        }
    }
}
C.2: Test suites for the shortest-path case study

C.2.1: Test suite 1 for the shortest-path case study

Test Suite 1 consists of the text files (which contain description of graph) and Java files (which contain JUnit test cases).

<table>
<thead>
<tr>
<th>Text Files</th>
<th>graph1.txt</th>
<th>graph2.txt</th>
<th>graph21.txt</th>
<th>graph22.txt</th>
<th>graph3.txt</th>
<th>graph31.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>A B 2</td>
<td>A B 6</td>
<td>A B 6</td>
<td>A B 6</td>
<td>A B 5</td>
<td>A B 4</td>
</tr>
<tr>
<td></td>
<td>B C 1</td>
<td>B C 1</td>
<td>B C 1</td>
<td>B C 4</td>
<td>A C 6</td>
<td>A C 2</td>
</tr>
<tr>
<td></td>
<td>A C 2</td>
<td>A C 8</td>
<td>A C 8</td>
<td>CD 2</td>
<td>A D 5</td>
<td>A D 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C B 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Java Files</th>
<th>GraphTest1</th>
<th>GraphTest2</th>
<th>GraphTest21</th>
<th>GraphTest22</th>
<th>GraphTest3</th>
<th>GraphTest31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>The contents of these files are provided below.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

GraphTest1.java

```java
public class GraphTest1 extends TestCase {

    public void testDijkstra() {
        Graph g = new Graph();
        try {
            FileReader fin = new FileReader("d:\TestSuite_1\graph1.txt");
            BufferedReader graphFile = new BufferedReader( fin );
            String line;
            while( ( line = graphFile.readLine() ) != null ) {
                StringTokenizer st = new StringTokenizer( line );
                try {
                    if( st.countTokens() != 3 ) continue; // skip incorrect line
                    String source = st.nextToken();
                    String dest   = st.nextToken();
                    int cost      = Integer.parseInt( st.nextToken() );
                    g.addEdge( source, dest, cost );
                } catch( NumberFormatException e ) { }
                String startName;
                String destName;
                try {
                    startName = "A";
                    g.dijkstra( startName );
                    String expectedPath;
                    String actualPath;
                    destName = "B";
                    expectedPath = "AB";
                    actualPath = g.getPath(destName);
                    assertEquals(actualPath, expectedPath);
                } catch(Exception exp) { }
            }
        } catch( IOException e ) { System.err.println( e ); }
    }
```
GraphTest2.java

public class GraphTest2 extends TestCase {

    public void testDijkstra() {
        Graph g = new Graph();
        try {
            FileReader fin = new FileReader("d:\\TestSuite_1\\graph2.txt");
            BufferedReader graphFile = new BufferedReader(fin);
            String line;
            while ( ( line = graphFile.readLine() ) != null ) {
                StringTokenizer st = new StringTokenizer(line);
                try {
                    if ( st.countTokens() != 3 ) continue;
                    String source = st.nextToken();
                    String dest = st.nextToken();
                    int cost = Integer.parseInt(st.nextToken());
                    g.addEdge(source, dest, cost);
                } catch (NumberFormatException e) { }
            }
            catch (IOException e) {
                System.err.println(e);
            }
            String startName = null;
            String destName = null;
            try {
                startName = "A";
                g.dijkstra(startName);
                string expectedPath;
                string actualPath;
                destName = "B";
                expectedPath = "AB";
                actualPath = g.getPath(destName);
                assertEquals(actualPath, expectedPath);
                destName = "C";
                expectedPath = "ABC";
                actualPath = g.getPath(destName);
                assertEquals(actualPath, expectedPath);
            } catch (Exception exp) { }
        }
    }
}
GraphTest21.java

```java
public class GraphTest21 extends TestCase {

    public void testDijkstra() {
        Graph g = new Graph();
        try {
            FileReader fin = new FileReader("d:\TestSuite_1\graph21.txt");
            BufferedReader graphFile = new BufferedReader(fin);
            String line;
            while ((line = graphFile.readLine()) != null) {
                StringTokenizer st = new StringTokenizer(line);
                try {
                    if (st.countTokens() != 3) continue;
                    String source = st.nextToken();
                    String dest = st.nextToken();
                    int cost = Integer.parseInt(st.nextToken());
                    g.addEdge(source, dest, cost);
                } catch (NumberFormatException e) { }
            }
            catch (IOException e) { System.err.println(e); }
            String startName = null;
            String destName = null;
            try {
                startName = "A";
                g.dijkstra(startName);
                String expectedResult;
                String actualPath;
                destName = "B";
                expectedResult = "AB";
                actualPath = g.getPath(destName);
                assertEquals(actualPath, expectedResult);
                destName = "C";
                expectedResult = "AC";
                actualPath = g.getPath(destName);
                assertEquals(actualPath, expectedResult);
            } catch (Exception exp) { }
        }
    }
}
```
GraphTest22.java

```java
public class GraphTest22 extends TestCase {

    public void testDijkstra() {
        Graph g = new Graph();
        try {
            FileReader fin = new FileReader("d:\\TestSuite_1\\graph22.txt");
            BufferedReader graphFile = new BufferedReader(fin);
            String line;
            while ( ( line = graphFile.readLine() ) != null ) {
                StringTokenizer st = new StringTokenizer(line);
                try {
                    if ( st.countTokens() != 3 ) continue;
                    String source = st.nextToken();
                    String dest = st.nextToken();
                    int cost = Integer.parseInt(st.nextToken());
                    g.addEdge(source, dest, cost);
                } catch (NumberFormatException e) {
                }
            }
            String startName = null;
            String destName = null;
            try {
                startName = "A";
                g.dijkstra(startName);
                String expectedPath;
                String actualPath;
                destName = "B";
                expectedPath = "AB";
                actualPath = g.getPath(destName);
                assertEquals(actualPath, expectedPath);
                destName = "C";
                expectedPath = "ABC";
                actualPath = g.getPath(destName);
                assertEquals(actualPath, expectedPath);
            } catch (Exception exp) {
            }
        } catch (IOException e) {
            System.err.println(e);
        }
    }
}
```
GraphTest3.java

public class GraphTest3 extends TestCase {

    public void testDijkstra() {
        Graph g = new Graph();

        try {
            FileReader fin = new FileReader("d:\\TestSuite_1\\graph3.txt");
            BufferedReader graphFile = new BufferedReader(fin);

            String line;
            while (line = graphFile.readLine() != null) {
                StringTokenizer st = new StringTokenizer(line);
                try {
                    if (st.countTokens() != 3) continue;
                    String source = st.nextToken();
                    String dest = st.nextToken();
                    int cost = Integer.parseInt(st.nextToken());
                    g.addEdge(source, dest, cost);
                }
                catch (NumberFormatException e) {
                }
            }
            catch (IOException e) { System.err.println(e); }

            String startName = null;
            String destName = null;
            try {
                startName = "A";
                g.dijkstra(startName);

                String expectedPath;
                String actualPath;

                destName = "B";
                expectedPath = "AB";
                actualPath = g.getPath(destName);
                assertEquals(actualPath, expectedPath);

                destName = "C";
                expectedPath = "ABC";
                actualPath = g.getPath(destName);
                assertEquals(actualPath, expectedPath);

                destName = "D";
                expectedPath = "AD";
                actualPath = g.getPath(destName);
                assertEquals(actualPath, expectedPath);

            } catch (Exception exp) {
            }
        }
    }
}
GraphTest31.java

public class GraphTest31 extends TestCase {

    public void testDijkstra() {
        Graph g = new Graph();

        try {
            FileReader fin = new FileReader("d:\TestSuite_1\graph31.txt");
            BufferedReader graphFile = new BufferedReader(fin);

            String line;
            while ((line = graphFile.readLine()) != null) {
                StringTokenizer st = new StringTokenizer(line);
                try {
                    if (st.countTokens() != 3) continue;
                    String source = st.nextToken();
                    String dest = st.nextToken();
                    int cost = Integer.parseInt(st.nextToken());
                    g.addEdge(source, dest, cost);
                } catch (NumberFormatException e) { }
            }
        } catch (IOException e) { System.err.println(e); }

        String startName;
        String destName;
        try {
            startName = "A";
            g.dijkstra(startName);
            String expectedPath, actualPath;
            destName = "B";
            expectedPath = "ACB";
            actualPath = g.getPath(destName);
            assertEquals(actualPath, expectedPath);

            destName = "C";
            expectedPath = "AC";
            actualPath = g.getPath(destName);
            assertEquals(actualPath, expectedPath);

            destName = "D";
            expectedPath = "ACD";
            actualPath = g.getPath(destName);
            assertEquals(actualPath, expectedPath);
        } catch (Exception exp) { }
    }
}

Model-Driven Framework for Context Dependent Testing of Components
C.2.2: Test suite 2 for the shortest-path case study

Test Suite 2 consists of the text files (which contain description of graph and the expected shortest paths) and Java files (which contain JUnit test cases).

<table>
<thead>
<tr>
<th>Text Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name</td>
</tr>
<tr>
<td>Contents</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Java Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name</td>
</tr>
<tr>
<td>Contents</td>
</tr>
</tbody>
</table>

Graph_Test2.java

```java
public class Graph_Test2 extends TestCase {

    private Graph graph = new Graph();
    private Map expectedMap = new HashMap();
    /**
     * Sets up the test fixture. Called before every test case method.
     */
    protected void setUp() {
        String graphDataFile = "D:\\TestSuite_2\\graph2.txt";
        String expectedDataFile = "D:\\TestSuite_2\\expected2.txt";
        // Load Graph Data
        try {
            FileReader fin = new FileReader(graphDataFile);
            BufferedReader graphFile = new BufferedReader( fin );
            String line;
            while( ( line = graphFile.readLine() ) != null ) {
                StringTokenizer st = new StringTokenizer( line );
                try {
                    if( st.countTokens() != 3 ) continue;
                    String source  = st.nextToken( );
                    String dest    = st.nextToken( );
                    int cost    = Integer.parseInt( st.nextToken( ) );
                    graph.addEdge( source, dest, cost );
                } catch( NumberFormatException e ) {} 
            }
        } catch( IOException e ) { System.err.println( e ); } 
    }
```
// Load Expected Data
try
{
    FileReader fin = new FileReader( expectedDataFile );
    BufferedReader graphFile = new BufferedReader( fin );
    String line;
    while( ( line = graphFile.readLine( ) ) != null )
    {
        StringTokenizer st = new StringTokenizer( line );
        try
        {
            if( st.countTokens( ) != 3 ) continue;
            String source  = st.nextToken( );
            String dest    = st.nextToken( );
            String expectedPath = st.nextToken( );
            expectedMap.put( source + "~" + dest, expectedPath );
        }
        catch( NumberFormatException e )
        {
        }
    }
    catch( IOException e ) { System.err.println( e ); }
}

/**
 * Tears down the test fixture. Called after every test case method.
 */
protected void tearDown()
{
    graph = null;
    expectedMap = null;
}

/**
 * Tests check shortest path form A vertex
 */
public void testDijkstra() throws Exception {

    String startName = ""
    String destName = ""
    String expectedPath;
    String actualPath;
    String value;
    Set vertexSet = graph.vertexMap.keySet();
    Iterator outerIt = vertexSet.iterator();
    Iterator innerIt = null;

    while ( outerIt.hasNext() ) {
        startName = (String) outerIt.next();
        try
        {
            graph.dijkstra( startName );
        }
        catch( IOException e ) { System.err.println( e ); }

        innerIt = vertexSet.iterator();
        while ( innerIt.hasNext() ) {
            destName = (String) innerIt.next();
            value = (String) expectedMap.get( startName + "~" + destName);
            expectedPath = value;
            actualPath = graph.getPath( destName );
            assertEquals(actualPath, expectedPath);
        }
    }
}
Graph_Test3.java

public class Graph_Test3 extends TestCase {
    
    private Graph graph = new Graph();
    private Map expectedMap = new HashMap();

    /**
     * Sets up the test fixture. Called before every test case method.
     */
    protected void setUp() {
        String graphDataFile = "D:\TestSuite_2\graph3.txt";
        String expectedDataFile = "D:\TestSuite_2\expected3.txt";
        // Load Graph Data
        try {
            FileReader fin = new FileReader(graphDataFile);
            BufferedReader graphFile = new BufferedReader( fin );
            String line;
            while( ( line = graphFile.readLine( ) ) != null ) {
                StringTokenizer st = new StringTokenizer( line );
                try {
                    if( st.countTokens( ) != 3 ) continue;
                    String source = st.nextToken();
                    String dest = st.nextToken();
                    int cost = Integer.parseInt( st.nextToken( ) );
                    graph.addEdge( source, dest, cost );
                } catch( NumberFormatException e ) { }
            } catch( IOException e ) { System.err.println( e ); }
        } catch( IOException e ) { System.err.println( e ); }

        // Load Expected Data
        try {
            FileReader fin = new FileReader( expectedDataFile );
            BufferedReader graphFile = new BufferedReader( fin );
            String line;
            while( ( line = graphFile.readLine( ) ) != null ) {
                StringTokenizer st = new StringTokenizer( line );
                try {
                    if( st.countTokens( ) != 3 ) continue;
                    String source = st.nextToken();
                    String dest = st.nextToken();
                    String expectedPath = st.nextToken();
                    expectedMap.put(source + "~" + dest, expectedPath);
                } catch( NumberFormatException e ) { }
            } catch( IOException e ) { System.out.println( e ); }
        } catch( IOException e ) { System.out.println( e ); }
    }
}
/**
 * Tears down the test fixture. Called after every test case method.
 */
protected void tearDown() {
    graph = null;
    expectedMap = null;
}

/**
 * Tests check shortest path form A vertex
 */
public void testDijkstra() throws Exception {
    String startName = "";
    String destName = "";
    String expectedPath;
    String actualPath;
    String value;
    Set vertexSet = graph.vertexMap.keySet();
    Iterator outerIt = vertexSet.iterator();
    Iterator innerIt = null;
    while (outerIt.hasNext()) {
        startName = (String)outerIt.next();
        try {
            graph.dijkstra(startName);
        } catch (IOException e) { System.err.println( e ); }
        innerIt = vertexSet.iterator();
        while (innerIt.hasNext()) {
            destName = (String)innerIt.next();
            value = (String)expectedMap.get(startName + "~" + destName);
            expectedPath = value;
            actualPath = graph.getPath(destName);
            assertEquals(actualPath, expectedPath);
        }
    }
}
public class Graph_Test4 extends TestCase {

    private Graph graph = new Graph();
    private Map expectedMap = new HashMap();

    /**
     * Sets up the test fixture. Called before every test case method.
     */
    protected void setUp() {

        String graphDataFile = "D:\TestSuite_2\graph4.txt";
        String expectedDataFile = "D:\TestSuite_2\expected4.txt";

        // Load Graph Data
        try {
            FileReader fin = new FileReader(graphDataFile);
            BufferedReader graphFile = new BufferedReader(fin);
            String line;
            while ((line = graphFile.readLine()) != null) {
                StringTokenizer st = new StringTokenizer(line);
                try {
                    if (st.countTokens() != 3) continue;
                    String source = st.nextToken();
                    String dest = st.nextToken();
                    int cost = Integer.parseInt(st.nextToken());
                    graph.addEdge(source, dest, cost);
                } catch (NumberFormatException e) { }
            }
        } catch (IOException e) { System.err.println(e); } 

        // Load Expected Data
        try {
            FileReader fin = new FileReader(expectedDataFile);
            BufferedReader graphFile = new BufferedReader(fin);
            String line;
            while ((line = graphFile.readLine()) != null) {
                StringTokenizer st = new StringTokenizer(line);
                try {
                    if (st.countTokens() != 3) continue;
                    String source = st.nextToken();
                    String dest = st.nextToken();
                    String expectedPath = st.nextToken();
                    expectedMap.put(source + "~" + dest, expectedPath);
                } catch (NumberFormatException e) { }
            }
        } catch (IOException e) { System.err.println(e); } 
    }
}
/**
 * Tears down the test fixture. Called after every test case method.
 */
protected void tearDown() {
    graph = null;
    expectedMap = null;
}

/**
 * Tests check shortest path form A vertex
 */
public void testDijkstra() throws Exception {

    String startName = "";
    String destName = "";
    String expectedPath;
    String actualPath;
    String value;
    Set vertexSet = graph.vertexMap.keySet();

    Iterator outerIt = vertexSet.iterator();
    Iterator innerIt = null;
    while (outerIt.hasNext()) {
        startName = (String)outerIt.next();
        try {
            graph.dijkstra(startName);
        } catch (IOException e) { System.out.println(e); }
        innerIt = vertexSet.iterator();
        while (innerIt.hasNext()) {
            destName = (String)innerIt.next();
            value = (String)expectedMap.get(startName + "~" + destName);
            expectedPath = value;
            actualPath = graph.getPath(destName);
            assertEquals(actualPath, expectedPath);
        }
    }
}
C.2.3: Test suite 3 the shortest-path case study

Test Suite 3 consists of text files (which contain description of graph) and Java files (which contain JUnit test cases).

<table>
<thead>
<tr>
<th>File Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>graph1.txt</td>
<td>&lt;empty file&gt;</td>
</tr>
<tr>
<td>graph2.txt</td>
<td>A B 6</td>
</tr>
<tr>
<td>graph3a.txt</td>
<td>A B 2</td>
</tr>
<tr>
<td></td>
<td>B C 1</td>
</tr>
<tr>
<td>graph3b.txt</td>
<td>A B 6</td>
</tr>
<tr>
<td>graph3c.txt</td>
<td>A B 9</td>
</tr>
<tr>
<td></td>
<td>B C 1</td>
</tr>
<tr>
<td></td>
<td>A C 2</td>
</tr>
<tr>
<td>graph4a.txt</td>
<td>A B B</td>
</tr>
<tr>
<td></td>
<td>B C 2</td>
</tr>
<tr>
<td></td>
<td>C D 7</td>
</tr>
<tr>
<td>graph4b.txt</td>
<td>A B 1</td>
</tr>
<tr>
<td></td>
<td>B C 2</td>
</tr>
<tr>
<td></td>
<td>C D 7</td>
</tr>
<tr>
<td></td>
<td>D A 2</td>
</tr>
<tr>
<td>graph4c.txt</td>
<td>A B B</td>
</tr>
<tr>
<td></td>
<td>B C 2</td>
</tr>
<tr>
<td></td>
<td>C D 7</td>
</tr>
<tr>
<td></td>
<td>D A 2</td>
</tr>
<tr>
<td></td>
<td>A C 2</td>
</tr>
<tr>
<td>graph4d.txt</td>
<td>A B 2</td>
</tr>
<tr>
<td></td>
<td>B C 2</td>
</tr>
<tr>
<td></td>
<td>C D 7</td>
</tr>
<tr>
<td></td>
<td>D A 2</td>
</tr>
<tr>
<td></td>
<td>A C 2</td>
</tr>
<tr>
<td></td>
<td>B D 1</td>
</tr>
<tr>
<td></td>
<td>A D 8</td>
</tr>
<tr>
<td>graph4e.txt</td>
<td>A B 1</td>
</tr>
<tr>
<td></td>
<td>B C 2</td>
</tr>
<tr>
<td></td>
<td>C B 1</td>
</tr>
<tr>
<td></td>
<td>C D 7</td>
</tr>
<tr>
<td></td>
<td>D C 2</td>
</tr>
<tr>
<td></td>
<td>A D 8</td>
</tr>
<tr>
<td></td>
<td>D A 2</td>
</tr>
<tr>
<td></td>
<td>A C 2</td>
</tr>
<tr>
<td></td>
<td>C A 1</td>
</tr>
<tr>
<td></td>
<td>B D 2</td>
</tr>
<tr>
<td></td>
<td>D B 1</td>
</tr>
</tbody>
</table>

LoadGraph.java

GraphTest2V.java

GraphTest3V.java

GraphTest4V.java

The contents of these files are provided below.
LoadGraph.java

```java
public class LoadGraph {
    public static Graph loadData(String fName)
    {
        Graph g = new Graph();
        try
        {
            FileReader fin = new FileReader(fName);
            BufferedReader graphFile = new BufferedReader(fin);
            String line;
            while ((line = graphFile.readLine()) != null)
            {
                StringTokenizer st = new StringTokenizer(line);
                try
                {
                    if (st.countTokens() < 3)
                        continue;
                    String source = st.nextToken();
                    String dest = st.nextToken();
                    int cost = Integer.parseInt(st.nextToken());
                    g.addEdge(source, dest, cost);
                }
                catch (NumberFormatException e)
                {
                }
            }
            catch (IOException exp)
            {
                System.err.println(exp);
            }
        }
        return g;
    }
}
```

