MULTI-VIEW RETRIEVAL OF SOFTWARE DESIGN SPECIFICATIONS USING SIMILARITY ASSESSMENT APPROACH

ALHASSAN ADAMU

UNIVERSITI SAINS MALAYSIA

2017

DAPATAN SEMULA BERBILANG-PANDANGAN BAGI SPESIFIKASI REKABENTUK PERISIAN MENGGUNAKAN PENDEKATAN PENILAIAN PERSAMAAN

ABSTRAK

Kajian ini mengkaji bagaimana reka bentuk perisian yang dimodelkan menggunakan Bahasa Pemodelan Seragam (UML) boleh diguna semula. Satu pemahaman yang jelas adalah bahawa UML memodelkan sistem perisian dari perspektif yang berbeza tetapi berkaitan. Isu utama yang timbul apabila mengguna semula reka bentuk ini ialah, bagaimana persamaan antara artifak UML boleh dikira dari perspektif yang berbagai. Bagaimanapun, tiada pendekatan muktamad yang mengira persamaan antara artifak UML merentasi pandangan sambil memelihara konsistensi merentasi pandangan-pandangan ini. Sehubungan itu, tesis ini mencadangkan satu pendekatan penilaian persamaan baru yang memudahkan pengiraan persamaan antara artifak UML dari perspektif yang berbagai. Pendekatan utama adalah untuk mengira persamaan artifak UML dari tiga perspektif bebas iaitu perspektif struktur, perspektif fungsian dan perspektif kelakuan. Persamaan berbilang pandangan dikira sebagai hasil tambah berwajaran perspektif bebas dan kemudian keputusannya diskalakan oleh faktor yang dipanggil penalti tak-konsisten. Penalti tak-konsisten menangani pemetaan berkonflik antara gambarajah struktur dengan gambarajah-gambarajah fungsian dan antara gambarajah struktur dengan gambarajah-gambarajah kelakuan. Sebagai tambahan, satu teknik pra-penapisan untuk menapis jumlah model-model gudang sebelum peringkat dapatan semula juga diperkenalkan. Keputusan eksperimen menunjukkan bahawa teknik dapatan semula berbilang pandangan yang dicadangkan mengatasi teknik dapatan semula pandangan tunggal dalam mendapatkan semula projek perisian yang paling relevan dari gudang dengan ketepatan purata bermakna (*Mean Average Precision*) sehingga 92%, dan korelasi dengan usaha penggunaan

semula sehingga 83.9%.Tambahan pula, cadangan teknik pra-penapisan telah membawa kepada pengurangan masa dapatan semula dengan kira-kira satu faktor 10. Oleh itu, pendekatan berbilang pandangan dicadangkan untuk digunakan semasa penggunaan semula reka bentuk perisian.

MULTI-VIEW RETRIEVAL OF SOFTWARE DESIGN SPECIFICATIONS USING SIMILARITY ASSESSMENT APPROACH

ABSTRACT

This study examines how software designs that are modelled using Unified Modelling Language (UML) can be reused. A notable understanding is that UML model software systems from different but related perspectives. The main issues that arise when reusing these designs is how the similarity between the UML artifacts can be computed from multiple perspectives. However, there is no definitive approach that computes the similarity between the UML artifacts across the views while maintaining the consistency across these views. Consequently, this thesis proposes a new similarity assessment approach that facilitates the computation of similarity between UML artifacts from multiple perspectives. The primary approach is to compute the similarity of UML artifacts from three independent perspectives of structural, functional, and behavioural perspectives. The Multiview similarity is computed as weighted sum of the independent perspectives and then scaled by the result of factor called inconsistency penalty. The inconsistency penalty handles the conflicting mapping between structured diagram and functional diagrams and structured diagram with behavioural diagrams. Additionally, a pre-filtering technique to sieve out the number of repository models prior to retrieval stage is proposed. The experimental results show that the proposed Multiview retrieval approach outperformed the single view retrieval approach in retrieving the most relevant software projects from repository with Mean Average Precision of up to 92% and correlation with reuse effort of 83.9%. Furthermore, the proposed pre-filtering technique leads to significant reduction in retrieval time by approximately a factor of 10. Therefore, it is recommended to use Multiview approach during software design reuse.

MULTI-VIEW RETRIEVAL OF SOFTWARE DESIGN SPECIFICATIONS USING SIMILARITY ASSESSMENT APPROACH

by

ALHASSAN ADAMU

Thesis submitted in fulfilment of the requirements
for the degree of
Doctor of Philosophy

August 2017

ACKNOWLEDGEMENT

All praises are due to the Almighty Allah, the most beneficent and most merciful who granted me with patience and courage to make this thesis successful.