GraphTest2V.java

```java
public class GraphTest2V extends TestCase {
    public void testDijkstra()
    {
        Graph g = new Graph();
        String startName = null;
        String destName = null;
        try
        {
            g = new Graph();
            g = LoadGraph.loadData("D:\\TestSuite_3\\graph2.txt");
            startName = "A";
            g.dijkstra(startName);
            String expectedPath;
            String actualPath;
            destName = "B";
            expectedPath = "AB";
            actualPath = g.getPath(destName);
            assertEquals(actualPath, expectedPath);
        }
        catch (Exception exp)
        {
        }
    }
}
```
public class GraphTest3V extends TestCase {

    Graph g = new Graph();

    public void testDijkstra() {
        String startName, destName, expectedPath, actualPath;

        try {
            startName = "A";
            g = new Graph();
            g = LoadGraph.loadData("D:\TestSuite_3\graph3a.txt");
            g.dijkstra( startName );

            destName = "B";
            expectedPath = "AB";
            actualPath = g.getPath(destName);
            assertEquals(actualPath, expectedPath);

            destName = "C";
            expectedPath = "ABC";
            actualPath = g.getPath(destName);
            assertEquals(actualPath, expectedPath);

            g = new Graph();
            g = LoadGraph.loadData("D:\TestSuite_3\graph3b.txt");
            g.dijkstra( startName );

            destName = "B";
            expectedPath = "AB";
            actualPath = g.getPath(destName);
            assertEquals(actualPath, expectedPath);

            destName = "C";
            expectedPath = "AC";
            actualPath = g.getPath(destName);
            assertEquals(actualPath, expectedPath);

            g = new Graph();
            g = LoadGraph.loadData("D:\TestSuite_3\graph3c.txt");
            g.dijkstra( startName );

            destName = "B";
            expectedPath = "AB";
            actualPath = g.getPath(destName);
            assertEquals(actualPath, expectedPath);

            destName = "C";
            expectedPath = "AC";
            actualPath = g.getPath(destName);
            assertEquals(actualPath, expectedPath);

        } catch ( Exception exp) { }
    }
}
**GraphTest4V.java**

```java
public class GraphTest4V extendsTestCase {
    Graph g = new Graph();

    public void testDijkstra() {
        String startName, destName, expectedPath, actualPath;

        try {
            startName = "A";
            g = new Graph();
            g = LoadGraph.loadData("D:\TestSuite_3\graph4a.txt");
            g.dijkstra(startName);

            destName = "B";
            expectedPath = "AB";
            actualPath = g.getPath(destName);
            assertEquals(actualPath, expectedPath);

            destName = "C";
            expectedPath = "ABC";
            actualPath = g.getPath(destName);
            assertEquals(actualPath, expectedPath);

            destName = "D";
            expectedPath = "ABCD";
            actualPath = g.getPath(destName);
            assertEquals(actualPath, expectedPath);

            g = new Graph();
            g = LoadGraph.loadData("D:\TestSuite_3\graph4b.txt");
            g.dijkstra(startName);

            destName = "B";
            expectedPath = "AB";
            actualPath = g.getPath(destName);
            assertEquals(actualPath, expectedPath);

            destName = "C";
            expectedPath = "AC";
            actualPath = g.getPath(destName);
            assertEquals(actualPath, expectedPath);

            destName = "D";
            expectedPath = "ABCD";
            actualPath = g.getPath(destName);
            assertEquals(actualPath, expectedPath);

            g = new Graph();
            g = LoadGraph.loadData("D:\TestSuite_3\graph4c.txt");
            g.dijkstra(startName);

            destName = "B";
            expectedPath = "AB";
            actualPath = g.getPath(destName);
            assertEquals(actualPath, expectedPath);

            destName = "C";
            expectedPath = "AC";
            actualPath = g.getPath(destName);
            assertEquals(actualPath, expectedPath);

            destName = "D";
        }
    }
}
```
expectedPath = "ACD";
actualPath = g.getPath(destName);
assertEquals(actualPath, expectedPath);

g = new Graph();
g = LoadGraph.loadData("D:\TestSuite_3\graph4d.txt");

g.dijkstra( startName );
destName = "B";
expectedPath = "AB";
actualPath = g.getPath(destName);
assertEquals(actualPath, expectedPath);
destName = "C";
expectedPath = "AC";
actualPath = g.getPath(destName);
assertEquals(actualPath, expectedPath);
destName = "D";
expectedPath = "ABD";
actualPath = g.getPath(destName);
assertEquals(actualPath, expectedPath);

g = new Graph();
g = LoadGraph.loadData("D:\TestSuite_3\graph4e.txt");

g.dijkstra( startName );
destName = "B";
expectedPath = "AB";
actualPath = g.getPath(destName);
assertEquals(actualPath, expectedPath);
destName = "C";
expectedPath = "AC";
actualPath = g.getPath(destName);
assertEquals(actualPath, expectedPath);
destName = "D";
expectedPath = "ABD";
actualPath = g.getPath(destName);
assertEquals(actualPath, expectedPath);

} catch(Exception exp) {} }
C.3: Execution traces of test suites for the shortest-path case study

**Test Suite 1**

<table>
<thead>
<tr>
<th>Test Suite 1</th>
<th>Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>e(A,B,2) d(A) p(B) expected=AB</td>
</tr>
<tr>
<td>T2</td>
<td>e(A,B,6) e(B,C,1) d(A) p(B) expected=AB</td>
</tr>
<tr>
<td>T3</td>
<td>e(A,B,6) e(B,C,1) d(A) p(C) expected=ABC</td>
</tr>
<tr>
<td>T4</td>
<td>e(A,B,6) e(B,C,1) e(A,C,2) d(A) p(B) expected=AB</td>
</tr>
<tr>
<td>T5</td>
<td>e(A,B,6) e(B,C,1) e(A,C,2) d(A) p(C) expected=AC</td>
</tr>
<tr>
<td>T6</td>
<td>e(A,B,6) e(B,C,1) e(A,C,8) d(A) p(B) expected=AB</td>
</tr>
<tr>
<td>T7</td>
<td>e(A,B,6) e(B,C,1) e(A,C,8) d(A) p(C) expected=ABC</td>
</tr>
<tr>
<td>T8</td>
<td>e(A,B,6) e(B,C,2) e(A,D,5) d(A) p(C) expected=AB</td>
</tr>
<tr>
<td>T9</td>
<td>e(A,B,5) e(B,C,4) e(C,D,2) e(A,D,5) d(A) p(C) expected=ABC</td>
</tr>
<tr>
<td>T10</td>
<td>e(A,B,5) e(B,C,4) e(C,D,2) e(A,D,5) d(A) p(D) expected=AD</td>
</tr>
<tr>
<td>T11</td>
<td>e(A,B,5) e(B,C,2) e(B,D,1) e(C,B,1) e(C,D,2) e(A,D,7) e(A,C,3) d(A) p(B) expected=ACB</td>
</tr>
<tr>
<td>T12</td>
<td>e(A,B,5) e(B,C,2) e(B,D,1) e(C,D,2) e(A,D,7) e(A,C,3) d(A) p(C) expected=AC</td>
</tr>
<tr>
<td>T13</td>
<td>e(A,B,5) e(B,C,2) e(B,D,1) e(C,B,1) e(C,D,2) e(A,D,7) e(A,C,3) d(A) p(D) expected=ACD</td>
</tr>
</tbody>
</table>

**Test Suite 2**

<table>
<thead>
<tr>
<th>Test Suite 2</th>
<th>Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>e(A,B,2) e(B,A,6) d(A) p(A) expected=0</td>
</tr>
<tr>
<td>T2</td>
<td>e(A,B,2) e(B,A,6) d(A) p(B) expected=AB</td>
</tr>
<tr>
<td>T3</td>
<td>e(A,B,2) e(B,A,6) d(A) p(A) expected=BA</td>
</tr>
<tr>
<td>T4</td>
<td>e(A,B,2) e(B,A,6) d(B) p(B) expected=0</td>
</tr>
<tr>
<td>T5</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) d(A) p(A) expected=AA</td>
</tr>
<tr>
<td>T6</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) d(A) p(B) expected=AB</td>
</tr>
<tr>
<td>T7</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) d(A) p(C) expected=AC</td>
</tr>
<tr>
<td>T8</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) d(B) p(A) expected=0</td>
</tr>
<tr>
<td>T9</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) d(B) p(C) expected=BC</td>
</tr>
<tr>
<td>T10</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) d(B) p(D) expected=BC</td>
</tr>
<tr>
<td>T11</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) d(D) p(A) expected=1</td>
</tr>
<tr>
<td>T12</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) d(D) p(C) expected=1</td>
</tr>
<tr>
<td>T13</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) d(D) p(D) expected=0</td>
</tr>
<tr>
<td>T14</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) e(B,D,2) e(a,A,5) e(c,D,3) d(A) p(A) expected=0</td>
</tr>
<tr>
<td>T15</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) e(B,D,2) e(a,A,5) e(c,D,3) d(A) p(B) expected=AB</td>
</tr>
<tr>
<td>T16</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) e(B,D,2) e(a,A,5) e(c,D,3) d(A) p(C) expected=ABC</td>
</tr>
<tr>
<td>T17</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) e(B,D,2) e(a,A,5) e(c,D,3) d(B) p(A) expected=ABD</td>
</tr>
<tr>
<td>T18</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) e(B,D,2) e(a,A,5) e(c,D,3) d(B) p(C) expected=BC</td>
</tr>
<tr>
<td>T19</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) e(B,D,2) e(a,A,5) e(c,D,3) d(B) p(D) expected=0</td>
</tr>
<tr>
<td>T20</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) e(B,D,2) e(a,A,5) e(c,D,3) d(D) p(A) expected=1</td>
</tr>
<tr>
<td>T21</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) e(B,D,2) e(a,A,5) e(c,D,3) d(D) p(B) expected=1</td>
</tr>
<tr>
<td>T22</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) e(B,D,2) e(a,A,5) e(c,D,3) d(D) p(C) expected=1</td>
</tr>
<tr>
<td>T23</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) e(B,D,2) e(a,A,5) e(c,D,3) d(D) p(D) expected=0</td>
</tr>
<tr>
<td>T24</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) e(B,D,2) e(a,A,5) e(c,D,3) d(D) p(D) expected=0</td>
</tr>
<tr>
<td>T25</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) e(B,D,2) e(a,A,5) e(c,D,3) d(D) p(D) expected=0</td>
</tr>
<tr>
<td>T26</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) e(B,D,2) e(a,A,5) e(c,D,3) d(D) p(D) expected=0</td>
</tr>
<tr>
<td>T27</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) e(B,D,2) e(a,A,5) e(c,D,3) d(D) p(D) expected=0</td>
</tr>
<tr>
<td>T28</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) e(B,D,2) e(a,A,5) e(c,D,3) d(D) p(D) expected=0</td>
</tr>
<tr>
<td>T29</td>
<td>e(A,B,2) e(B,A,4) e(A,C,8) e(B,D,2) e(a,A,5) e(c,D,3) d(D) p(D) expected=0</td>
</tr>
</tbody>
</table>

**Test Suite 3**

<table>
<thead>
<tr>
<th>Test Suite 3</th>
<th>Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>e(A,B,2) d(A) p(B) expected=AB</td>
</tr>
<tr>
<td>T2</td>
<td>e(A,B,2) e(B,C,1) d(A) p(B) expected=AB</td>
</tr>
<tr>
<td>T3</td>
<td>e(A,B,2) e(B,C,1) d(A) p(C) expected=ABC</td>
</tr>
<tr>
<td>T4</td>
<td>e(A,B,2) e(B,C,1) d(A) p(C) expected=AC</td>
</tr>
<tr>
<td>T5</td>
<td>e(A,B,2) e(B,C,1) d(A) p(C) expected=AC</td>
</tr>
<tr>
<td>T6</td>
<td>e(A,B,2) e(B,C,1) e(C,B,2) e(A,C,2) d(A) p(B) expected=AB</td>
</tr>
<tr>
<td>T7</td>
<td>e(A,B,1) e(B,A,3) e(B,C,1) e(C,B,2) e(A,C,2) d(A) p(B) expected=AB</td>
</tr>
<tr>
<td>T8</td>
<td>e(A,B,8) e(B,C,2) e(C,D,7) d(A) p(B) expected=AB</td>
</tr>
<tr>
<td>T9</td>
<td>e(A,B,8) e(B,C,2) e(C,D,7) d(A) p(C) expected=ABC</td>
</tr>
<tr>
<td>T10</td>
<td>e(A,B,8) e(B,C,2) e(C,D,7) d(A) p(D) expected=ABCD</td>
</tr>
<tr>
<td>T11</td>
<td>e(A,B,8) e(B,C,2) e(C,D,7) e(D,A,2) d(A) p(B) expected=AB</td>
</tr>
<tr>
<td>T12</td>
<td>e(A,B,1) e(B,C,2) e(C,D,7) e(D,A,2) d(A) p(C) expected=AC</td>
</tr>
<tr>
<td>T13</td>
<td>e(A,B,1) e(B,C,2) e(C,D,7) e(D,A,2) d(A) p(C) expected=ACD</td>
</tr>
<tr>
<td>T14</td>
<td>e(A,B,8) e(B,C,2) e(C,D,7) e(D,A,2) e(A,C,2) d(A) p(B) expected=AB</td>
</tr>
<tr>
<td>T15</td>
<td>e(A,B,8) e(B,C,2) e(C,D,7) e(D,A,2) e(A,C,2) d(A) p(C) expected=AC</td>
</tr>
<tr>
<td>T16</td>
<td>e(A,B,8) e(B,C,2) e(C,D,7) e(D,A,2) e(A,C,2) d(A) p(D) expected=ACD</td>
</tr>
<tr>
<td>T17</td>
<td>e(A,B,8) e(B,C,2) e(C,D,7) e(D,A,2) e(A,C,2) d(A) p(D) expected=ACD</td>
</tr>
<tr>
<td>T18</td>
<td>e(A,B,8) e(B,C,2) e(C,D,7) e(D,A,2) e(A,C,2) d(A) p(D) expected=ACD</td>
</tr>
<tr>
<td>T19</td>
<td>e(A,B,8) e(B,C,2) e(C,D,7) e(D,A,2) e(A,C,2) d(A) p(D) expected=ACD</td>
</tr>
<tr>
<td>T20</td>
<td>e(A,B,8) e(B,C,2) e(C,D,7) e(D,A,2) e(A,C,2) e(A,C,2) d(A) p(D) expected=ACD</td>
</tr>
<tr>
<td>T21</td>
<td>e(A,B,8) e(B,C,2) e(C,D,7) e(D,A,2) e(A,C,2) e(A,C,2) d(A) p(D) expected=ACD</td>
</tr>
<tr>
<td>T22</td>
<td>e(A,B,8) e(B,C,2) e(C,D,7) e(D,A,2) e(A,C,2) e(A,C,2) d(A) p(D) expected=ACD</td>
</tr>
</tbody>
</table>

Note: The methods addEdge, dijkstra and getPath are abbreviated as e, d and p respectively.
### C.4: Equivalence classes for the shortest-path case study

<table>
<thead>
<tr>
<th>Graph Nodes (L1)</th>
<th>Shortest Path Edges (L2)</th>
<th>Scenario (L3)</th>
<th>Description</th>
<th>Equivalence Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
<td>NA</td>
<td>0.0.1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Same source and destination</td>
<td>1.0.1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>Same source and destination</td>
<td>2.0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>No path exists between the two nodes</td>
<td>2.0.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>The shortest path contains one edge</td>
<td>2.1.1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>Same source and destination</td>
<td>3.0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>No path exists between the two nodes</td>
<td>3.0.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>The shortest path is the only path between the two nodes</td>
<td>3.1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>There exists a path (not the shortest one) with more than one edge</td>
<td>3.1.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>There exists a path that has the minimum cost but that has more than one edge</td>
<td>3.1.3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>The shortest path is the only path between the two nodes</td>
<td>3.2.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>There exists a path (not the shortest) with fewer than two edges</td>
<td>3.2.2</td>
</tr>
<tr>
<td>≥ 4</td>
<td>0</td>
<td>1</td>
<td>Same source and destination</td>
<td>4.0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>No path exists between the two nodes</td>
<td>4.0.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Shortest path is the only path between the two nodes</td>
<td>4.1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>There exists a path (not the shortest one) with more than one edge</td>
<td>4.1.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>There exists a path that has the minimum cost but that has more than one edge</td>
<td>4.1.3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Shortest path is the only path between the two nodes</td>
<td>4.2.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>There exists a path (not the shortest one) with more than two edges</td>
<td>4.2.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>There exists a path (not the shortest one) with fewer than two edges</td>
<td>4.2.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>There exists a path with fewer than two edges, and a path with more than two edges but they are not the shortest ones</td>
<td>4.2.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>There exists a path that has the minimum cost but that has more than two edges</td>
<td>4.2.5</td>
</tr>
<tr>
<td>≥ 3</td>
<td>1</td>
<td>1</td>
<td>Shortest path is the only path between the two nodes</td>
<td>4.3.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>There exists a path (not the shortest one) with more than three edges</td>
<td>4.3.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>There exists a path (not the shortest one) with fewer than three edges</td>
<td>4.3.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>There exists a path with fewer than three edges, and a path with more than three edges but they are not the shortest ones</td>
<td>4.3.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>There exists a path that has the minimum cost but that has more edges than the shortest path</td>
<td>4.3.5</td>
</tr>
</tbody>
</table>
C.5: Implementation of Boyer Moore's pattern-matching algorithm

```java
public class BoyerMoore {

    public static final int ALPHABET_SIZE = Character.MAX_VALUE + 1;

    private String text;
    private String pattern;

    private int[] last;
    private int[] match;
    private int[] suffix;

    public BoyerMoore(String pattern, String text) {
        this.text = text;
        this.pattern = pattern;
        last = new int[ALPHABET_SIZE];
        match = new int[ pattern.length() ];
        suffix = new int[ pattern.length() ];
    }

    public int match() {
        computeLast();
        computeMatch();

        int i = pattern.length() - 1;
        int j = pattern.length() - 1;

        while (i < text.length()) {
            if (pattern.charAt(j) == text.charAt(i)) {
                if (j == 0) {
                    return i;
                }
                j--;
                i--;
            }
            else {
                i += pattern.length()-j-1 + Math.max(j-last[text.charAt(i)],match[j]);
                j = pattern.length() - 1;
            }
        }
        return -1;
    }

    private void computeLast() {
        for (int k = 0; k < last.length; k++) {
            last[k] = -1;
        }

        for (int j = pattern.length()-1; j >= 0; j--) {
            if (last[pattern.charAt(j)] < 0) {
                last[pattern.charAt(j)] = j;
            }
        }
    }
```

private void computeMatch() {
for (int j = 0; j < match.length; j++) {
    match[j] = match.length;
}
computeSuffix();
for (int i = 0; i < match.length - 1; i++) {
    int j = suffix[i + 1] - 1;
    if (suffix[i] > j) {
        match[j] = j - i;
    } else {
        match[j] = Math.min(j - i + match[i], match[j]);
    }
}
if (suffix[0] < pattern.length()) {
    for (int j = suffix[0] - 1; j >= 0; j--) {
        if (suffix[0] < match[j]) {
            match[j] = suffix[0];
        }
    }
    int j = suffix[0];
    for (int k = suffix[j]; k < pattern.length(); k = suffix[k]) {
        while (j < k) {
            if (match[j] > k) {
                match[j] = k;
            }
            j++;
        }
    }
}
private void computeSuffix() {
    suffix[suffix.length-1] = suffix.length;
    int j = suffix.length - 1;
    for (int i = suffix.length - 2; i >= 0; i--) {
        while (j < suffix.length - 1 && pattern.charAt(j) != pattern.charAt(i)) {
            j = suffix[j + 1] - 1;
        }
        if (pattern.charAt(j) == pattern.charAt(i)) {
            j--;
        }
    }
    suffix[i] = j + 1;
}
C.6: Test suites for the pattern-matching case study

C.6.1: Test suite 1 for the pattern-matching case study

```java
import junit.framework.TestCase;

public class TS_1 extends TestCase {
    public static void testSuite1() {
        BoyerMoore boyerMoore;
        int ARRAY_SIZE = 26;
        String [][] testData = new String [ARRAY_SIZE][3];

        testData[0][0] = ""; // Test Case: 1
        testData[0][1] = "";
        testData[0][2] = "-1";

        testData[1][0] = ""; // Test Case: 2
        testData[1][1] = "A";
        testData[1][2] = "-1";

        testData[2][0] = "A"; // Test Case: 3
        testData[2][1] = "";
        testData[2][2] = "-1";

        testData[3][0] = "A"; // Test Case: 4
        testData[3][1] = "a";
        testData[3][2] = "-1";

        testData[4][0] = "A"; // Test Case: 5
        testData[4][1] = "Aa";
        testData[4][2] = "-1";

        testData[5][0] = "Aa"; // Test Case: 6
        testData[5][1] = "";
        testData[5][2] = "-1";

        testData[6][0] = "Aa"; // Test Case: 7
        testData[6][1] = "a";
        testData[6][2] = "1";

        testData[7][0] = "Aa"; // Test Case: 8
        testData[7][1] = "b";
        testData[7][2] = "-1";

        testData[8][0] = "Aa"; // Test Case: 9
        testData[8][1] = "Aa";
        testData[8][2] = "0";

        testData[9][0] = "Aa"; // Test Case: 10
        testData[9][1] = "Ab";
        testData[9][2] = "-1";

        testData[10][0] = "A "; // Test Case: 11
        testData[10][1] = "A ";
        testData[10][2] = "0";

        testData[11][0] = " A"; // Test Case: 12
        testData[11][1] = "A ";
        testData[11][2] = "0";

        testData[12][0] = "Aaa"; // Test Case: 13
        testData[12][1] = "";
        testData[12][2] = "-1";

        testData[13][0] = "Aaa"; // Test Case: 14
        testData[13][1] = "a";
        testData[13][2] = "1";
    }
}
```
Model-Driven Framework for Context Dependent Testing of Components

```java
try {
    int returnValue = -1;
    for (int i=0; i<ARRAY_SIZE; i++) {
        boyerMoore = new BoyerMoore( testData[i][1], testData[i][0] );
        returnValue = boyerMoore.match();
    }
} catch (Exception exp) {
    System.out.println(exp.getMessage());
}
```

```java
testData[14][0] = "Aaa";  // Test Case: 15
testData[14][1] = "Aa";
testData[14][2] = "0";

testData[15][0] = "Aaa";  // Test Case: 16
testData[15][1] = "aaa";
testData[15][2] = "-1";

testData[16][0] = "A a";  // Test Case: 17
testData[16][1] = " ";
testData[16][2] = "1";

testData[17][0] = "Aa ";  // Test Case: 18
testData[17][1] = " ";
testData[17][2] = "2";

testData[18][0] = "abcdefg hj";  // Test Case: 19
testData[18][1] = "k";
testData[18][2] = "-1";

testData[19][0] = "abcdefg hj";  // Test Case: 20
testData[19][1] = "ab";
testData[19][2] = "0";

testData[20][0] = "abcdefg hj";  // Test Case: 21
testData[20][1] = "abcde";
testData[20][2] = "0";

testData[21][0] = "abcdefg hj";  // Test Case: 22
testData[21][1] = "fghij";
testData[21][2] = "5";

testData[22][0] = "abcdefg hj";  // Test Case: 23
testData[22][1] = "bcdefg hj";
testData[22][2] = "4";

testData[23][0] = "abcdefg hj";  // Test Case: 24
testData[23][1] = "abcdeff hj";
testData[23][2] = "0";

testData[24][0] = "abcdefg hj";  // Test Case: 25
testData[24][1] = "abcdefg h";
testData[24][2] = "-1";

testData[25][0] = "abcdefg hj";  // Test Case: 26
```
C.6.2: Test suite 2 for the pattern-matching case study

```java
import junit.framework.TestCase;

public class TS_2 extends TestCase {
    public static void testSuite2() {
        BoyerMoore bm;
        String str;
        int result;
        String [] subStrings = {
            "all", // Test Case 1
            "gold.", // Test Case 2
            "is not", // Test Case 3
            "is", // Test Case 4
            "God", // Test Case 5
            "is", // Test Case 6
            "All", // Test Case 7
            "is", // Test Case 8
            "All that glitters is not gold.", // Test Case 9
            "is", // Test Case 10
            "god", // Test Case 11
            "gld", // Test Case 12
            "that glitters is not gold.", // Test Case 13
            "", // Test Case 14
            "gold", // Test Case 15
            "", // Test Case 16
            " ", // Test Case 17
            "abc", // Test Case 18
            "", // Test Case 19
            " glitters" }; // Test Case 20

        int [] expectedValue = {-1, -1, 18, -1, -1, -1, -1, -1, -1, 25, -1, 4, -1, 25, -1, -1, -1, -1, 0};

        try {
            str = "All that glitters is not gold.";
            for (int i=0; i < 15; i++) {
                bm = new BoyerMoore(subStrings[i], str);
                result = bm.match();
                assertEquals(result, expectedValue[i]);
                if (result!=expectedValue[i]) {
                    throw new Exception();
                }
            }

            str = "";
            for (int i=15; i < 18; i++) {
                bm = new BoyerMoore(subStrings[i], str);
                result = bm.match();
                assertEquals(result, expectedValue[i]);
                if (result!=expectedValue[i]) {
                    throw new Exception();
                }
            }

            str = " glitters";
            for (int i=18; i <= 19; i++) {
                bm = new BoyerMoore(subStrings[i], str);
                result = bm.match();
                assertEquals(result, expectedValue[i]);
                if (result!=expectedValue[i]) {
                    throw new Exception();
                }
            }
        } catch(Exception exp) {
            System.out.print(exp.getMessage());
        }
    }
}
```
C.6.3: Test suite 3 for the pattern-matching case study

```java
import junit.framework.TestCase;

public class TS_3 extends TestCase{
    public static void testSuite3() {
        BoyerMoore bm;
        int result;
        final int TEST_CASES = 16;

        BMTestCase[] testCases = {
            new BMTestCase("", "", -1), // TC: 1
            new BMTestCase("The Lord of the Rings", "", -1), // TC: 2
            new BMTestCase("The Lord of the Rings", "xyz", 3), // TC: 3
            new BMTestCase("The Lord of the Rings", "The", 0), // TC: 4
            new BMTestCase("The Lord of the Rings", "The", 0), // TC: 5
            new BMTestCase("The Lord of the Rings", "the", 12), // TC: 6
            new BMTestCase("The Lord of the Rings", "Lord", 4), // TC: 7
            new BMTestCase("The Lord of the Rings", "Lord of", 4), // TC: 8
            new BMTestCase("The Lord of the Rings", "the rings", -1), // TC: 9
            new BMTestCase("The Lord of the Rings", "Lord ", 3), // TC: 10
            new BMTestCase("The Lord of the Rings", "of the Rings", 9), // TC: 11
            new BMTestCase("The Lord of the Rings", "The lord of The", -1), // TC: 12
            new BMTestCase("The Lord of the Rings", "TheLordoftheRings", -1), // TC: 13
            new BMTestCase("The Lord of the Rings", "The Lord of the Ring", 0), // TC: 14
            new BMTestCase("The Lord of the Rings", "The Lord of the Rings", 0), // TC: 15
            new BMTestCase("The Lord of the Rings", "The Lord of the Rings.", -1) // TC: 16
        };

        try {
            for (int i=0; i < TEST_CASES; i++) {
                bm = new BoyerMoore(testCases[i].getPattern(), testCases[i].getStr());
                result = bm.match();
                assertEquals(result, testCases[i].getExp());
            }
        } catch(Exception exp) { System.out.print(exp.getMessage()); }
    } // End of Method
}

class BMTestCase {
    private String str;
    private String pattern;
    private int exp;

    public BMTestCase(String str, String pattern, int exp){
        this.str = str;
        this.pattern = pattern;
        this.exp = exp;
    }

    public String getStr(){
        return this.str;
    }

    public String getPattern(){
        return this.pattern;
    }

    public int getExp(){
        return this.exp;
    }
}
```

Model-Driven Framework for Context Dependent Testing of Components 250
### C.7: Execution traces of test suites for the pattern-matching case study

#### Test Suite 1

1. BoyerMoore("","") match() expected=-1
2. BoyerMoore("A","") match() expected=-1
3. BoyerMoore("A","A") match() expected=-1
4. BoyerMoore("a","A") match() expected=-1
5. BoyerMoore("aa","A") match() expected=-1
6. BoyerMoore("","aa") match() expected=-1
7. BoyerMoore("a","aa") match() expected=1
8. BoyerMoore("b","aa") match() expected=-1
9. BoyerMoore("aa","aa") match() expected=0
10. BoyerMoore("Ab","aa") match() expected=-1
11. BoyerMoore("A","A") match() expected=0
12. BoyerMoore("A","A") match() expected=0
13. BoyerMoore("","Aa") match() expected=-1
14. BoyerMoore("a","Aa") match() expected=1
15. BoyerMoore("Aa","Aa") match() expected=0
16. BoyerMoore("aaa","Aa") match() expected=-1
17. BoyerMoore("","Aa") match() expected=1
18. BoyerMoore("","Aa") match() expected=2
19. BoyerMoore("k","abcdefghi") match() expected=-1
20. BoyerMoore("ab","abcdefghi") match() expected=0
21. BoyerMoore("abcde","abcdefghi") match() expected=0
22. BoyerMoore("fgbhi","abcdefghi") match() expected=5
23. BoyerMoore("bcdefghj","abcdefghi") match()
24. ch() expected=1
25. BoyerMoore("abcdefgij","abcdefghi") match() expected=0
26. BoyerMoore("abcdeghi","abcdefghi") match() expected=-1
27. BoyerMoore("abcdefgik","abcdefghi") match() expected=-1

#### Test Suite 2

1. BoyerMoore("all","All that glitters is not gold.") match() expected=-1
2. BoyerMoore("gold","All that glitters is not gold.") match() expected=-1
3. BoyerMoore("is not","All that glitters is not gold.") match() expected=18
4. BoyerMoore("is","All that glitters is not gold.") match() expected=17
5. BoyerMoore("God","All that glitters is not gold.") match() expected=-1
6. BoyerMoore("Is","All that glitters is not gold.") match() expected=-1
7. BoyerMoore("All","All that glitters is not gold.") match() expected=-1
8. BoyerMoore("si","All that glitters is not gold.") match() expected=-1
9. BoyerMoore("All that glitters is not Gold.","All that glitters is not gold.") match() expected=-1
10. BoyerMoore("is","All that glitters is not gold.") match() expected=-1
11. BoyerMoore("go!","All that glitters is not gold.") match() expected=25
12. BoyerMoore("gld","All that glitters is not gold.") match() expected=-1
13. BoyerMoore("that glitters is not gold.","All that glitters is not gold.") match() expected=4
14. BoyerMoore("God","All that glitters is not gold.") match() expected=-1
15. BoyerMoore("gld","All that glitters is not gold.") match() expected=25
16. BoyerMoore("",""") match() expected=-1
17. BoyerMoore("",""") match() expected=-1
18. BoyerMoore("abc",""") match() expected=-1
19. BoyerMoore("","glitters") match() expected=-1
20. BoyerMoore("glitters","glitters") match() expected=0

#### Test Suite 3

1. BoyerMoore("","") match() expected=-1
2. BoyerMoore("","The Lord of the Rings") match() expected=-1
3. BoyerMoore("","The Lord of the Rings") match() expected=3
4. BoyerMoore("xyz","The Lord of the Rings") match() expected=-1
5. BoyerMoore("The","The Lord of the Rings") match() expected=0
6. BoyerMoore("the","The Lord of the Rings") match() expected=12
7. BoyerMoore("Lord","The Lord of the Rings") match() expected=4
8. BoyerMoore("Lord of","The Lord of the Rings") match() expected=4
9. BoyerMoore("the rings","The Lord of the Rings") match() expected=-1
10. BoyerMoore("Lord","The Lord of the Rings") match() expected=3
11. BoyerMoore("of the Rings","The Lord of the Rings") match() expected=9
12. BoyerMoore("The lord of","The Lord of the Rings") match() expected=-1
13. BoyerMoore("TheLordOfTheRings","The Lord of the Rings") match() expected=-1
14. BoyerMoore("TheLord of the Ring","The Lord of the Rings") match() expected=0
15. BoyerMoore("The Lord of the Rings","The Lord of the Rings") match() expected=0
16. BoyerMoore("The Lord of the Rings","The Lord of the Rings") match() expected=-1
C.8: Equivalence classes for the pattern-matching case study

<table>
<thead>
<tr>
<th>String Length (L1)</th>
<th>Pattern Length (L2)</th>
<th>Presence of Pattern (L3)</th>
<th>Equivalence Class (L1,L2,L3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1. The pattern does not occur in the string.</td>
<td>0.0.1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1. The pattern does not occur in the string.</td>
<td>0.1.1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1. The pattern does not occur in the string.</td>
<td>1.0.1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1. The pattern does not occur in the string.</td>
<td>1.1.1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1. The pattern does not occur in the string.</td>
<td>1.2.1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1. The pattern does not occur in the string.</td>
<td>2.0.1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1. The pattern occurs with different case.</td>
<td>2.1.2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1. The pattern occurs in the beginning of the string.</td>
<td>2.1.3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1. The pattern occurs at the end of the string.</td>
<td>2.1.4</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1. The pattern occurs in the middle of the string.</td>
<td>2.1.5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1. The pattern occurs more than once.</td>
<td>2.1.6</td>
</tr>
<tr>
<td>≥ 3</td>
<td>1</td>
<td>1. The pattern does not occur in the string.</td>
<td>3.0.1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1. The pattern occurs with different case.</td>
<td>3.1.2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1. The pattern occurs in the beginning of the string.</td>
<td>3.1.3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1. The pattern occurs at the end of the string.</td>
<td>3.1.4</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1. The pattern occurs in the middle of the string.</td>
<td>3.1.5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1. The pattern occurs more than once.</td>
<td>3.1.6</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1. The pattern does not occur in the string.</td>
<td>3.2.1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1. The pattern occurs with different case.</td>
<td>3.2.2</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1. The pattern occurs in the beginning of the string.</td>
<td>3.2.3</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1. The pattern occurs at the end of the string.</td>
<td>3.2.4</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1. The pattern occurs more than once.</td>
<td>3.2.5</td>
</tr>
<tr>
<td>≥ 4</td>
<td>1</td>
<td>1. The pattern does not occur in the string.</td>
<td>3.3.1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1. The pattern occurs with different case.</td>
<td>3.3.2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1. The pattern occurs in the beginning of the string.</td>
<td>3.3.3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1. The pattern occurs at the end of the string.</td>
<td>3.3.4</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1. The pattern occurs in the middle of the string.</td>
<td>3.3.5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1. The pattern occurs more than once.</td>
<td>3.3.6</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1. The pattern does not occur in the string.</td>
<td>4.0.1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1. The pattern does not occur in the string.</td>
<td>4.1.1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1. The pattern occurs with different case.</td>
<td>4.1.2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1. The pattern occurs in the beginning of the string.</td>
<td>4.1.3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1. The pattern occurs at the end of the string.</td>
<td>4.1.4</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1. The pattern occurs in the middle of the string.</td>
<td>4.1.5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1. The pattern occurs more than once.</td>
<td>4.1.6</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1. The pattern does not occur in the string.</td>
<td>4.2.1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1. The pattern occurs with different case.</td>
<td>4.2.2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1. The pattern occurs in the beginning of the string.</td>
<td>4.2.3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1. The pattern occurs at the end of the string.</td>
<td>4.2.4</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1. The pattern occurs in the middle of the string.</td>
<td>4.2.5</td>
</tr>
<tr>
<td></td>
<td>6. The pattern occurs more than once.</td>
<td>4.2.6</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1. The pattern does not occur in the string.</td>
<td>4.3.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. The pattern occurs with different case.</td>
<td>4.3.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. The pattern occurs in the beginning of the string.</td>
<td>4.3.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. The pattern occurs at the end of the string.</td>
<td>4.3.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5. The pattern occurs in the middle of the string.</td>
<td>4.3.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6. The pattern occurs more than once.</td>
<td>4.3.6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1. The pattern does not occur in the string.</td>
<td>4.4.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. The pattern occurs with different case.</td>
<td>4.4.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. The pattern occurs in the beginning of the string.</td>
<td>4.4.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. The pattern occurs at the end of the string.</td>
<td>4.4.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5. The pattern occurs in the middle of the string.</td>
<td>4.4.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6. The pattern occurs more than once.</td>
<td>4.4.6</td>
<td></td>
</tr>
</tbody>
</table>
C.9: Defects seeded in the pattern-matching case study

<table>
<thead>
<tr>
<th>Seed #</th>
<th>Line #</th>
<th>Actual Code</th>
<th>Mutant Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td><code>int i = pattern.length()-1;</code></td>
<td><code>int i = pattern.length();</code></td>
</tr>
<tr>
<td>2</td>
<td>26</td>
<td><code>int j = pattern.length()-1;</code></td>
<td><code>int j = pattern.length();</code></td>
</tr>
<tr>
<td>3</td>
<td>27</td>
<td><code>while (i &lt; text.length())</code></td>
<td><code>while (i &lt; text.length()-1)</code></td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td><code>return i;</code></td>
<td><code>return j;</code></td>
</tr>
<tr>
<td>5</td>
<td>46</td>
<td><code>int j = pattern.length()-1</code></td>
<td><code>int j = pattern.length()</code></td>
</tr>
<tr>
<td>6</td>
<td>54</td>
<td><code>j &lt; match.length</code></td>
<td><code>j &lt;= match.length</code></td>
</tr>
<tr>
<td>7</td>
<td>63</td>
<td><code>match[j] = j - i;</code></td>
<td><code>match[j] = i - j;</code></td>
</tr>
<tr>
<td>8</td>
<td>71</td>
<td><code>if (suffix[0] &lt; match[j])</code></td>
<td><code>if (suffix[i] &lt; match[j])</code></td>
</tr>
<tr>
<td>9</td>
<td>72</td>
<td><code>match[j] = suffix[0];</code></td>
<td><code>match[j] = suffix[i];</code></td>
</tr>
<tr>
<td>10</td>
<td>75</td>
<td><code>int j = suffix[0];</code></td>
<td><code>int i = suffix[0];</code></td>
</tr>
<tr>
<td>11</td>
<td>76</td>
<td><code>k &lt; pattern.length();</code></td>
<td><code>k &lt;= pattern.length();</code></td>
</tr>
<tr>
<td>12</td>
<td>77</td>
<td><code>while (j &lt; k)</code></td>
<td><code>while (j &lt;= k)</code></td>
</tr>
<tr>
<td>13</td>
<td>78</td>
<td><code>if (match[j] &gt; k)</code></td>
<td><code>if (match[j] &gt;= k)</code></td>
</tr>
<tr>
<td>14</td>
<td>88</td>
<td><code>int j = suffix.length - 1;</code></td>
<td><code>int j = suffix.length;</code></td>
</tr>
<tr>
<td>15</td>
<td>90</td>
<td><code>j &lt; suffix.length - 1</code></td>
<td><code>j &lt; suffix.length</code></td>
</tr>
<tr>
<td>16</td>
<td>91</td>
<td><code>j = suffix[j + 1] - 1;</code></td>
<td><code>j = suffix[j + 1];</code></td>
</tr>
</tbody>
</table>
# APPENDIX D: LUCENE CASE STUDY

## D.1: Unique traces of CDCT (CDCT\_UNIQUE\_TRACES)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>M1.SpanOr.Sor2.E2</td>
<td>102</td>
</tr>
<tr>
<td>103</td>
<td>M4.SpanOr.Sor2.E4</td>
<td>104</td>
</tr>
<tr>
<td>105</td>
<td>M6.SpanOr.Sor2.E6</td>
<td>106</td>
</tr>
<tr>
<td>107</td>
<td>M8.SpanOr.Sor2.E1</td>
<td>108</td>
</tr>
</tbody>
</table>
D.2: Unique traces of CT (CT\textsubscript{unique-traces})

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

---

_Model-Driven Framework for Context Dependent Testing of Components_ 256
### D.3: Pairs in CDCTUNIQUE.TRACES (CDCTPAIRS)