I would like to extend my sincere appreciation and profound gratitude to my main supervisor, Dr. Wan Mohd Nazmee Wan Zainon for his great interest, scholarly criticism, irreplaceable guides, complete patience, contribution, and support from inception to completion of this study. I have learned not only from his knowledge and experience but also from his remarkable personality. I am also very grateful and honoured for the help and support given by my co-supervisor Dr. Hamza Onoruoiza Salami.

I am greatly indebted to my brother, Abdullahi Adamu Kofa for making me realise the value of education and helping me to acquire it. I acknowledge the support and word of encouragement from Engr. Muhammad Hadi Ayagi, his support and contribution make my entire life journey easy.

Thank you to the research review committee members, Dr. Sharifah Mashita Syed Mohamad, Associate Prof. Dr. Cheah Yu-N and Dr. Sukumar Letchmunan for their help and contributions, I have really learned a lot from their comments.

Word cannot express my gratitude to my mother for the love and prayers, your prayers have always be my principle guide to my entire life.

I am very grateful to my friend Hamza Salihu Adamu who stood by my side since from the beginning of my PhD struggle up to the end of the program; no words can express my appreciation. Not to forget with the Nigerian Community in USM who help me in no small way, to make Malaysia a home away from home. I am grateful to my colleagues at home for their prayers, support, encouragement and assistance.

I would like to acknowledge the study leave and financial support given to me by Kano University of Science and Technology, Wudil and Tertiary Education Trust Fund, TETfund. Without their support and fund, I would not be able to finish my PhD. I want to extend my special gratitude to Dr. Musa Babayo, former chairman board of trustees TETfund for helping me to make my dream comes true.

I would not be writing these lines if not for the patience and understanding of my wife, Aisha Lawan Bala and my three gems; Amina, Abdallah and Muhammad who have been my emotional anchors through not only during my study years, but my entire life. They are also my backbone who keep me strong throughout the struggle of finishing my PhD. I thank my parents-in-law for their support and help since from my Master study to PhD.

Special thanks to my brother, Umar Saleh Anka for the concern and advices. To all my family members, I am grateful for your concerns and prayers during my study.

TABLE OF CONTENTS

ACKNOWLEDGEMENT ii

TABLE OF CONTENTS iv

LIST OF TABLES viii

LIST OF FIGURES x

LIST OF ABBREVIATIONS xii

ABSTRAK xiii

ABSTRACT xv

CHAPTER 1: INTRODUCTION 1

1.1 Introduction 1

1.2 Problem Statement 3

1.3 Research Questions 6

1.4 Research Objectives 7

1.5 Scope and Limitation 7

1.6 Research Methodology 8

1.7 Research Contributions 15

1.8 Thesis Structure 17

CHAPTER 2: LITERATURE REVIEW 18

2.1 Software Designs 18

2.1.1 Software Design Reuse 19

2.1.1(a) Software Product Lines 21

2.1.3(b) Unified Modelling Language 22

2.2 Software Retrieval 24

2.2.1 UML Retrieval Techniques 25

2.2.1(a) Information Retrieval 26
2.2.1(b) Case-based Reasoning 28
2.2.1(c) Ontology Based 30
2.2.1(d) Structural Based 32
2.2.1(e) Multi-view Retrieval Approach 35
2.2.1(f) Pre-filtering 37
2.3 Metaheuristics Search Techniques 40
2.3.1 Simulated Annealing (SA) 41
2.3.2 Hill Climbing (HC) 41
2.3.3 Genetic Algorithm (GA) 42
2.3.4 Cuckoo Search Algorithm (CSA) 42
2.4 Summary 43
CHAPTER 3: MULTI-VIEW RETRIEVAL OF SOFTWARE DESIGN 46
3.1 Retrieval Approach 46
3.2 Pre-filtering Process 47
3.2.1 Computation of Metric Based Similarity 48
3.3 Multiview Retrieval 53
3.3.1 Structural Similarity Computation 53
3.3.1(a) Classifiers' Similarity Matrix 54
3.3.1(b) Concept Similarity (CSim) 56
3.3.2 Functional Similarity Computation 65
3.3.2(a) Similarity Assessment Between Sequence Diagrams 65
3.3.2(b) Dynamic Programming Method for LCSMM 66
3.3.2(c) Similarity Score of Two Sequence Diagrams 68

3.3.2(d) Pairwise Similarity Assessment of Classes in Sequence Diagrams……………………………………………….67
3.3.2(e) Similarity Score of Two Sets of Sequence Diagrams 71
3.3.2(f) Similarity Score of Sets of Sequence Diagrams 74
3.3.3 Behavioural Similarity Computation 77
3.3.3(a) Graphical Representation of State Machine Diagrams 77
3.3.3(b) Similarity Measure 79
3.3.3(c) Pairwise Similarity of States in State Machine Diagrams .. 81
3.3.3(d) Similarity Computation for Sets of State Machines Diagrams 83
3.3.4 Aggregation of Similarity Measures 85
3.4 Retrieval Engine 88
3.4.1 Mapping with Hungarian algorithm (HGA) 88
3.4.2 Retrieval with Cuckoo search algorithm (CSA) 89
3.4.2(a) Matching of Classifiers using CSA 92
3.4.2(b) Matching of Sequence Diagrams using CSA 93
3.4.2(c) Matching of State Machine Diagrams using CSA 95
3.5 Summary 95
CHAPTER 4: EVALUATION 98
4.1 Retrieval System 98
4.2 Experiments 99
4.2.1 Structural View 99
4.2.1(a) Data Sample 99
4.2.1(b) Results and Findings 102
4.2.2 Functional View 107