<table>
<thead>
<tr>
<th>Method : Execution Condition</th>
<th>Query Type : Execution Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $M_1 : E_1$</td>
<td>163. Term : $E_1$</td>
</tr>
<tr>
<td>2. $M_1 : E_2$</td>
<td>164. Term : $E_2$</td>
</tr>
<tr>
<td>3. $M_1 : E_3$</td>
<td>165. Term : $E_3$</td>
</tr>
<tr>
<td>4. $M_1 : E_4$</td>
<td>166. Term : $E_4$</td>
</tr>
<tr>
<td>5. $M_1 : E_5$</td>
<td>167. Term : $E_5$</td>
</tr>
<tr>
<td>6. $M_1 : E_6$</td>
<td>168. Term : $E_6$</td>
</tr>
<tr>
<td>7. $M_1 : E_7$</td>
<td>169. Range : $E_1$</td>
</tr>
<tr>
<td>8. $M_1 : E_8$</td>
<td>170. Range : $E_1$</td>
</tr>
<tr>
<td>9. $M_1 : E_9$</td>
<td>171. Range : $E_1$</td>
</tr>
<tr>
<td>10. $M_1 : E_{10}$</td>
<td>172. Range : $E_2$</td>
</tr>
<tr>
<td>11. $M_1 : E_{11}$</td>
<td>173. Range : $E_2$</td>
</tr>
<tr>
<td>12. $M_1 : E_{12}$</td>
<td>174. Range : $E_2$</td>
</tr>
<tr>
<td>13. $M_1 : E_{13}$</td>
<td>175. Boolean : $E_1$</td>
</tr>
<tr>
<td>14. $M_1 : E_{14}$</td>
<td>176. Boolean : $E_2$</td>
</tr>
<tr>
<td>15. $M_1 : E_{15}$</td>
<td>177. Boolean : $E_3$</td>
</tr>
<tr>
<td>16. $M_1 : E_{16}$</td>
<td>178. Boolean : $E_4$</td>
</tr>
<tr>
<td>17. $M_1 : E_{17}$</td>
<td>179. Boolean : $E_5$</td>
</tr>
<tr>
<td>18. $M_1 : E_{18}$</td>
<td>180. Boolean : $E_6$</td>
</tr>
<tr>
<td>19. $M_1 : E_{19}$</td>
<td>181. Prefix : $E_1$</td>
</tr>
<tr>
<td>20. $M_1 : E_{20}$</td>
<td>182. Prefix : $E_1$</td>
</tr>
<tr>
<td>21. $M_1 : E_{21}$</td>
<td>183. Prefix : $E_1$</td>
</tr>
<tr>
<td>22. $M_1 : E_{22}$</td>
<td>184. Prefix : $E_1$</td>
</tr>
<tr>
<td>23. $M_1 : E_{23}$</td>
<td>185. Prefix : $E_1$</td>
</tr>
<tr>
<td>24. $M_1 : E_{24}$</td>
<td>186. Prefix : $E_1$</td>
</tr>
<tr>
<td>25. $M_1 : E_{25}$</td>
<td>187. Wildcard : $E_1$</td>
</tr>
<tr>
<td>26. $M_1 : E_{26}$</td>
<td>188. Wildcard : $E_2$</td>
</tr>
<tr>
<td>27. $M_1 : E_{27}$</td>
<td>189. Wildcard : $E_3$</td>
</tr>
<tr>
<td>28. $M_1 : E_{28}$</td>
<td>190. Wildcard : $E_4$</td>
</tr>
<tr>
<td>29. $M_1 : E_{29}$</td>
<td>191. Wildcard : $E_5$</td>
</tr>
<tr>
<td>30. $M_1 : E_{30}$</td>
<td>192. Wildcard : $E_6$</td>
</tr>
<tr>
<td>31. $M_2 : E_1$</td>
<td>193. Fuzzy : $E_1$</td>
</tr>
<tr>
<td>32. $M_2 : E_2$</td>
<td>194. Fuzzy : $E_2$</td>
</tr>
<tr>
<td>33. $M_2 : E_3$</td>
<td>195. Fuzzy : $E_2$</td>
</tr>
<tr>
<td>34. $M_2 : E_4$</td>
<td>196. Fuzzy : $E_2$</td>
</tr>
<tr>
<td>35. $M_2 : E_5$</td>
<td>197. Fuzzy : $E_2$</td>
</tr>
<tr>
<td>36. $M_2 : E_6$</td>
<td>198. Fuzzy : $E_2$</td>
</tr>
<tr>
<td>37. $M_2 : E_7$</td>
<td>199. Phrase : $E_1$</td>
</tr>
<tr>
<td>38. $M_2 : E_8$</td>
<td>200. Phrase : $E_1$</td>
</tr>
<tr>
<td>39. $M_2 : E_9$</td>
<td>201. Phrase : $E_1$</td>
</tr>
<tr>
<td>40. $M_2 : E_{10}$</td>
<td>202. Phrase : $E_1$</td>
</tr>
<tr>
<td>41. $M_2 : E_{11}$</td>
<td>203. Phrase : $E_1$</td>
</tr>
<tr>
<td>42. $M_2 : E_{12}$</td>
<td>204. Phrase : $E_1$</td>
</tr>
<tr>
<td>43. $M_2 : E_{13}$</td>
<td>205. MultiPhrase : $E_1$</td>
</tr>
<tr>
<td>44. $M_2 : E_{14}$</td>
<td>206. MultiPhrase : $E_2$</td>
</tr>
<tr>
<td>45. $M_2 : E_{15}$</td>
<td>207. MultiPhrase : $E_3$</td>
</tr>
<tr>
<td>46. $M_2 : E_{16}$</td>
<td>208. MultiPhrase : $E_4$</td>
</tr>
<tr>
<td>47. $M_2 : E_{17}$</td>
<td>209. MultiPhrase : $E_5$</td>
</tr>
<tr>
<td>49. $M_3 : E_1$</td>
<td>211. SpanFirst : $E_1$</td>
</tr>
<tr>
<td>50. $M_3 : E_2$</td>
<td>212. SpanFirst : $E_2$</td>
</tr>
<tr>
<td>51. $M_3 : E_3$</td>
<td>213. SpanFirst : $E_3$</td>
</tr>
<tr>
<td>52. $M_3 : E_4$</td>
<td>214. SpanFirst : $E_4$</td>
</tr>
<tr>
<td>53. $M_3 : E_5$</td>
<td>215. SpanFirst : $E_5$</td>
</tr>
<tr>
<td>54. $M_3 : E_6$</td>
<td>216. SpanFirst : $E_6$</td>
</tr>
</tbody>
</table>
### D.4: Pairs in \textsc{CTunique_traces} (\textsc{Ctpairs})

<table>
<thead>
<tr>
<th>Method : Execution Condition</th>
<th>Query Type : Execution Condition</th>
<th>Query Type : Data Category</th>
<th>Query Type : Data Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. (M_1 : E_1)</td>
<td>66. Term : (E_1)</td>
<td>107. Term : (T_1)</td>
<td>124. MultiPhrase : (M_2)</td>
</tr>
<tr>
<td>2. (M_2 : E_1)</td>
<td>67. Term : (E_1)</td>
<td>108. Term : (T_1)</td>
<td>125. MultiPhrase : (M_3)</td>
</tr>
<tr>
<td>3. (M_3 : E_1)</td>
<td>68. Term : (E_1)</td>
<td>109. Range : (R_1)</td>
<td>126. SpanFirst : (S_1)</td>
</tr>
<tr>
<td>4. (M_4 : E_1)</td>
<td>69. Term : (E_1)</td>
<td>110. Range : (R_2)</td>
<td>127. SpanFirst : (S_2)</td>
</tr>
<tr>
<td>5. (M_5 : E_1)</td>
<td>70. Term : (E_1)</td>
<td>111. Boolean : (B_1)</td>
<td>128. SpanNear : (S_1)</td>
</tr>
<tr>
<td>6. (M_6 : E_1)</td>
<td>71. Term : (E_1)</td>
<td>112. Boolean : (B_2)</td>
<td>129. SpanNear : (S_2)</td>
</tr>
<tr>
<td>7. (M_7 : E_1)</td>
<td>72. Range : (E_1)</td>
<td>113. Boolean : (B_1)</td>
<td>130. SpanNot : (S_1)</td>
</tr>
<tr>
<td>8. (M_8 : E_1)</td>
<td>73. Range : (E_1)</td>
<td>114. Boolean : (B_2)</td>
<td>131. SpanNot : (S_2)</td>
</tr>
<tr>
<td>9. (M_9 : E_1)</td>
<td>74. Range : (E_1)</td>
<td>115. Prefix : (P_1)</td>
<td>132. SpanOr : (Sor)</td>
</tr>
<tr>
<td>10. (M_{10} : E_1)</td>
<td>75. Range : (E_1)</td>
<td>116. Prefix : (P_1)</td>
<td></td>
</tr>
<tr>
<td>11. (M_{11} : E_1)</td>
<td>76. Boolean : (E_1)</td>
<td>117. Wildcard : (W_1)</td>
<td></td>
</tr>
<tr>
<td>12. (M_{12} : E_1)</td>
<td>77. Boolean : (E_2)</td>
<td>118. Wildcard : (W_2)</td>
<td></td>
</tr>
<tr>
<td>13. (M_{13} : E_1)</td>
<td>78. Boolean : (E_1)</td>
<td>119. Wildcard : (W_3)</td>
<td></td>
</tr>
<tr>
<td>14. (M_{14} : E_1)</td>
<td>79. Boolean : (E_2)</td>
<td>120. Fuzzy : (F_2)</td>
<td></td>
</tr>
<tr>
<td>15. (M_{15} : E_1)</td>
<td>80. Boolean : (E_1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16. (M_{16} : E_1)</td>
<td>81. Boolean : (E_2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17. (M_{17} : E_1)</td>
<td>82. Prefix : (E_1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18. (M_{18} : E_1)</td>
<td>83. Prefix : (E_2)</td>
<td>121. Phrase : (Ph_1)</td>
<td></td>
</tr>
<tr>
<td>19. (M_{19} : E_1)</td>
<td>84. Wildcard : (E_1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20. (M_{20} : E_1)</td>
<td>85. Wildcard : (E_2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21. (M_{21} : E_1)</td>
<td>86. Wildcard : (E_3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22. (M_{22} : E_1)</td>
<td>87. Fuzzy : (E_1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23. (M_{23} : E_1)</td>
<td>88. Fuzzy : (E_2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24. (M_{24} : E_1)</td>
<td>89. Fuzzy : (E_3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25. (M_{25} : E_1)</td>
<td>90. Phrase : (E_1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26. (M_{26} : E_1)</td>
<td>91. Phrase : (E_2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27. (M_{27} : E_1)</td>
<td>92. Phrase : (E_3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28. (M_{28} : E_1)</td>
<td>93. Phrase : (E_4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29. (M_{29} : E_1)</td>
<td>94. Phrase : (E_5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30. (M_{30} : E_1)</td>
<td>95. MultiPhrase : (E_1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31. (M_{31} : E_1)</td>
<td>96. MultiPhrase : (E_2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32. (M_{32} : E_1)</td>
<td>97. MultiPhrase : (E_3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33. (M_{33} : E_1)</td>
<td>98. MultiPhrase : (E_4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34. (M_{34} : E_1)</td>
<td>99. SpanFirst : (E_1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35. (M_{35} : E_1)</td>
<td>100. SpanFirst : (E_2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36. (M_{36} : E_1)</td>
<td>101. SpanFirst : (E_3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37. (M_{37} : E_1)</td>
<td>102. SpanNear : (E_1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38. (M_{38} : E_1)</td>
<td>103. SpanNear : (E_2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39. (M_{39} : E_1)</td>
<td>104. SpanNear : (E_3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40. (M_{40} : E_1)</td>
<td>105. SpanOr : (E_1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41. (M_{41} : E_1)</td>
<td>106. SpanOr : (E_2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>42. (M_{42} : E_1)</td>
<td>107. SpanOr : (E_3)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
D.5: Abstract test suite for enriching CDCT ($\Delta$CDCT)

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Executing Condition</th>
<th>Method</th>
<th>Parameter</th>
<th>Query Type</th>
<th>Filter</th>
<th>Sort</th>
<th>int</th>
<th>Expected Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Empty Index - Index has zero documents ($E_1$)**

1. $E_1$ M1 BooleanQuery: +duck +parrot 0
2. $E_1$ M1 RangeQuery: [1980 TO 1981] 0
3. $E_1$ M1 WildcardQuery: mode* 0
4. $E_1$ M1 SpanFirstQuery: uses 0
5. $E_1$ M1 SpanNearQuery: model based 0
6. $E_1$ M1 SpanNotQuery: model testing 0
7. $E_1$ M1 SpanOrQuery: model based 0
8. $E_1$ M2 BooleanQuery: +duck +parrot $d_1$ 0
9. $E_1$ M2 FuzzyQuery: moden~ $d_1$ 0
10. $E_1$ M3 BooleanQuery: +duck +parrot $d_1$ author 0
11. $E_1$ M4 BooleanQuery: +duck +parrot author 0
12. $E_1$ M5 BooleanQuery: +duck +parrot 2 0
13. $E_1$ M5 PhraseQuery: testing uses 2 0
14. $E_1$ M5 PrefixQuery: /birds 2 0
15. $E_1$ M7 RangeQuery: [1980 TO 1981] $d_1$ author 2 0

**Index containing one document ($E_2$ and $E_3$)**

Document 1 ($d_1$) = { author: “beizer”,
body: “model based testing uses software models”,
year: “1980”,
pets: “/birds/anatidae/duck” }

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$E_2$</td>
<td>M1</td>
<td>FuzzyQuery: OCL~</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$E_2$</td>
<td>M2</td>
<td>PrefixQuery: /birds</td>
<td></td>
<td>$d_1$</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$E_2$</td>
<td>M2</td>
<td>WildcardQuery: mode*</td>
<td></td>
<td>$d_1$</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$E_2$</td>
<td>M2</td>
<td>SpanFirstQuery: uses</td>
<td></td>
<td>$d_1$</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$E_2$</td>
<td>M2</td>
<td>SpanNearQuery: UML OCL</td>
<td></td>
<td>$d_1$</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$E_2$</td>
<td>M2</td>
<td>SpanNotQuery: OCL testing</td>
<td></td>
<td>$d_1$</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$E_2$</td>
<td>M2</td>
<td>SpanOrQuery: UML OCL</td>
<td></td>
<td>$d_1$</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$E_2$</td>
<td>M3</td>
<td>FuzzyQuery: OCL~</td>
<td></td>
<td>$d_1$</td>
<td>author</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$E_2$</td>
<td>M4</td>
<td>FuzzyQuery: OCL~</td>
<td></td>
<td></td>
<td>author</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$E_2$</td>
<td>M5</td>
<td>FuzzyQuery: moden~</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$E_2$</td>
<td>M7</td>
<td>MultiphaseQuery: UML OCL</td>
<td>$d_1$</td>
<td>author</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$E_3$</td>
<td>M1</td>
<td>MultiphaseQuery: UML OCL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$E_3$</td>
<td>M3</td>
<td>MultiphaseQuery: UML OCL</td>
<td></td>
<td>$d_1$</td>
<td>author</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$E_3$</td>
<td>M3</td>
<td>WildcardQuery: mode*</td>
<td></td>
<td>$d_1$</td>
<td>author</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$E_3$</td>
<td>M3</td>
<td>SpanFirstQuery: uses</td>
<td></td>
<td>$d_1$</td>
<td>author</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$E_3$</td>
<td>M3</td>
<td>SpanNearQuery: model based</td>
<td>$d_1$</td>
<td>author</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$E_3$</td>
<td>M3</td>
<td>SpanNotQuery: testing based</td>
<td>$d_1$</td>
<td>author</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$E_3$</td>
<td>M3</td>
<td>SpanOrQuery: driven based</td>
<td>$d_1$</td>
<td>author</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$E_3$</td>
<td>M4</td>
<td>MultiphaseQuery: model based</td>
<td></td>
<td></td>
<td>author</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$E_3$</td>
<td>M5</td>
<td>MultiphaseQuery: model based</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$E_3$</td>
<td>M7</td>
<td>FuzzyQuery: moden~</td>
<td></td>
<td>$d_1$</td>
<td>author</td>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$E_3$</td>
<td>M7</td>
<td>PrefixQuery: /birds</td>
<td>$d_1$</td>
<td>author</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$E_3$</td>
<td>M8</td>
<td>FuzzyQuery: moden~</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Model-Driven Framework for Context Dependent Testing of Components
Index containing more than one document (E₄, E₅ and E₆)

document 1 (d₁) = { author: “beizer”,
body: “model based testing uses software models”,
year: “1980”,
pets: “/birds/anatidae/duck” }

document 2 (d₂) = { author: “utting”,
body: “model based testing uses software models”,
year: “1981”,
pets: “/birds/anatidae/swan” }

document 3 (d₃) = { author: “poston”,
body: “model based testing uses software models”,
year: “1982”,
pets: “/birds/anatidae/seagull” }

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>E₄</th>
<th>M₁</th>
<th>PhraseQuery: model OCL</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>E₄</td>
<td>M₁</td>
<td>PhraseQuery: model OCL</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>40</td>
<td>E₄</td>
<td>M₃</td>
<td>PhraseQuery: model OCL</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>41</td>
<td>E₄</td>
<td>M₄</td>
<td>PhraseQuery: model OCL</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>42</td>
<td>E₄</td>
<td>M₄</td>
<td>PhraseQuery: model OCL</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>43</td>
<td>E₄</td>
<td>M₄</td>
<td>SpanNotQuery: OCL testing</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>44</td>
<td>E₄</td>
<td>M₈</td>
<td>PrefixQuery:/animals</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>45</td>
<td>E₅</td>
<td>M₁</td>
<td>PrefixQuery:/duck</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>46</td>
<td>E₅</td>
<td>M₃</td>
<td>PrefixQuery:/duck</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>47</td>
<td>E₅</td>
<td>M₄</td>
<td>PrefixQuery:/duck</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>48</td>
<td>E₅</td>
<td>M₅</td>
<td>SpanNotQuery: OCL testing</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>49</td>
<td>E₅</td>
<td>M₅</td>
<td>SpanOrQuery: OCL driven</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>50</td>
<td>E₆</td>
<td>M₂</td>
<td>MultiphraseQuery: model testing</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>51</td>
<td>E₆</td>
<td>M₆</td>
<td>PhraseQuery: testing uses</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>52</td>
<td>E₆</td>
<td>M₂</td>
<td>RangeQuery: [1980 TO 1981]</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>53</td>
<td>E₆</td>
<td>M₃</td>
<td>RangeQuery: [1980 TO 1981]</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>54</td>
<td>E₆</td>
<td>M₃</td>
<td>TermQuery: model</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>55</td>
<td>E₆</td>
<td>M₄</td>
<td>RangeQuery: [1980 TO 1981]</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>56</td>
<td>E₆</td>
<td>M₄</td>
<td>SpanFirstQuery: model</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>57</td>
<td>E₆</td>
<td>M₄</td>
<td>SpanOrQuery: driven based</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>58</td>
<td>E₆</td>
<td>M₄</td>
<td>TermQuery: model</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>59</td>
<td>E₆</td>
<td>M₄</td>
<td>WildcardQuery: mode*</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>60</td>
<td>E₆</td>
<td>M₅</td>
<td>RangeQuery: [1980 TO 1981]</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>61</td>
<td>E₆</td>
<td>M₅</td>
<td>SpanFirstQuery: model based</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>62</td>
<td>E₆</td>
<td>M₅</td>
<td>WildcardQuery: mode*</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>63</td>
<td>E₆</td>
<td>M₇</td>
<td>PhraseQuery: testing uses</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>64</td>
<td>E₆</td>
<td>M₇</td>
<td>SpanFirstQuery: uses</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>65</td>
<td>E₆</td>
<td>M₇</td>
<td>SpanNearQuery: driven based</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>66</td>
<td>E₆</td>
<td>M₇</td>
<td>SpanNotQuery: testing driven</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>67</td>
<td>E₆</td>
<td>M₇</td>
<td>SpanOrQuery: driven based</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>68</td>
<td>E₆</td>
<td>M₇</td>
<td>WildcardQuery: mode*</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>69</td>
<td>E₆</td>
<td>M₈</td>
<td>RangeQuery: [1980 TO 1981]</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>70</td>
<td>E₆</td>
<td>M₈</td>
<td>WildcardQuery: mode*</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>71</td>
<td>E₆</td>
<td>M₉</td>
<td>FuzzyQuery: moden~</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>72</td>
<td>E₆</td>
<td>M₉</td>
<td>PrefixQuery:/birds</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>73</td>
<td>E₆</td>
<td>M₉</td>
<td>RangeQuery: [1980 TO 1981]</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
<tr>
<td>74</td>
<td>E₆</td>
<td>M₉</td>
<td>WildcardQuery: mode*</td>
<td>d₁, d₂</td>
<td>author: 0</td>
</tr>
</tbody>
</table>
In this table, \{ \} represents a document, and x:y represents a field (a name-value pair) in the document. The column “Executing Condition” specifies the executing conditions of the search method, e.g. search method which is invoked, e.g. \( E_6 \). The “Method” column specifies the overloaded search method which is invoked, e.g. \( M_2 \). The column “Query Type:Query Data” specifies the type of the query being executed along with the search text. The column “Filters” specifies the scope of the search, e.g. \( d1 \) shows that the search is limited to the document 1 in index. The column “Sort” shows the field on which the result set is sorted. The column “Expected Matches” shows the number of documents which are expected to match this query.
D.6: Sequence diagrams for the use cases of Searcher in ΔCDCT

UC-1: Execute Term Query using m1
UC-2: Execute Range Query using m2
UC-3: Execute Prefix Query using m3
UC-4: Execute Boolean using m4
UC-5: Execute Wildcard Query using m5
UC-6: Execute Fuzzy Query using m6

Let's execute the Fuzzy Query using the m6 Lucene Client Analyzer.

```
<create>
  analyzer: SimpleAnalyzer
</create>

<create>
  directory: RAMDirectory
</create>

<create>
  (directory, analyzer, true)
</create>

writer: IndexWriter
```

Loop

```
<create>
  doc: Document
</create>

<create>
  (name, value, Field.Store.NO, Field.Index.ANALYSED)
</create>

field: Field
```

```
addField()
```

```
addDocument(doc)
```

```
close()
```

```
<create>
  (field, analyzer)
</create>

parser: QueryParser
```

```
parse(queryText)
```

```
fuzzyQuery: FuzzyQuery
```

```
new Filter()
```

```
filter:
```

```
fuzzyQuery: FuzzyQuery
```

```
<create>
  (directory)
</create>

searcher: IndexSearcher
```

```
search(termQuery, filter, 2)
```

```
hits: ScoreDocs
```

```
close()
```

---

Model-Driven Framework for Context Dependent Testing of Components 267
UC-7: Execute Phrase Query using m7

Lucene Client >> Analyzer >> RAM Directory >> Document >> Field >> Index Writer >> Phrase Query >> Searcher >> Filter >> Sort

- create analyzer: SimpleAnalyzer
- create directory: RAM Directory
- create (directory, analyzer, true)
- writer: IndexWriter

loop
- create doc: Document
- create (name, value, Field: Store: NO, Field: Index: ANALYSED)
- add field
- addDocument(doc)
- close()

- create phraseQuery: PhraseQuery
  - add firstTermOfPhrase
  - add secondTermOfPhrase
  - filter: Filter
    - <<create>> (field)
    - sort: Sort

Filter
- add( secondTermOfPhrase)
- new Filter( )
  - filter: Filter
    - <<create>> (field)
    - sort: Sort

Sort
- <<create>> (field)
- searcher: IndexSearcher
  - search( termQuery, filter, 2)
  - hits: ScoreDocs
- close()
UC-8: Execute Multiphase Query using m8

Lucene Client

Analyzer <<Components>>

RAM Directory

Document

Field

Indexer <<Components>>

Multiphase Query

Searcher <<Component>>

TopDoc Collector

<<create>>

analyzer: SimpleAnalyzer

<<create>>

directory: RAMDirectory

<<create>> (directory, analyzer, true)

writer: IndexWriter

loop

<<create>>

doc: Document

<<create>> (name, value, Field.Store.NO, Field.Index.ANALYSED)

field

add field

addDocument(doc)

close()

<<create>>

multiphaseQuery: MultiphaseQuery

add(terms[])

new TopDocCollector(1000)

topDocCollector: TopDocCollector

<<create>> (directory)

searcher: IndexSearcher

search(termQuery, topDocCollector)

close()
UC-9: Execute SpanFirst Query using m9

Lucene Client

Analyzer <<Component>>

RAM Directory

Document

Field <<Component>>

SpanFirst Query

Searcher <<Component>>

Filter

TopDoc Collector

Term

<create>

analyzer: SimpleAnalyzer

<create>

directory: RAMDirectory

<create> [directory, analyzer, true]

writer: IndexWriter

loop

<create>

doc: Document

<create>

(name, value, Field.Store.NO, Field.Index.ANALYSED)

field: Field

add field

addDocument doc

close

<create>

(fieldName, termText)

term: Term

<create>

term: 3

spanFirst: SpanFirst

parse(filterQueryText)

range: RangeQuery

newQueryFilter rangeQuery

filterQueryFilter

new TopDocCollector (1000)

topDocCollector

<create>

directory

searcher: IndexSearcher

search(termQuery, filter, topDocCollector)

<create>

directory

searcher: IndexSearcher

search(termQuery, filter, topDocCollector)

<create>

directory

close()
UC-10: Execute SpanNear Query using m1

Lucene Client <<Component>> Analyzer <<Component>> RAM Directory Document Field Indexer <<Component>> SpanNearQuery Searcher <<Component>> TermQuery

Lucene Client

Analyzer: SimpleAnalyzer

<<create>>
directory: RAMDirectory

<<create>>
directory, analyzer, true

writer: IndexWriter

loop

<<create>>
doc: Document

<<create>>
(name, value: Field.Store.NO, Field.Index.ANALYSED)

field: Field

add field

addDocument(doc)

close()

<<create>>
termText

termQuery1: TermQuery

<<create>>
termQuery2: TermQuery

<<create>>
(SpanQuery[] [termQuery1, termQuery2], 4, true)

SpanNearQuery: SpanNearQuery

<<create>>
directory

searcher: IndexSearcher

search(SpanNearQuery)

hits: Hits

close()
UC-11: Execute SpanNot Query using m1

```
Lucene Client
Analyzer Component
RAM Directory
Document Field
Indexer Component
SpanNearQuery Searcher
TermQuery SpanQuery

<create>
analyser.SimpleAnalyzer
<create>
directory.RAMDirectory
<create> (directory, analyzer, true)
writer.IndexWriter
<create>
document
<create>
(name, value, Field.Store.NO, Field.Index.ANALYSED)
field.Field
<create>
document
add field
addDocument (doc)
close

<create>
termText
<create>
TermQuery1
<create>
TermQuery2
<create>
TermQuery3
<create> (SpanQuery1, TermQuery1, TermQuery2, 4, true)
<create>
SpanNearQuery
<create> (SpanNearQuery, term3)
spanNotQuery:SpanNotQuery
<create> (directory)
searcher.IndexSearcher
searchY (spanNotQuery)
Hits.Hits
close
```
UC-12: Execute SpanOr Query using m1
D.7: SMC Model for smc1

- Model SMC Model
  - Interaction smc1
    - Message SETUP_RAMDirectory
    - Message SETUP_SimpleAnalyzer
    - Message SETUP_IndexWriter
    - Message SETUP_IndexSearcher
    - Message SETUP_SORT
      - Owner Class SORT
        - Constructor Parameter sortField1
    - Message SETUP_TopNDocs
      - Owner Class INTEGER
        - Constructor Parameter intTopDocs1
    - Message SETUP_intSpanEnd1
      - Owner Class INTEGER
        - Constructor Parameter intSpanEnd1
    - Message SETUP_intSlop1
      - Owner Class INTEGER
        - Constructor Parameter intSlop1
    - Message SETUP_QueryParser
    - Message optimize
    - Message close
    - Message SETUP_BooleanText1
    - Message SETUP_RangeText1
    - Message SETUP_WildcardText1
    - Message SETUP_FuzzyText1
    - Message SETUP_PhraseText1
    - Message SETUP_TermText1
    - Message SETUP_TermText2
    - Message SETUP_TermText3
    - Message SETUP_PrefixText1
    - Message SETUP_Term1
    - Message SETUP_Term2
    - Message SETUP_Term3
    - Message SETUP_Term4
    - Message SETUP_Term5
    - Message SETUP_PrefixQuery1
    - Message parse
    - Message parse
    - Message parse
    - Message parse
    - Message parse
    - Message parse
    - Message parse
    - Message parse
    - Message parse
    - Message parse
  - Message SETUP_ResetTermQuery1
  - Message SETUP_ResetTermQuery2
  - Message SETUP_ResetTermQuery3
  - Message SETUP_ResetTermQuery4
  - Message SETUP_ResetFirstQuery1
  - Message SETUP_ResetQuery1
  - Message SETUP_ResetQuery2
  - Message SETUP_ResetNearQuery1
  - Message SETUP_ResetOrQuery1
  - Message search
  - Message getHitDocs
  - Message search
  - Message getHitDocs
  - Message search
  - Message getHitDocs
  - Message search
  - Message getHitDocs
  - Message search
  - Message getHitDocs
  - Message search
  - Message getHitDocs
  - Message search
  - Message getScoreDocs
  - Message search
  - Message getScoreDocs
  - Message search
  - Message getScoreDocs
  - Message search
  - Message getScoreDocs
D.8: Test data for smc1

<?xml version="1.0" encoding="UTF-8"?>
<SequenceDiagram name="smc1">

<Message name="IndexWriter::SETUP_IndexWriter"> <ConstructorParameter name="boolean1" value="true"/>
</Message>

<Message name="Sort::SETUP_Sort"> <ConstructorParameter name="sortField1" value="author"/>
</Message>

<Message name="Integer::SETUP_TopNDocs"> <ConstructorParameter name="intTopDocs1" value="2"/>
</Message>

<Message name="Integer::SETUP_intSpanEnd1"> <ConstructorParameter name="intSpanEnd1" value="3"/>
</Message>

<Message name="Integer::SETUP_intSlop1"> <ConstructorParameter name="intSlop1" value="1"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ConstructorParameter name="field1" value="body"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ConstructorParameter name="termText1" value="testing uses"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ConstructorParameter name="termText2" value="model"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ConstructorParameter name="termText3" value="based"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ConstructorParameter name="termText4" value="testing"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ConstructorParameter name="prefixText1" value="birds"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ExpectedValue name="expected_hitsVector1" value="new HitsVector()"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ExpectedValue name="expected_hitsVector2" value="expected_hitsVector1"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ExpectedValue name="expected_hitsVector3" value="expected_hitsVector1"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ExpectedValue name="expected_hitsVector4" value="expected_hitsVector1"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ExpectedValue name="expected_hitsVector5" value="expected_hitsVector1"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ExpectedValue name="expected_hitsVector6" value="expected_hitsVector1"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ExpectedValue name="expected_hitsVector7" value="expected_hitsVector1"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ExpectedValue name="expected_hitsVector8" value="expected_hitsVector1"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ExpectedValue name="expected_hitsVector9" value="expected_hitsVector1"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ExpectedValue name="expected_hitsVector10" value="expected_hitsVector1"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ExpectedValue name="expected_hitsVector11" value="expected_hitsVector1"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ExpectedValue name="expected_hitsVector12" value="expected_hitsVector1"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ExpectedValue name="expected_hitsVector13" value="expected_hitsVector1"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ExpectedValue name="expected_hitsVector14" value="expected_hitsVector1"/>
</Message>

<Message name="Array::SETUP_arrayParameter"> <ExpectedValue name="expected_hitsVector15" value="expected_hitsVector1"/>
</Message>

</SequenceDiagram>
D.9: xUnit Model for smc1

Model-Driven Framework for Context Dependent Testing of Components
D.10: Concrete test cases for ΔCDCT

Test Suite for SMC1

```java
package org.apache.lucene.search;

import java.util.BitSet;
import java.util.Vector;
import junit.framework.*;
import org.apache.lucene.analysis.*;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.index.*;
import org.apache.lucene.document.*;
import org.apache.lucene.document.Field.Index;
import org.apache.lucene.document.Field.Store;
import org.apache.lucene.document.Field.*;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.spans.*;
import org.apache.lucene.store.*;
import org.apache.lucene.util.DocIdBitSet;
import org.apache.lucene.search.HitDoc;
import java.util.*;

public class TestSuite_smc1 extends TestCase {

    public static void main(String args[]) {
        TestSuite testSuite = new TestSuite(TestSuite_smc1.class);
        testSuite.run(new TestResult());
    }

    public void test_smc1() {
        try {
            RAMDirectory rAMDirectory1 = new RAMDirectory();
            SimpleAnalyzer simpleAnalyzer1 = new SimpleAnalyzer();
            boolean boolean1 = true;
            IndexWriter indexWriter1 = new IndexWriter(rAMDirectory1, simpleAnalyzer1, boolean1);
            IndexSearcher indexSearcher1 = new IndexSearcher(rAMDirectory1);
            String sortField1 = "author";
            Sort sort1 = new Sort(sortField1);
            int intTopDocs1 = 2;
            Integer integer1 = new Integer(intTopDocs1);
            int intSpanEnd1 = 3;
            Integer integer2 = new Integer(intSpanEnd1);
            int intSlop1 = 1;
            Integer integer3 = new Integer(intSlop1);
            String field1 = "body";
            QueryParser queryParser1 = new QueryParser(field1, simpleAnalyzer1);
            indexWriter1.optimize();
            indexWriter1.close();
            String booleanText1 = "+duck +parrot";
            String string1 = new String(booleanText1);
            String rangeText1 = "[1980 TO 1981]";
            String string2 = new String(rangeText1);
            String wildcardText1 = "mode*";
            String string3 = new String(wildcardText1);
            String fuzzyText1 = "moden~";
            String string4 = new String(fuzzyText1);
            String phraseText1 = "testing uses";
            String string5 = new String(phraseText1);
            String termText1 = "uses";
            String string6 = new String(termText1);
            String termText2 = "model";
            String string7 = new String(termText2);
            String termText3 = "based";
            String string8 = new String(termText3);
            String termText4 = "testing";
            String string9 = new String(termText4);
            String prefixText1 = "birds";
            String string10 = new String(prefixText1);
            Term term1 = new Term(field1, termText1);
            Term term2 = new Term(field1, termText2);
        }
    }
}
```
Model-Driven Framework for Context Dependent Testing of Components

Term term3 = new Term(field1, termText3);
Term term4 = new Term(field1, termText4);
Term term5 = new Term(field1, prefixText1);
PrefixQuery prefixQuery1 = new PrefixQuery(term5);
Query query1 = (Query) queryParser1.parse(booleanText1);
Query query2 = (Query) queryParser1.parse(rangeText1);
Query query3 = (Query) queryParser1.parse(wildcardText1);
Query query4 = (Query) queryParser1.parse(fuzzyText1);
Query query5 = (Query) queryParser1.parse(phraseText1);
Query query6 = (Query) queryParser1.parse(termText1);
Query query7 = (Query) queryParser1.parse(termText2);
Query query8 = (Query) queryParser1.parse(termText3);
Query query9 = (Query) queryParser1.parse(termText4);
SpanTermQuery spanTermQuery1 = new SpanTermQuery(term1);
SpanTermQuery spanTermQuery2 = new SpanTermQuery(term2);
SpanTermQuery spanTermQuery3 = new SpanTermQuery(term3);
SpanTermQuery spanTermQuery4 = new SpanTermQuery(term4);
SpanFirstQuery spanFirstQuery1 = new SpanFirstQuery(spanTermQuery1, intSpanEnd1);
SpanQuery[] spanQuery1 = {spanTermQuery1, spanTermQuery2, spanTermQuery4};
SpanQuery[] spanQuery2 = {spanTermQuery2, spanTermQuery3};
SpanNearQuery spanNearQuery1 = new SpanNearQuery(spanQuery1, intSlop1, boolean1);
SpanNotQuery spanNotQuery1 = new SpanNotQuery(spanTermQuery1, spanTermQuery2, spanTermQuery4);
SpanOrQuery spanOrQuery1 = new SpanOrQuery(spanQuery2);
Hits hits1 = (Hits) indexSearcher1.search(query1);
HitsVector expected_hitsVector1 = new HitsVector();
HitsVector hitsVector1 = (HitsVector) hits1.getHitDocs();
assertTrue(hitsVector1.equals(expected_hitsVector1)); // Test Case # 1

Hits hits2 = (Hits) indexSearcher1.search(query2);
HitsVector expected_hitsVector2 = expected_hitsVector1;
HitsVector hitsVector2 = (HitsVector) hits2.getHitDocs();
assertTrue(hitsVector2.equals(expected_hitsVector2)); // Test Case # 2

Hits hits3 = (Hits) indexSearcher1.search(query3);
HitsVector expected_hitsVector3 = expected_hitsVector1;
HitsVector hitsVector3 = (HitsVector) hits3.getHitDocs();
assertTrue(hitsVector3.equals(expected_hitsVector3)); // Test Case # 3

Hits hits4 = (Hits) indexSearcher1.search(spanFirstQuery1);
HitsVector expected_hitsVector4 = expected_hitsVector1;
HitsVector hitsVector4 = (HitsVector) hits4.getHitDocs();
assertTrue(hitsVector4.equals(expected_hitsVector4)); // Test Case # 4

Hits hits5 = (Hits) indexSearcher1.search(spanNearQuery1);
HitsVector expected_hitsVector5 = expected_hitsVector1;
HitsVector hitsVector5 = (HitsVector) hits5.getHitDocs();
assertTrue(hitsVector5.equals(expected_hitsVector5)); // Test Case # 5

Hits hits6 = (Hits) indexSearcher1.search(spanNotQuery1);
HitsVector expected_hitsVector6 = expected_hitsVector1;
HitsVector hitsVector6 = (HitsVector) hits6.getHitDocs();
assertTrue(hitsVector6.equals(expected_hitsVector6)); // Test Case # 6

Hits hits7 = (Hits) indexSearcher1.search(spanOrQuery1);
HitsVector expected_hitsVector7 = expected_hitsVector1;
HitsVector hitsVector7 = (HitsVector) hits7.getHitDocs();
assertTrue(hitsVector7.equals(expected_hitsVector7)); // Test Case # 7

Hits hits8 = (Hits) indexSearcher1.search(query1, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSet bitset = new BitSet(1); bitset.set(0); bitset.set(1); return new DocIdBitSet(bitset); } });
HitsVector expected_hitsVector8 = expected_hitsVector1;
HitsVector hitsVector8 = (HitsVector) hits8.getHitDocs();
assertTrue(hitsVector8.equals(expected_hitsVector8)); // Test Case # 8

Hits hits9 = (Hits) indexSearcher1.search(query4, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSet bitset = new BitSet(1); bitset.set(0); bitset.set(1); return new DocIdBitSet(bitset); } });
HitsVector expected_hitsVector9 = expected_hitsVector1;
HitsVector hitsVector9 = (HitsVector) hits9.getHitDocs();
assertTrue(hitsVector9.equals(expected_hitsVector9)); // Test Case # 9
Hits hits10 = (Hits) indexSearcher1.search(query1, new Filter() {
    public DocIdSet getDocIdSet(IndexReader reader) {
        BitSet bitset = new BitSet(1); bitset.set(0); bitset.set(1); return new DocIdBitSet(bitset); }
    }, sort1);
HitsVector hitsVector10 = (HitsVector) hits10.getHitDocs();
assertTrue(hitsVector10.equals(expected_hitsVector10)); // Test Case # 10

Hits hits11 = (Hits) indexSearcher1.search(query1, sort1);
HitsVector hitsVector11 = (HitsVector) hits11.getHitDocs();
assertTrue(hitsVector11.equals(expected_hitsVector11)); // Test Case # 11

TopDocs topDocs1 = (TopDocs) indexSearcher1.search(query1, intTopDocs1);
HitsVector hitsVector12 = (HitsVector) topDocs1.getScoreDocs();
assertTrue(hitsVector12.equals(expected_hitsVector12)); // Test Case # 12

TopDocs topDocs2 = (TopDocs) indexSearcher1.search(query5, intTopDocs1);
HitsVector hitsVector13 = (HitsVector) topDocs2.getScoreDocs();
assertTrue(hitsVector13.equals(expected_hitsVector13)); // Test Case # 13

TopDocs topDocs3 = (TopDocs) indexSearcher1.search(prefixQuery1, intTopDocs1);
HitsVector hitsVector14 = (HitsVector) topDocs3.getScoreDocs();
assertTrue(hitsVector14.equals(expected_hitsVector14)); // Test Case # 14

TopDocs topDocs4 = (TopDocs) indexSearcher1.search(query2, new Filter() {
    public DocIdSet getDocIdSet(IndexReader reader) {
        BitSet bitset = new BitSet(1); bitset.set(0); bitset.set(1); return new DocIdBitSet(bitset); }
    }, intTopDocs1, sort1);
HitsVector hitsVector15 = (HitsVector) topDocs4.getScoreDocs();
assertTrue(hitsVector15.equals(expected_hitsVector15)); // Test Case # 15

} catch (Exception exp) {
    System.out.println(exp.toString());
    fail("Exception occurred during test case execution");
}

} //End of Method

//End of Class
Test Suite for SMC2
package org.apache.lucene.search;

import java.util.BitSet;
import java.util.Vector;
import junit.framework.*;
import org.apache.lucene.analysis.*;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.index.*;
import org.apache.lucene.document.*;
import org.apache.lucene.document.Field.Index;
import org.apache.lucene.document.Field.Store;
import org.apache.lucene.document.Field.*;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.spans.*;
import org.apache.lucene.store.*;
import org.apache.lucene.util.DocIdBitSet;
import java.util.*;

public class TestSuite_smc2 extends TestCase {

    public static void main(String args[]) {
        TestSuite testSuite = new TestSuite(TestSuite_smc2.class);
        testSuite.run(new TestResult());
    }

    public void test_smc2() {
        try {
            RAMDirectory rAMDirectory1 = new RAMDirectory();
            SimpleAnalyzer simpleAnalyzer1 = new SimpleAnalyzer();
            boolean boolean1 = true;
            IndexWriter indexWriter1 = new IndexWriter(rAMDirectory1, simpleAnalyzer1, boolean1);
            IndexSearcher indexSearcher1 = new IndexSearcher(rAMDirectory1);
            String sortField1 = "author";
            Sort sort1 = new Sort(sortField1);
            int intTopDocs1 = 2;
            Integer integer1 = new Integer(intTopDocs1);
            int intSpanEnd1 = 3;
            Integer integer2 = new Integer(intSpanEnd1);
            int intSlop1 = 1;
            Integer integer3 = new Integer(intSlop1);
            String name_A_1 = "body";
            String value_A_1 = "model based testing uses software models";
            Store store_A_1 = Field.Store.YES;
            Index index_A_1 = Field.Index.ANALYZED;
            Field field_A_1 = new Field(name_A_1, value_A_1, store_A_1, index_A_1);
            Document document1 = new Document();
            document1.add(field_A_1);
            QueryParser queryParser1 = new QueryParser(name_A_1, simpleAnalyzer1);
            indexWriter1.optimize();
            indexWriter1.close();
            String wildcardText1 = "mode*";
            String string1 = new String(wildcardText1);
            String fuzzyText1 = "OCL~";
            String string2 = new String(fuzzyText1);
            String termText1 = "uses";
            String string3 = new String(termText1);
        }
    }
}
String termText2 = "UML";
String string4 = new String( termText2 );
String termText3 = "OCL";
String string5 = new String( termText3 );
String termText4 = "testing";
String string6 = new String( termText4 );
String prefixText1 = "birds";
String string7 = new String( prefixText1 );
Term term1 = new Term( name_A_1, termText1 );
Term term2 = new Term( name_A_1, termText2 );
Term term3 = new Term( name_A_1, termText3 );
Term term4 = new Term( name_A_1, termText4 );
Term term5 = new Term( name_A_1, prefixText1 );
PrefixQuery prefixQuery1 = new PrefixQuery( term5 );
MultiPhraseQuery multiPhraseQuery1 = new MultiPhraseQuery( );
multiPhraseQuery1.add(term2);
multiPhraseQuery1.add(term3);
Query query1 = (Query) queryParser1.parse( wildcardText1 );
Query query2 = (Query) queryParser1.parse( fuzzyText1 );
Query query3 = (Query) queryParser1.parse( termText1 );
Query query4 = (Query) queryParser1.parse( termText2 );
Query query5 = (Query) queryParser1.parse( termText3 );
Query query6 = (Query) queryParser1.parse( termText4 );
SpanTermQuery spanTermQuery1 = new SpanTermQuery( term1 );
SpanTermQuery spanTermQuery2 = new SpanTermQuery( term2 );
SpanTermQuery spanTermQuery3 = new SpanTermQuery( term3 );
SpanTermQuery spanTermQuery4 = new SpanTermQuery( term4 );
SpanFirstQuery spanFirstQuery1 = new SpanFirstQuery( spanTermQuery3, intSpanEnd1 );
SpanQuery[] spanQuery1 = new SpanQuery[] { spanTermQuery2, spanTermQuery3 }; 
SpanNearQuery spanNearQuery1 = new SpanNearQuery( spanQuery1, intSlop1, boolean1 );
SpanOrQuery spanOrQuery1 = new SpanOrQuery( spanQuery1 );
Hits hits1 = (Hits) indexSearcher1.search( query1, 
SpanTermQuery( term5 );
PrefixQuery( term5 );
Term( name_A_1, termText1 );
Term( name_A_1, termText2 );
Term( name_A_1, termText1 );
Term( name_A_1, prefixText1 );
new BitSet(2); bitset.set(0);bitset.set(1);
return new DocIdBitSet( bitset ); } );
HitsVector expected_hitsVector1 = new HitsVector();
HitsVector hitsVector1 = (HitsVector) hits1.getHitDocs( );
assertTrue( hitsVector1.equals( expected_hitsVector1 ) );

Hits hits2 = (Hits) indexSearcher1.search( prefixQuery1, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { 
BitSet bitset = new BitSet(2); 
bitset.set(0);bitset.set(1); return new DocIdBitSet(bitset); } });
HitsVector expected_hitsVector2 = new HitsVector();
HitsVector hitsVector2 = (HitsVector) hits2.getHitDocs( );
assertTrue( hitsVector2.equals( expected_hitsVector2 ) );

Hits hits3 = (Hits) indexSearcher1.search( query1, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { 
BitSet bitset = new BitSet(2); 
bitset.set(0);bitset.set(1); return new DocIdBitSet(bitset); } });
HitsVector expected_hitsVector3 = new HitsVector();
HitsVector hitsVector3 = (HitsVector) hits3.getHitDocs( );
assertTrue( hitsVector3.equals( expected_hitsVector3 ) );

Hits hits4 = (Hits) indexSearcher1.search( spanFirstQuery1, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { 
BitSet bitset = new BitSet(2); 
bitset.set(0);bitset.set(1); return new DocIdBitSet(bitset); } });
HitsVector expected_hitsVector4 = new HitsVector();
HitsVector hitsVector4 = (HitsVector) hits4.getHitDocs( );
assertTrue( hitsVector4.equals( expected_hitsVector4 ) );

Hits hits5 = (Hits) indexSearcher1.search( spanNearQuery1, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { 
BitSet bitset = new BitSet(2); 
bitset.set(0);bitset.set(1); return new DocIdBitSet(bitset); } });
HitsVector expected_hitsVector5 = new HitsVector();
HitsVector hitsVector5 = (HitsVector) hits5.getHitDocs( );
assertTrue( hitsVector5.equals( expected_hitsVector5 ) );

Hits hits6 = (Hits) indexSearcher1.search( spanNotQuery1, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { 
BitSet bitset = new BitSet(2); 
bitset.set(0);bitset.set(1); return new DocIdBitSet(bitset); } });
HitsVector expected_hitsVector6 = new HitsVector();
HitsVector hitsVector6 = (HitsVector) hits6.getHitDocs( );
assertTrue( hitsVector6.equals( expected_hitsVector6 ) );

Hits hits7 = (Hits) indexSearcher1.search( spanOrQuery1, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { 
BitSet bitset = new BitSet(2); 
bitset.set(0);bitset.set(1); return new DocIdBitSet(bitset); } });
HitsVector expected_hitsVector7 = new HitsVector();

Model-Driven Framework for Context Dependent Testing of Components 281
HitsVector hitsVector7 = (HitsVector) hits7.getHitDocs();
assertTrue( hitsVector7.equals( expected_hitsVector7 ) );    // Test Case # 22

Hits hits8 = (Hits) indexSearcher1.search( query2, new Filter() {
    public DocIdSet getDocIdSet(IndexReader reader) {
        BitSet bitset = new BitSet(2); bitset.set(0); bitset.set(1);
        return new DocIdBitSet(bitset); }
}, sort1 );
HitsVector expected_hitsVector8 = expected_hitsVector1;
HitsVector hitsVector8 = (HitsVector) hits8.getHitDocs();
assertTrue( hitsVector8.equals( expected_hitsVector8 ) );    // Test Case # 23

Hits hits9 = (Hits) indexSearcher1.search( query2, sort1 );
HitsVector expected_hitsVector9 = expected_hitsVector1;
HitsVector hitsVector9 = (HitsVector) hits9.getHitDocs();
assertTrue( hitsVector9.equals( expected_hitsVector9 ) );    // Test Case # 24

TopDocs topDocs1 = (TopDocs) indexSearcher1.search( query2, intTopDocs1 );
HitsVector expected_hitsVector10 = expected_hitsVector1;
HitsVector hitsVector10 = (HitsVector) topDocs1.getScoreDocs();
assertTrue( hitsVector10.equals( expected_hitsVector10 ) );    // Test Case # 25

TopDocs topDocs2 = (TopDocs) indexSearcher1.search( multiPhraseQuery1, new Filter() {
    public DocIdSet getDocIdSet(IndexReader reader) {
        BitSet bitset = new BitSet(2); bitset.set(0); bitset.set(1);
        return new DocIdBitSet(bitset); }
}, intTopDocs1, sort1 );
HitsVector expected_hitsVector11 = expected_hitsVector1;
HitsVector hitsVector11 = (HitsVector) topDocs2.getScoreDocs();
assertTrue( hitsVector11.equals( expected_hitsVector11 ) );    // Test Case # 26

} catch ( Exception exp ) {
    System.out.println( exp.toString() );
    fail("Exception occurred during test case execution");
}

} //End of Method

} //End of Class
Test Suite for SMC3

package org.apache.lucene.search;

import java.util.BitSet;
import java.util.Vector;
import junit.framework.*;
import org.apache.lucene.analysis.*;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.index.*;
import org.apache.lucene.document.*;
import org.apache.lucene.document.Field.Index;
import org.apache.lucene.document.Field.Store;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.spans.*;
import org.apache.lucene.store.*;
import org.apache.lucene.util.DocIdBitSet;
import java.util.*;

public class TestSuite_smc3 extends TestCase {

    public static void main(String args[]) {
        TestSuite testSuite = new TestSuite(TestSuite_smc3.class);
        testSuite.run(new TestResult());
    }

    public void test_smc3() {
        try {
            RAMDirectory rAMDirectory1 = new RAMDirectory();
            SimpleAnalyzer simpleAnalyzer1 = new SimpleAnalyzer();
            boolean boolean1 = true;
            IndexWriter indexWriter1 = new IndexWriter(rAMDirectory1, simpleAnalyzer1, boolean1);
            IndexSearcher indexSearcher1 = new IndexSearcher(rAMDirectory1);
            String sortField1 = "author";
            Sort sort1 = new Sort(sortField1);
            int intTopDocs1 = 2;
            Integer integer1 = new Integer(intTopDocs1);
            int intSpanEnd1 = 3;
            Integer integer2 = new Integer(intSpanEnd1);
            int intSlop1 = 1;
            Integer integer3 = new Integer(intSlop1);
            String name_A_1 = "body";
            String value_A_1 = "model based testing uses software models";
            Store store_A_1 = Field.Store.YES;
            Index index_A_1 = Field.Index.ANALYZED;
            Field field_A_1 = new Field(name_A_1, value_A_1, store_A_1, index_A_1);
            queryParser1 = new QueryParser(name_A_1, simpleAnalyzer1);
            Document document1 = new Document();
            document1.add(field_A_1);
            document1.add(field_B_1);
            document1.add(field_C_1);
            indexWriter1.optimize();
            indexWriter1.close();
            String wildcardText1 = "mode*";
            String string1 = new String(wildcardText1);
            String fuzzyText1 = "moden~";
            String termText1 = "uses";
            String string3 = new String(termText1);
        }
    }
}
Model-Driven Framework for Context Dependent Testing of Components
HitsVector hitsVector1 = (HitsVector) hits1.getHitDocs();
assertTrue( hitsVector1.equals( expected_hitsVector1 ) ); // Test Case # 27

HitsVector expected_hitsVector2 = expected_hitsVector1;
HitsVector hitsVector2 = (HitsVector) hits2.getHitDocs();
assertTrue( hitsVector2.equals( expected_hitsVector2 ) ); // Test Case # 28

HitsVector expected_hitsVector3 = expected_hitsVector1;
HitsVector hitsVector3 = (HitsVector) hits3.getHitDocs();
assertTrue( hitsVector3.equals( expected_hitsVector3 ) ); // Test Case # 29

HitsVector expected_hitsVector4 = expected_hitsVector1;
HitsVector hitsVector4 = (HitsVector) hits4.getHitDocs();
assertTrue( hitsVector4.equals( expected_hitsVector4 ) ); // Test Case # 30

HitsVector expected_hitsVector5 = expected_hitsVector1;
HitsVector hitsVector5 = (HitsVector) hits5.getHitDocs();
assertTrue( hitsVector5.equals( expected_hitsVector5 ) ); // Test Case # 31

HitsVector expected_hitsVector6 = expected_hitsVector1;
HitsVector hitsVector6 = (HitsVector) hits6.getHitDocs();
assertTrue( hitsVector6.equals( expected_hitsVector6 ) ); // Test Case # 32

HitsVector expected_hitsVector7 = expected_hitsVector1;
HitsVector hitsVector7 = (HitsVector) hits7.getHitDocs();
assertTrue( hitsVector7.equals( expected_hitsVector7 ) ); // Test Case # 33

HitsVector expected_hitsVector8 = expected_hitsVector1;
HitsVector hitsVector8 = (HitsVector) hits8.getHitDocs();
assertTrue( hitsVector8.equals( expected_hitsVector8 ) ); // Test Case # 34

Vector expected_vector2 = expected_hitsVector1;
Vector vector2 = (Vector) vector1.clone();
assertTrue( vector2.equals( expected_vector2 ) ); // Test Case # 35

HitsVector expected_hitsVector9 = expected_hitsVector1;
HitsVector hitsVector9 = (HitsVector) topDocs1.getScoreDocs();
assertTrue( hitsVector9.equals( expected_hitsVector9 ) ); // Test Case # 36

HitsVector expected_hitsVector10 = expected_hitsVector1;
HitsVector hitsVector10 = (HitsVector) topDocs2.getScoreDocs();
assertTrue( hitsVector10.equals( expected_hitsVector10 ) ); // Test Case # 37

HitsVector expected_hitsVector11 = expected_hitsVector1;
HitsVector hitsVector11 = (HitsVector) topDocs3.getScoreDocs();
assertTrue( hitsVector11.equals( expected_hitsVector11 ) ); // Test Case # 38

} catch ( Exception exp ) {
    System.out.println( exp.toString() );
    fail("Exception occurred during test case execution");
}

} //End of Method

} //End of Class
Test Suite for SMC4

```java
package org.apache.lucene.search;

import java.util.BitSet;
import java.util.Vector;
import junit.framework.*;
import org.apache.lucene.analysis.*;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.index.*;
import org.apache.lucene.document.*;
import org.apache.lucene.document.Field.Index;
import org.apache.lucene.document.Field.Store;
import org.apache.lucene.document.Field.*;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.spans.*;
import org.apache.lucene.store.*;
import org.apache.lucene.util.DocIdBitSet;
import java.util.*;

public class TestSuite_smc6 extends TestCase {

    public static void main(String args[]) {
        TestSuite testSuite = new TestSuite(TestSuite_smc6.class);
        testSuite.run(new TestResult());
    }

    public void test_smc6() {
        try {
            RAMDirectory rAMDirectory1 = new RAMDirectory();
            SimpleAnalyzer simpleAnalyzer1 = new SimpleAnalyzer();
            boolean boolean1 = true;
            IndexWriter indexWriter1 = new IndexWriter(rAMDirectory1, simpleAnalyzer1, boolean1);
            IndexSearcher indexSearcher1 = new IndexSearcher(rAMDirectory1);
            String sortField1 = "author";
            Sort sort1 = new Sort(sortField1);
            int intTopDocs1 = 2;
            Integer integer1 = new Integer(intTopDocs1);
            int intSpanEnd1 = 3;
            Integer integer2 = new Integer(intSpanEnd1);
            int intSlop1 = 1;
            Integer integer3 = new Integer(intSlop1);
            String name_A_1 = "body";
            String value_A_1 = "model based testing uses software models";
            Store store_A_1 = Field.Store.YES;
            Index index_A_1 = Field.Index.ANALYZED;
            Field field_A_1 = new Field(name_A_1, value_A_1, store_A_1, index_A_1);
            String name_A_2 = "author";
            String value_A_2 = "model driven testing uses model transformation";
            Store store_A_2 = Field.Store.YES;
            Index index_A_2 = Field.Index.ANALYZED;
            Field field_A_2 = new Field(name_A_2, value_A_2, store_A_2, index_A_2);
            String name_A_3 = "body";
            String value_A_3 = "to automate testing, OCL can be used";
            Store store_A_3 = Field.Store.YES;
            Index index_A_3 = Field.Index.ANALYZED;
            Field field_A_3 = new Field(name_A_3, value_A_3, store_A_3, index_A_3);
            String name_B_1 = "author";
            String value_B_1 = "beizer";
            Store store_B_1 = Field.Store.YES;
            Index index_B_1 = Field.Index.ANALYZED;
            Field field_B_1 = new Field(name_B_1, value_B_1, store_B_1, index_B_1);
            String name_B_2 = "author";
            String value_B_2 = "putting";
            Store store_B_2 = Field.Store.YES;
            Index index_B_2 = Field.Index.ANALYZED;
            Field field_B_2 = new Field(name_B_2, value_B_2, store_B_2, index_B_2);
            String name_B_3 = "poston";
            String value_B_3 = "poston";
            Store store_B_3 = Field.Store.YES;
        }
    }
}
```
Index index_B_3 = Field.Index.ANALYZED;
Field field_B_3 = new Field( name_B_3, value_B_3, store_B_3, index_B_3 );
String name_C_1 = "pets";
String value_C_1 = "/birds/anatidae/duck";
Store store_C_1 = Field.Store.YES;
Index index_C_1 = Field.Index.ANALYZED;
Field field_C_1 = new Field( name_C_1, value_C_1, store_C_1, index_C_1 );
String name_C_2 = "pets";
String value_C_2 = "/birds/anatidae/swan";
Store store_C_2 = Field.Store.YES;
Index index_C_2 = Field.Index.ANALYZED;
Field field_C_2 = new Field( name_C_2, value_C_2, store_C_2, index_C_2 );
String name_C_3 = "pets";
String value_C_3 = "/birds/anatidae/seagull";
Store store_C_3 = Field.Store.YES;
Index index_C_3 = Field.Index.ANALYZED;
Field field_C_3 = new Field( name_C_3, value_C_3, store_C_3, index_C_3 );
String name_D_1 = "year";
String value_D_1 = "1980";
Store store_D_1 = Field.Store.YES;
Index index_D_1 = Field.Index.ANALYZED;
Field field_D_1 = new Field( name_D_1, value_D_1, store_D_1, index_D_1 );
String name_D_2 = "year";
String value_D_2 = "1981";
Store store_D_2 = Field.Store.YES;
Index index_D_2 = Field.Index.ANALYZED;
Field field_D_2 = new Field( name_D_2, value_D_2, store_D_2, index_D_2 );
String name_D_3 = "year";
String value_D_3 = "1982";
Store store_D_3 = Field.Store.YES;
Index index_D_3 = Field.Index.ANALYZED;
Field field_D_3 = new Field( name_D_3, value_D_3, store_D_3, index_D_3 );
Document document1 = new Document();
document1.add(field_A_1);
document1.add(field_B_1);
document1.add(field_C_1);
document1.add(field_D_1);
document2.add(field_A_2);
document2.add(field_B_2);
document2.add(field_C_2);
document2.add(field_D_2);
document3.add(field_A_3);
document3.add(field_B_3);
document3.add(field_C_3);
document3.add(field_D_3);
QueryParser queryParser1 = new QueryParser( name_A_1, simpleAnalyzer1 );
indexWriter1.optimize();
indexWriter1.close();
String rangeText1 = "[1980 TO 1981]";
String string1 = new String( rangeText1 );
String wildcardText1 = "mode*";
String string2 = new String( wildcardText1 );
String fuzzyText1 = "moden~";
String string3 = new String( fuzzyText1 );
String phraseText1 = "testing uses";
String string4 = new String( phraseText1 );
String prefixText1 = "birds";
String string5 = new String( prefixText1 );
String termText1 = "uses";
String string6 = new String( termText1 );
String termText2 = "model";
String string7 = new String( termText2 );
String termText3 = "based";
String string8 = new String( termText3 );
String termText4 = "driven";
String string9 = new String( termText4 );
String termText5 = "testing";
String string10 = new String( termText5 );
Term term1 = new Term( name_A_1, termText1 );
Term term2 = new Term( name_A_1, termText2 );
Term term3 = new Term( name_A_1, termText3 );
Term term4 = new Term( name_A_1, termText4 );
Term term5 = new Term( name_A_1, termText5 );
Term term6 = new Term( name_A_1, prefixText1 );

PrefixQuery prefixQuery1 = new PrefixQuery( term6 );
MultiPhraseQuery multiPhraseQuery11 = new MultiPhraseQuery( );
multiPhraseQuery11.add(term2);
multiPhraseQuery11.add(term5);

Query query1 = (Query) queryParser1.parse( rangeText1 );
Query query2 = (Query) queryParser1.parse( wildcardText1 );
Query query3 = (Query) queryParser1.parse( fuzzyText1 );
Query query4 = (Query) queryParser1.parse( phraseText1 );
Query query5 = (Query) queryParser1.parse( prefixText1 );

SpanTermQuery spanTermQuery1 = new SpanTermQuery( term1 );
SpanTermQuery spanTermQuery2 = new SpanTermQuery( term2 );
SpanTermQuery spanTermQuery3 = new SpanTermQuery( term3 );
SpanTermQuery spanTermQuery4 = new SpanTermQuery( term4 );
SpanTermQuery spanTermQuery5 = new SpanTermQuery( term5 );

SpanNearQuery spanNearQuery1 = new SpanNearQuery( spanTermQuery2, intSpanEnd1 );
SpanNotQuery spanNotQuery1 = new SpanNotQuery( spanTermQuery5, spanTermQuery4 );

HitDoc hitDoc1 = new HitDoc( score1, id1 );
HitDoc hitDoc2 = new HitDoc( score2, id2 );

Hits hits1 = (Hits) indexSearcher1.search( multiPhraseQuery11, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSet bitset = new BitSet(2); bitset.set(0); bitset.set(1); return new DocIdBitSet(bitset); } } );
HitsVector expected_hitsVector1 = new HitsVector();
expected_hitsVector1.add( hitDoc1 );
expected_hitsVector1.add( hitDoc2 );

Hits hits2 = (Hits) indexSearcher1.search( query4, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSet bitset = new BitSet(2); bitset.set(0); bitset.set(1); return new DocIdBitSet(bitset); } } );

Hits hits3 = (Hits) indexSearcher1.search( query1, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSet bitset = new BitSet(2); bitset.set(0); bitset.set(1); return new DocIdBitSet(bitset); } } );

Hits hits4 = (Hits) indexSearcher1.search( spanFirstQuery1, sort1 );

Hits hits5 = (Hits) indexSearcher1.search( query6, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSet bitset = new BitSet(2); bitset.set(0); bitset.set(1); return new DocIdBitSet(bitset); } } );

Hits hits6 = (Hits) indexSearcher1.search( spanFirstQuery1, sort1 );

HitsVector expected_hitsVector4 = expected_hitsVector1;
HitsVector expected_hitsVector5 = expected_hitsVector1;
HitsVector expected_hitsVector6 = expected_hitsVector1;
assertTrue( hitsVector7.equals( expected_hitsVector7 ) ); // Test Case # 56

Hits hits8 = (Hits) indexSearcher1.search( spanOrQuery1, sort1 );
HitsVector expected_hitsVector8 = expected_hitsVector1;
HitsVector hitsVector8 = (HitsVector) hits8.getHitDocs( );
assertTrue( hitsVector8.equals( expected_hitsVector8 ) ); // Test Case # 57

Hits hits9 = (Hits) indexSearcher1.search( query6, sort1 );
HitsVector expected_hitsVector9 = expected_hitsVector1;
HitsVector hitsVector9 = (HitsVector) hits9.getHitDocs( );
assertTrue( hitsVector9.equals( expected_hitsVector9 ) ); // Test Case # 58

Hits hits10 = (Hits) indexSearcher1.search( query2, sort1 );
HitsVector expected_hitsVector10 = expected_hitsVector1;
HitsVector hitsVector10 = (HitsVector) hits10.getHitDocs( );
assertTrue( hitsVector10.equals( expected_hitsVector10 ) ); // Test Case # 59

TopDocs topDocs1 = (TopDocs) indexSearcher1.search( query1, intTopDocs1 );
HitsVector expected_hitsVector11 = expected_hitsVector1;
HitsVector hitsVector11 = (HitsVector) topDocs1.getScoreDocs( );
assertTrue( hitsVector11.equals( expected_hitsVector11 ) ); // Test Case # 60

TopDocs topDocs2 = (TopDocs) indexSearcher1.search( spanFirstQuery1, intTopDocs1 );
HitsVector expected_hitsVector12 = expected_hitsVector1;
HitsVector hitsVector12 = (HitsVector) topDocs2.getScoreDocs( );
assertTrue( hitsVector12.equals( expected_hitsVector12 ) ); // Test Case # 61

TopDocs topDocs3 = (TopDocs) indexSearcher1.search( query2, intTopDocs1 );
HitsVector expected_hitsVector13 = expected_hitsVector1;
HitsVector hitsVector13 = (HitsVector) topDocs3.getScoreDocs( );
assertTrue( hitsVector13.equals( expected_hitsVector13 ) ); // Test Case # 62

TopDocs topDocs4 = (TopDocs) indexSearcher1.search( query4, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSetbitset = new BitSet(2); bitset.set(0);bitset.set(1); return new DocIdBitSet(bitset); } }, intTopDocs1, sort1 );

HitsVector expected_hitsVector14 = expected_hitsVector1;
HitsVector hitsVector14 = (HitsVector) topDocs4.getScoreDocs( );
assertTrue( hitsVector14.equals( expected_hitsVector14 ) ); // Test Case # 63

TopDocs topDocs5 = (TopDocs) indexSearcher1.search( spanFirstQuery1, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSetbitset = new BitSet(2); bitset.set(0);bitset.set(1); return new DocIdBitSet(bitset); } }, intTopDocs1, sort1 );

HitsVector expected_hitsVector15 = expected_hitsVector1;
HitsVector hitsVector15 = (HitsVector) topDocs5.getScoreDocs( );
assertTrue( hitsVector15.equals( expected_hitsVector15 ) ); // Test Case # 64

TopDocs topDocs6 = (TopDocs) indexSearcher1.search( spanNearQuery1, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSetbitset = new BitSet(2); bitset.set(0);bitset.set(1); return new DocIdBitSet(bitset); } }, intTopDocs1, sort1 );

HitsVector expected_hitsVector16 = expected_hitsVector1;
HitsVector hitsVector16 = (HitsVector) topDocs6.getScoreDocs( );
assertTrue( hitsVector16.equals( expected_hitsVector16 ) ); // Test Case # 65

TopDocs topDocs7 = (TopDocs) indexSearcher1.search( spanNotQuery1, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSetbitset = new BitSet(2); bitset.set(0);bitset.set(1); return new DocIdBitSet(bitset); } }, intTopDocs1, sort1 );

HitsVector expected_hitsVector17 = expected_hitsVector1;
HitsVector hitsVector17 = (HitsVector) topDocs7.getScoreDocs( );
assertTrue( hitsVector17.equals( expected_hitsVector17 ) ); // Test Case # 66

TopDocs topDocs8 = (TopDocs) indexSearcher1.search( spanOrQuery1, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSetbitset = new BitSet(2); bitset.set(0);bitset.set(1); return new DocIdBitSet(bitset); } }, intTopDocs1, sort1 );

HitsVector expected_hitsVector18 = expected_hitsVector1;
HitsVector hitsVector18 = (HitsVector) topDocs8.getScoreDocs( );
assertTrue( hitsVector18.equals( expected_hitsVector18 ) ); // Test Case # 67

TopDocs topDocs9 = (TopDocs) indexSearcher1.search( query2, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSetbitset = new BitSet(2); bitset.set(0);bitset.set(1); return new DocIdBitSet(bitset); } }, intTopDocs1, sort1 );

HitsVector expected_hitsVector19 = expected_hitsVector1;
HitsVector hitsVector19 = (HitsVector) topDocs9.getScoreDocs();
assertTrue(hitsVector19.equals(expected_hitsVector19)); // Test Case # 68

final Vector vector1 = new Vector();
indexSearcher1.search(query1, new HitCollector() { public void collect(int doc, float score) { vector1.add(new 
Integer(doc)); } });
Vector expected_vector2 = expected_hitsVector1;
Vector vector2 = (Vector) vector1.clone();
assertTrue(vector2.equals(expected_vector2)); // Test Case # 69

final Vector vector3 = new Vector();
indexSearcher1.search(query2, new HitCollector() { public void collect(int doc, float score) { vector3.add(new 
Integer(doc)); } });
Vector expected_vector4 = expected_hitsVector1;
Vector vector4 = (Vector) vector3.clone();
assertTrue(vector4.equals(expected_vector4)); // Test Case # 70

final Vector vector5 = new Vector();
indexSearcher1.search(query3, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSet bitset = new 
BitSet(2); bitset.set(0); bitset.set(1); return new DocIdBitSet(bitset); } }, new HitCollector() { public void collect(int doc, float score) { vector5.add(new Integer(doc)); } });
Vector expected_vector6 = expected_hitsVector1;
Vector vector6 = (Vector) vector5.clone();
assertTrue(vector6.equals(expected_vector6)); // Test Case # 71

final Vector vector7 = new Vector();
indexSearcher1.search(prefixQuery1, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSet bitset = new 
BitSet(2); bitset.set(0); bitset.set(1); return new DocIdBitSet(bitset); } }, new HitCollector() { public void collect(int doc, float score) { vector7.add(new Integer(doc)); } });
Vector expected_vector8 = expected_hitsVector1;
Vector vector8 = (Vector) vector7.clone();
assertTrue(vector8.equals(expected_vector8)); // Test Case # 72

final Vector vector9 = new Vector();
indexSearcher1.search(query2, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSet bitset = new 
BitSet(2); bitset.set(0); bitset.set(1); return new DocIdBitSet(bitset); } }, new HitCollector() { public void collect(int doc, float score) { vector9.add(new Integer(doc)); } });
Vector expected_vector10 = expected_hitsVector1;
Vector vector10 = (Vector) vector9.clone();
assertTrue(vector10.equals(expected_vector10)); // Test Case # 73

final Vector vector11 = new Vector();
indexSearcher1.search(query2, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSet bitset = new 
BitSet(2); bitset.set(0); bitset.set(1); return new DocIdBitSet(bitset); } }, new HitCollector() { public void collect(int doc, float score) { vector11.add(new Integer(doc)); } });
Vector expected_vector12 = expected_hitsVector1;
Vector vector12 = (Vector) vector11.clone();
assertTrue(vector12.equals(expected_vector12)); // Test Case # 74
}

} catch ( Exception exp ) {
System.out.println(exp.toString());
fail("Exception occured during test case execution");
}

//End of Method

} //End of Class
Test Suite for SMCS

```java
package org.apache.lucene.search;

import java.util.BitSet;
import java.util.Vector;
import junit.framework.*;
import org.apache.lucene.analysis.*;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.index.*;
import org.apache.lucene.document.*;
import org.apache.lucene.document.Field.Index;
import org.apache.lucene.document.Field.Store;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.spans.*;
import org.apache.lucene.store.*;
import org.apache.lucene.util.DocIdBitSet;
import java.util.*;

public class TestSuite_smc5 extends TestCase {
    public static void main( String args[] ) {
        TestSuite testSuite = new TestSuite(TestSuite_smc5.class);
        testSuite.run( new TestResult() );
    }

    public void test_smc5() {
        try {
            RAMDirectory rAMDirectory1 = new RAMDirectory();
            SimpleAnalyzer simpleAnalyzer1 = new SimpleAnalyzer();
            boolean boolean1 = true;
            IndexWriter indexWriter1 = new IndexWriter(rAMDirectory1, simpleAnalyzer1, boolean1);
            IndexSearcher indexSearcher1 = new IndexSearcher(rAMDirectory1);
            String sortField1 = "author";
            Sort sort1 = new Sort(sortField1);
            int intTopDocs1 = 2;
            Integer integer1 = new Integer(intTopDocs1);
            int intSpanEnd1 = 3;
            Integer integer2 = new Integer(intSpanEnd1);
            int intSlop1 = 1;
            Integer integer3 = new Integer(intSlop1);
            String name_A_1 = "body";
            String value_A_1 = "model based testing uses software models";
            Store store_A_1 = Field.Store.YES;
            Index index_A_1 = Field.Index.ANALYZED;
            Field field_A_1 = new Field(name_A_1, value_A_1, store_A_1, index_A_1);
            String name_A_2 = "body";
            String value_A_2 = "model driven testing uses model transformation";
            Store store_A_2 = Field.Store.YES;
            Index index_A_2 = Field.Index.ANALYZED;
            Field field_A_2 = new Field(name_A_2, value_A_2, store_A_2, index_A_2);
            String name_A_3 = "body";
            String value_A_3 = "to automate testing, OCL can be used";
            Store store_A_3 = Field.Store.YES;
            Index index_A_3 = Field.Index.ANALYZED;
            Field field_A_3 = new Field(name_A_3, value_A_3, store_A_3, index_A_3);
            String name_B_1 = "author";
            String value_B_1 = "beizer";
            Store store_B_1 = Field.Store.YES;
            Index index_B_1 = Field.Index.ANALYZED;
            Field field_B_1 = new Field(name_B_1, value_B_1, store_B_1, index_B_1);
            String name_B_2 = "author";
            String value_B_2 = "cutting";
            Store store_B_2 = Field.Store.YES;
            Index index_B_2 = Field.Index.ANALYZED;
            Field field_B_2 = new Field(name_B_2, value_B_2, store_B_2, index_B_2);
            String name_B_3 = "author";
            String value_B_3 = "poston";
            Store store_B_3 = Field.Store.YES;
        }
    }
}
```
Index index_B_3 = Field.Index.ANALYZED;
Field field_B_3 = new Field(name_B_3, value_B_3, store_B_3, index_B_3);
String name_C_1 = "pets";
String value_C_1 = "birds/anatidae/duck";
Store store_C_1 = Field.Store.YES;
Index index_C_1 = Field.Index.ANALYZED;
Field field_C_1 = new Field(name_C_1, value_C_1, store_C_1, index_C_1);
String name_C_2 = "pets";
String value_C_2 = "birds/anatidae/swan";
Store store_C_2 = Field.Store.YES;
Index index_C_2 = Field.Index.ANALYZED;
Field field_C_2 = new Field(name_C_2, value_C_2, store_C_2, index_C_2);
String name_C_3 = "pets";
String value_C_3 = "birds/anatidae/seagull";
Store store_C_3 = Field.Store.YES;
Index index_C_3 = Field.Index.ANALYZED;
Field field_C_3 = new Field(name_C_3, value_C_3, store_C_3, index_C_3);
String name_D_1 = "year";
String value_D_1 = "1980";
Store store_D_1 = Field.Store.YES;
Index index_D_1 = Field.Index.ANALYZED;
Field field_D_1 = new Field(name_D_1, value_D_1, store_D_1, index_D_1);
String name_D_2 = "year";
String value_D_2 = "1981";
Store store_D_2 = Field.Store.YES;
Index index_D_2 = Field.Index.ANALYZED;
Field field_D_2 = new Field(name_D_2, value_D_2, store_D_2, index_D_2);
String name_D_3 = "year";
String value_D_3 = "1982";
Store store_D_3 = Field.Store.YES;
Index index_D_3 = Field.Index.ANALYZED;
Field field_D_3 = new Field(name_D_3, value_D_3, store_D_3, index_D_3);
QueryParser queryParser1 = new QueryParser(name_A_1, simpleAnalyzer1);
Document document1 = new Document();
Document document2 = new Document();
Document document3 = new Document();
document1.add(field_A_1);
document1.add(field_B_1);
document1.add(field_C_1);
document1.add(field_D_1);
document2.add(field_A_2);
document2.add(field_B_2);
document2.add(field_C_2);
document2.add(field_D_2);
document3.add(field_A_3);
document3.add(field_B_3);
document3.add(field_C_3);
document3.add(field_D_3);
indexWriter1.optimize();
indexWriter1.close();
String termText1 = "OCL";
String string3 = new String( termText1 );
String termText2 = "driven";
String string4 = new String( termText2 );
String termText3 = "testing";
String string5 = new String( termText3 );
String prefixText1 = "duck";
String string7 = new String( prefixText1 );
Term term1 = new Term( name_A_1, termText1 );
Term term2 = new Term( name_A_1, termText2 );
Term term3 = new Term( name_A_1, termText3 );
SpanTermQuery spanTermQuery1 = new SpanTermQuery( term1 );
SpanTermQuery spanTermQuery2 = new SpanTermQuery( term2 );
SpanTermQuery spanTermQuery3 = new SpanTermQuery( term3 );
SpanQuery[] spanQuery1 = new SpanQuery[] { spanTermQuery1, spanTermQuery2, spanTermQuery3, spanTermQuery3 };
SpanOrQuery spanOrQuery1 = new SpanOrQuery( spanQuery1 );
SpanNotQuery spanNotQuery1 = new SpanNotQuery( spanTermQuery1, spanTermQuery3 );
float score1 = 1.0f;
int id1 = 0;
HitDoc hitDoc1 = new HitDoc( score1, id1 );
Term term4 = new Term( name_A_1, prefixText1 );

Model-Driven Framework for Context Dependent Testing of Components
PrefixQuery prefixQuery1 = new PrefixQuery(term4);
Hits hits1 = (Hits)indexSearcher1.search(prefixQuery1);
Hits hits2 = (Hits)indexSearcher1.search(prefixQuery1, new Filter() {
    public DocIdSet getDocIdSet(IndexReader reader) {
        BitSetbitset = new BitSet(2);
        bitset.set(0);
        bitset.set(1);
        return new DocIdBitSet(bitset);
    }
}, sort1);
Hits hits3 = (Hits)indexSearcher1.search(prefixQuery1, sort1);
HitsVector expected_hitsVector1 = new HitsVector();
expected_hitsVector1.add(hitDoc1);
HitsVector hitsVector1 = hits1.getHitDocs();
assertTrue(hitsVector1.equals(expected_hitsVector1));
HitsVector expected_hitsVector2 = expected_hitsVector1;
HitsVector hitsVector2 = hits2.getHitDocs();
assertTrue(hitsVector2.equals(expected_hitsVector2));
HitsVector expected_hitsVector3 = expected_hitsVector1;
HitsVector hitsVector3 = hits3.getHitDocs();
assertTrue(hitsVector3.equals(expected_hitsVector3));
TopDocs topDocs1 = (TopDocs)indexSearcher1.search(spanNotQuery1, intTopDocs1);
HitsVector expected_hitsVector4 = expected_hitsVector1;
HitsVector hitsVector4 = topDocs1.getScoreDocs();
assertTrue(hitsVector4.equals(expected_hitsVector4));
TopDocs topDocs2 = (TopDocs)indexSearcher1.search(spanOrQuery1, intTopDocs1);
HitsVector expected_hitsVector5 = expected_hitsVector1;
HitsVector hitsVector5 = topDocs2.getScoreDocs();
assertTrue(hitsVector5.equals(expected_hitsVector5));

} catch (Exception exp) {
    System.out.println(exp.toString());
    fail("Exception occurred during test case execution");
}

} //End of Method

} //End of Class
Test Suite for SMC6

```java
package org.apache.lucene.search;

import java.util.BitSet;
import java.util.Vector;
import junit.framework.*;
import org.apache.lucene.analysis.*;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.index.*;
import org.apache.lucene.document.*;
import org.apache.lucene.document.Field.Index;
import org.apache.lucene.document.Field.Store;
import org.apache.lucene.document.Field.*;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.spans.*;
import org.apache.lucene.store.*;
import org.apache.lucene.util.DocIdBitSet;
import java.util.*;

public class TestSuite_smc6 extends TestCase {

    public static void main(String args[]) {
        TestSuite testSuite = new TestSuite(TestSuite_smc6.class);
        testSuite.run(new TestResult());
    }

    public void test_smc6() {
        try {
            RAMDirectory rAMDirectory1 = new RAMDirectory();
            SimpleAnalyzer simpleAnalyzer1 = new SimpleAnalyzer();
            boolean boolean1 = true;
            IndexWriter indexWriter1 = new IndexWriter( rAMDirectory1, simpleAnalyzer1, boolean1 );
            IndexSearcher indexSearcher1 = new IndexSearcher( rAMDirectory1 );
            String sortField1 = "author";
            Sort sort1 = new Sort( sortField1 );
            int intTopDocs1 = 2;
            Integer integer1 = new Integer( intTopDocs1 );
            int intSpanEnd1 = 3;
            Integer integer2 = new Integer( intSpanEnd1 );
            int intSlop1 = 1;
            Integer integer3 = new Integer( intSlop1 );
            String name_A_1 = "body";
            String value_A_1 = "model based testing uses software models";
            Field field_A_1 = new Field( name_A_1, value_A_1, Field.Store.YES, Field.Index.ANALYZED);
            Field field_A_2 = new Field( name_A_2, value_A_2, Field.Store.YES, Field.Index.ANALYZED);
            Field field_A_3 = new Field( name_A_3, value_A_3, Field.Store.YES, Field.Index.ANALYZED);
            String name_B_1 = "author";
            String value_B_1 = "putting";
            Field field_B_1 = new Field( name_B_1, value_B_1, Field.Store.YES, Field.Index.ANALYZED);
            Field field_B_2 = new Field( name_B_2, value_B_2, Field.Store.YES, Field.Index.ANALYZED);
            Field field_B_3 = new Field( name_B_3, value_B_3, Field.Store.YES, Field.Index.ANALYZED);
```
Index index_B_3 = Field.Index.ANALYZED;
Field field_B_3 = new Field( name_B_3, value_B_3, store_B_3, index_B_3 );
String name_C_1 = "pets";
String value_C_1 = "/birds/anatidae/duck";
Store store_C_1 = Field.Store.YES;
Index index_C_1 = Field.Index.ANALYZED;
Field field_C_1 = new Field( name_C_1, value_C_1, store_C_1, index_C_1 );
String name_C_2 = "pets";
String value_C_2 = "/birds/anatidae/swan";
Store store_C_2 = Field.Store.YES;
Index index_C_2 = Field.Index.ANALYZED;
Field field_C_2 = new Field( name_C_2, value_C_2, store_C_2, index_C_2 );
String name_C_3 = "pets";
String value_C_3 = "/birds/anatidae/seagull";
Store store_C_3 = Field.Store.YES;
Index index_C_3 = Field.Index.ANALYZED;
Field field_C_3 = new Field( name_C_3, value_C_3, store_C_3, index_C_3 );
String name_D_1 = "year";
String value_D_1 = "1980";
Store store_D_1 = Field.Store.YES;
Index index_D_1 = Field.Index.ANALYZED;
Field field_D_1 = new Field( name_D_1, value_D_1, store_D_1, index_D_1 );
String name_D_2 = "year";
String value_D_2 = "1981";
Store store_D_2 = Field.Store.YES;
Index index_D_2 = Field.Index.ANALYZED;
Field field_D_2 = new Field( name_D_2, value_D_2, store_D_2, index_D_2 );
String name_D_3 = "year";
String value_D_3 = "1982";
Store store_D_3 = Field.Store.YES;
Index index_D_3 = Field.Index.ANALYZED;
Field field_D_3 = new Field( name_D_3, value_D_3, store_D_3, index_D_3 );
Document document1 = new Document();
Document document2 = new Document();
Document document3 = new Document();
document1.add(field_A_1);
document1.add(field_B_1);
document1.add(field_C_1);
document1.add(field_D_1);
document2.add(field_A_2);
document2.add(field_B_2);
document2.add(field_C_2);
document2.add(field_D_2);
document3.add(field_A_3);
document3.add(field_B_3);
document3.add(field_C_3);
document3.add(field_D_3);
QueryParser queryParser1 = new QueryParser( name_A_1, simpleAnalyzer1 );
indexWriter1.optimize();
indexWriter1.close();
String rangeText1 = "[1980 TO 1981]";
String string1 = new String( rangeText1 );
String wildcardText1 = "mode*";
String string2 = new String( wildcardText1 );
String fuzzyText1 = "moden~";
String string3 = new String( fuzzyText1 );
String phraseText1 = "testing uses";
String string4 = new String( phraseText1 );
String prefixText1 = "birds";
String string5 = new String( prefixText1 );
String termText1 = "uses";
String string6 = new String( termText1 );
String termText2 = "model";
String string7 = new String( termText2 );
String termText3 = "based";
String string8 = new String( termText3 );
String termText4 = "driven";
String string9 = new String( termText4 );
String termText5 = "testing";
String string10 = new String( termText5 );
Term term1 = new Term( name_A_1, termText1 );
Term term2 = new Term(name_A_1, termText2);
Term term3 = new Term(name_A_1, termText3);
Term term4 = new Term(name_A_1, termText4);
Term term5 = new Term(name_A_1, termText5);
Term term6 = new Term(name_A_1, prefixText1);
PrefixQuery prefixQuery1 = new PrefixQuery(term6);
MultiPhraseQuery multiPhraseQuery11 = new MultiPhraseQuery();
multiPhraseQuery11.add(term2);
multiPhraseQuery11.add(term5);
Query query1 = (Query) queryParser1.parse(rangeText1);
Query query2 = (Query) queryParser1.parse(wildcardText1);
Query query3 = (Query) queryParser1.parse(fuzzyText1);
Query query4 = (Query) queryParser1.parse(phraseText1);
Query query5 = (Query) queryParser1.parse(prefixText1);
Query query6 = (Query) queryParser1.parse(termText1);
SpanTermQuery spanTermQuery1 = new SpanTermQuery(term1);
SpanTermQuery spanTermQuery2 = new SpanTermQuery(term2);
SpanTermQuery spanTermQuery3 = new SpanTermQuery(term3);
SpanTermQuery spanTermQuery4 = new SpanTermQuery(term4);
SpanTermQuery spanTermQuery5 = new SpanTermQuery(term5);
SpanFirstQuery spanFirstQuery1 = new SpanFirstQuery(spanTermQuery2, intSpanEnd1);
SpanNearQuery spanNearQuery1 = new SpanNearQuery(spanTermQuery4, spanTermQuery3);
SpanNotQuery spanNotQuery1 = new SpanNotQuery(spanTermQuery5, spanTermQuery4);
SpanOrQuery spanOrQuery1 = new SpanOrQuery(spanQuery1);
float score1 = 1.0f;
int id1 = 0;
HitDoc hitDoc1 = new HitDoc(score1, id1);
float score2 = 1.0f;
int id2 = 1;
HitDoc hitDoc2 = new HitDoc(score2, id2);
Hits hits1 = (Hits) indexSearcher1.search(multiPhraseQuery11, new Filter()
{ public DocIdSet getDocIdSet(IndexReader reader) { BitSet bitset = new BitSet(2); bitset.set(0);bitset.set(1); return new DocIdBitSet(bitset); } });
HitsVector expected_hitsVector1 = new HitsVector();
expected_hitsVector1.add(hitDoc1);
expected_hitsVector1.add(hitDoc2);
HitsVector hitsVector1 = (HitsVector) hits1.getHitDocs();
assertTrue(hitsVector1.equals(expected_hitsVector1));
// Test Case # 50

Hits hits2 = (Hits) indexSearcher1.search(query4, new Filter()
{ public DocIdSet getDocIdSet(IndexReader reader) { BitSet bitset = new BitSet(2); bitset.set(0);bitset.set(1); return new DocIdBitSet(bitset); } });
HitsVector expected_hitsVector2 = new HitsVector();
expected_hitsVector2.add(hitDoc1);
expected_hitsVector2.add(hitDoc2);
HitsVector hitsVector2 = (HitsVector) hits2.getHitDocs();
assertTrue(hitsVector2.equals(expected_hitsVector2));
// Test Case # 51

Hits hits3 = (Hits) indexSearcher1.search(query1, new Filter()
{ public DocIdSet getDocIdSet(IndexReader reader) { BitSet bitset = new BitSet(2); bitset.set(0);bitset.set(1); return new DocIdBitSet(bitset); } });
HitsVector expected_hitsVector3 = new HitsVector();
expected_hitsVector3.add(hitDoc1);
expected_hitsVector3.add(hitDoc2);
HitsVector hitsVector3 = (HitsVector) hits3.getHitDocs();
assertTrue(hitsVector3.equals(expected_hitsVector3));
// Test Case # 52

Hits hits4 = (Hits) indexSearcher1.search(query6, new Filter()
{ public DocIdSet getDocIdSet(IndexReader reader) { BitSet bitset = new BitSet(2); bitset.set(0);bitset.set(1); return new DocIdBitSet(bitset); } });
HitsVector expected_hitsVector4 = new HitsVector();
expected_hitsVector4.add(hitDoc1);
expected_hitsVector4.add(hitDoc2);
HitsVector hitsVector4 = (HitsVector) hits4.getHitDocs();
assertTrue(hitsVector4.equals(expected_hitsVector4));
// Test Case # 53

Hits hits5 = (Hits) indexSearcher1.search(spanFirstQuery1, new Filter()
{ public DocIdSet getDocIdSet(IndexReader reader) { BitSet bitset = new BitSet(2); bitset.set(0);bitset.set(1); return new DocIdBitSet(bitset); } });
HitsVector expected_hitsVector5 = new HitsVector();
expected_hitsVector5.add(hitDoc1);
expected_hitsVector5.add(hitDoc2);
HitsVector hitsVector5 = (HitsVector) hits5.getHitDocs();
assertTrue(hitsVector5.equals(expected_hitsVector5));
// Test Case # 54

Hits hits6 = (Hits) indexSearcher1.search(query1, sort1);
HitsVector expected_hitsVector6 = new HitsVector();
expected_hitsVector6.add(hitDoc1);
expected_hitsVector6.add(hitDoc2);
HitsVector hitsVector6 = (HitsVector) hits6.getHitDocs();
assertTrue(hitsVector6.equals(expected_hitsVector6));
// Test Case # 55
assertTrue( hitsVector7.equals(expected_hitsVector7) );  // Test Case # 56

Hits hits8 = (Hits) indexSearcher1.search( spanOrQuery1, sort1 );
HitsVector expected_hitsVector8 = expected_hitsVector1;
HitsVector hitsVector8 = (HitsVector) hits8.getHitDocs();
assertTrue( hitsVector8.equals(expected_hitsVector8) );  // Test Case # 57

Hits hits9 = (Hits) indexSearcher1.search( query6, sort1 );
HitsVector expected_hitsVector9 = expected_hitsVector1;
HitsVector hitsVector9 = (HitsVector) hits9.getHitDocs();
assertTrue( hitsVector9.equals(expected_hitsVector9) );  // Test Case # 58

Hits hits10 = (Hits) indexSearcher1.search( query2, sort1 );
HitsVector expected_hitsVector10 = expected_hitsVector1;
HitsVector hitsVector10 = (HitsVector) hits10.getHitDocs();
assertTrue( hitsVector10.equals(expected_hitsVector10) );  // Test Case # 59

TopDocs topDocs1 = (TopDocs) indexSearcher1.search( query1, intTopDocs1 );
HitsVector expected_hitsVector11 = expected_hitsVector1;
HitsVector hitsVector11 = (HitsVector) topDocs1.getScoreDocs();
assertTrue( hitsVector11.equals(expected_hitsVector11) );  // Test Case # 60

TopDocs topDocs2 = (TopDocs) indexSearcher1.search( spanFirstQuery1, intTopDocs1 );
HitsVector expected_hitsVector12 = expected_hitsVector1;
HitsVector hitsVector12 = (HitsVector) topDocs2.getScoreDocs();
assertTrue( hitsVector12.equals(expected_hitsVector12) );  // Test Case # 61

TopDocs topDocs3 = (TopDocs) indexSearcher1.search( query2, intTopDocs1 );
HitsVector expected_hitsVector13 = expected_hitsVector1;
HitsVector hitsVector13 = (HitsVector) topDocs3.getScoreDocs();
assertTrue( hitsVector13.equals(expected_hitsVector13) );  // Test Case # 62

TopDocs topDocs4 = (TopDocs) indexSearcher1.search( query4, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSetbitset = new BitSet2; bitset.set0();bitset.set1(); return new DocIdBitSet(bitset); } }, intTopDocs1, sort1 );
HitsVector expected_hitsVector14 = expected_hitsVector1;
HitsVector hitsVector14 = (HitsVector) topDocs4.getScoreDocs();
assertTrue( hitsVector14.equals(expected_hitsVector14) );  // Test Case # 63

TopDocs topDocs5 = (TopDocs) indexSearcher1.search( spanFirstQuery1, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSetbitset = new BitSet2; bitset.set0();bitset.set1(); return new DocIdBitSet(bitset); } }, intTopDocs1, sort1 );
HitsVector expected_hitsVector15 = expected_hitsVector1;
HitsVector hitsVector15 = (HitsVector) topDocs5.getScoreDocs();
assertTrue( hitsVector15.equals(expected_hitsVector15) );  // Test Case # 64

TopDocs topDocs6 = (TopDocs) indexSearcher1.search( spanNearQuery1, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSetbitset = new BitSet2; bitset.set0();bitset.set1(); return new DocIdBitSet(bitset); } }, intTopDocs1, sort1 );
HitsVector expected_hitsVector16 = expected_hitsVector1;
HitsVector hitsVector16 = (HitsVector) topDocs6.getScoreDocs();
assertTrue( hitsVector16.equals(expected_hitsVector16) );  // Test Case # 65

TopDocs topDocs7 = (TopDocs) indexSearcher1.search( spanNotQuery1, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSetbitset = new BitSet2; bitset.set0();bitset.set1(); return new DocIdBitSet(bitset); } }, intTopDocs1, sort1 );
HitsVector expected_hitsVector17 = expected_hitsVector1;
HitsVector hitsVector17 = (HitsVector) topDocs7.getScoreDocs();
assertTrue( hitsVector17.equals(expected_hitsVector17) );  // Test Case # 66

TopDocs topDocs8 = (TopDocs) indexSearcher1.search( spanOrQuery1, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSetbitset = new BitSet2; bitset.set0();bitset.set1(); return new DocIdBitSet(bitset); } }, intTopDocs1, sort1 );
HitsVector expected_hitsVector18 = expected_hitsVector1;
HitsVector hitsVector18 = (HitsVector) topDocs8.getScoreDocs();
assertTrue( hitsVector18.equals(expected_hitsVector18) );  // Test Case # 67

TopDocs topDocs9 = (TopDocs) indexSearcher1.search( query2, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSetbitset = new BitSet2; bitset.set0();bitset.set1(); return new DocIdBitSet(bitset); } }, intTopDocs1, sort1 );
HitsVector expected_hitsVector19 = expected_hitsVector1;
HitsVector hitsVector19 = (HitsVector) topDocs9.getScoreDocs();
assertTrue( hitsVector19.equals( expected_hitsVector19 ) );  
// Test Case # 68

final Vector vector1 = new Vector();
indexSearcher1.search(query1, new HitCollector() { public void collect(int doc, float score) { vector1.add(new
Integer(doc)); } });

Vector expected_vector2 = expected_hitsVector1;
Vector vector2 = (Vector) vector1.clone();
assertTrue( vector2.equals( expected_vector2 ) );  
// Test Case # 69

final Vector vector3 = new Vector();
indexSearcher1.search(query2, new HitCollector() { public void collect(int doc, float score) { vector3.add(new
Integer(doc)); } });

Vector expected_vector4 = expected_hitsVector1;
Vector vector4 = (Vector) vector3.clone();
assertTrue( vector4.equals( expected_vector4 ) );  
// Test Case # 70

final Vector vector5 = new Vector();
indexSearcher1.search(query3, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSet bitset = new
BitSet(2); bitset.set(0); bitset.set(1); return new DocIdBitSet(bitset); } }, new HitCollector() { public void collect(int doc, float score) { vector5.add(new Integer(doc)); } });

Vector expected_vector6 = expected_hitsVector1;
Vector vector6 = (Vector) vector5.clone();
assertTrue( vector6.equals( expected_vector6 ) );  
// Test Case # 71

final Vector vector7 = new Vector();
indexSearcher1.search(prefixQuery1, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSet bitset = new
BitSet(2); bitset.set(0); bitset.set(1); return new DocIdBitSet(bitset); } }, new HitCollector() { public void collect(int doc, float score) { vector7.add(new Integer(doc)); } });

Vector expected_vector8 = expected_hitsVector1;
Vector vector8 = (Vector) vector7.clone();
assertTrue( vector8.equals( expected_vector8 ) );  
// Test Case # 72

final Vector vector9 = new Vector();
indexSearcher1.search(query1, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSet bitset = new
BitSet(2); bitset.set(0); bitset.set(1); return new DocIdBitSet(bitset); } }, new HitCollector() { public void collect(int doc, float score) { vector9.add(new Integer(doc)); } });

Vector expected_vector10 = expected_hitsVector1;
Vector vector10 = (Vector) vector9.clone();
assertTrue( vector10.equals( expected_vector10 ) );  
// Test Case # 73

final Vector vector11 = new Vector();
indexSearcher1.search(query2, new Filter() { public DocIdSet getDocIdSet(IndexReader reader) { BitSet bitset = new
BitSet(2); bitset.set(0); bitset.set(1); return new DocIdBitSet(bitset); } }, new HitCollector() { public void collect(int doc, float score) { vector11.add(new Integer(doc)); } });

Vector expected_vector12 = expected_hitsVector1;
Vector vector12 = (Vector) vector11.clone();
assertTrue( vector12.equals( expected_vector12 ) );  
// Test Case # 74

} catch ( Exception exp ) {
System.out.println( exp.toString() );
fail("Exception occurred during test case execution");
}

} //End of Method

} //End of Class
D.11: Methods added for comparing outputs

**Class: Hits**
```java
public Vector getHitDocs() {
    HitsVector hitsVector = new HitsVector();
    hitsVector.addAll(this.hitDocs);
    return hitsVector;
}
```

**Class: TopDocs**
```java
public HitsVector getScoreDocs() {
    ScoreDoc[] scoreDocs = this.scoreDocs;
    HitsVector hitsVector = new HitsVector();
    for(int i=0; i<scoreDocs.length; i++) {
        hitsVector.add(scoreDocs[i]);
    }
    return hitsVector;
}
```

**Class: HitsVector**
```java
class HitsVector extends Vector {
    public boolean equalsOrdered(Object object) {
        Vector vector;
        if( object.getClass().getName().equals("org.apache.lucene.search.TopDocs") ) {
            TopDocs topDocs = (TopDocs) object;
            vector = (Vector) topDocs.getScoreDocs();
        } else if( object.getClass().getName().equals("org.apache.lucene.search.TopFieldDocs") ) {
            TopFieldDocs topFieldDocs = (TopFieldDocs) object;
            vector = (Vector) topFieldDocs.getScoreDocs();
        } else if( object.getClass().getName().equals("org.apache.lucene.search.Hits") ) {
            Hits hits = (Hits) object;
            vector = (Vector) hits.getHitDocs();
        } else {
            vector = (Vector) object;
        }
        // check the number of documents
        if (this == null & vector == null) {
            return true;
        } else if (this == null & vector != null) {
            return false;
        } else if (this != null & vector == null) {
            return false;
        } else if (this.size() != vector.size()) {
            return false;
        }
        // check the documents returned in same order as expected
        for (int i = 0; i < this.size(); i++) {
            HitDoc doc = (HitDoc) this.get(i);
            // Different search methods return a collection of different types of objects
            String vectorType = vector.get(i).getClass().getName();
            if (vectorType.equals("org.apache.lucene.search.FieldDoc")
                || vectorType.equals("org.apache.lucene.search.ScoreDoc")) {
                ScoreDoc doc1 = new ScoreDoc(doc.id, doc.score);
                ScoreDoc doc2 = (ScoreDoc) vector.get(i);
                if (doc1 != doc2) return false;
            } else if (vectorType.equals("java.lang.Integer")) {
                int doc2 = ((Integer) vector.get(i)).intValue();
                if (doc1 != doc2) return false;
            } else {
                HitDoc doc2 = (HitDoc) vector.get(i);
                if (doc1 != doc2) return false;
            }
        }
        return true;
    }
}
```