4.2.2(a) Data Sample 107

4.2.2(b) Results and Findings 109

4.2.3 Behavioural View 112

4.2.3(a) Data Sample 112

4.2.3(b) Results and Findings 113

4.2.4 Aggregation Similarity 114

4.2.4(a) Aggregation Similarity (2views) 114

4.2.4(b) Aggregation Similarity (3views) 115

4.2.4(c) Results and Discussions 116

4.2.5 Pre-filtering 122

4.2.5(a) Results and Discussion 122

4. 3 Discussions 125

4. 3.1 Threats to Validity 128

4. 4 Summary 130

CHAPTER 5: CONCLUSIONS AND FUTURE WORK 131

5.1 Conclusions 131

5.2 Future Work 134

REFERENCES 136

LIST OF PUBLICATIONS 143

LIST OF TABLES

Page

Table 2- 1: Description of Metrics for Class and Sequence Diagrams 39

Table 3- 1: Description of State Machine Diagram Metrics used during Pre-filtering 50

Table 3- 2: Feature Vectors Extracted from Software Projects during Pre-filtering 51

Table 3- 3: Pre-filtering Similarity Matrix.. 52

Table 3- 4: Description of Features of each Classifiers (OMG, 2011) 54

Table 3- 5: Number of Classifiers Features appears in X of Figure 3-3 55

Table 3- 6: Classifiers Similarity Matrix M.. 55

Table 3- 7: Values for LCS_B used in computing LB... 67

Table 3- 8: Properties' Matrix of Classes in Sequence Diagrams of Figure 3-6... 70

Table 3- 9: Similarity Matrix of Sequence Diagrams Classes of Figure 3-6........ 70

Table 3- 10: Properties Matrix for Objects in Sequence Diagram in Figure 3-7.... 74

Table 3- 11: Properties Matrix for Sequence Diagram in Figure 3-7 75

Table 3- 12: Objects Similarity Matrix for Sequence Diagrams in Figure 3-7....... 75

Table 3- 13: Similarity Matrix for Sequence Diagrams in Figure 3-7.................... 75

Table 3- 14: Adjacency Matrix of s 78

Table 3- 15: Similarity Function (SF)... 78

Table 3- 16: Feature Set of each State in State Machine Diagram 80

Table 3- 17: States' Features of s and t.. 81

Table 3- 18: States' Similarity Matrix (*SSM*) of s and t 81

Table 4- 1: Description of Software in the Repository (Salami, 2015) 101

Table 4- 2: Description of Parameters Settings 102

Table 4- 3: Comparison of Results of the Similarity Assessments Techniques . 106

Table 4- 5: Description of state machines diagrams in the repository 112

Table 4- 6: Behavioral Similarity Assessment Results 113

Table 4- 7: Configurations weights for the Aggregation Similarity assessment experiments 115

Table 4- 8: Description of Query used for 3-views experiment 116

Table 4- 9: Description of Projects in the Repository 116

Table 4- 10: Performance of Standalone Pre-filtering 123

Table 4- 11: Comparison of MAP and Retrieval Time for different Retrieval Technique 124

LIST OF FIGURES

Page

Figure 1- 1: Research Methodology Flow 9

Figure 2- 1: Taxonomy of Reusable Software Artifacts 20

Figure 2- 2: Sample Class Diagrams and equivalent Graph Representation 33

Figure 3- 1: Multi-view Retrieval Approach 47

Figure 3- 2: Sample Diagram obtained by Reverse Engineering from Java Source Code 51

Figure 3- 3: Properties of Classifiers in Class Diagrams 54

Figure 3- 4: Class diagrams X and Y 55

Figure 3- 5: Hierarchical Representation of Conceptual Similarity Computation 57

Figure 3- 6: Example of Repository and Query Diagrams 64

Figure 3- 7: Two sample sequence diagrams a and b 68

Figure 3- 8: Two sets of Sequence Diagrams A and B 74

Figure 3- 9: State Machine Diagrams s 78

Figure 3- 10: Graph Representation of s 78

Figure 3- 11: Two State Machine Diagrams s and t 80

Figure 3- 12: Schematic Diagram for Computation of Aggregation of Similarity 87

Figure 3- 13: Cuckoo Search Algorithm 90

Figure 3- 14: Initialisation of Population 92

Figure 3- 15: Solution encoding for comparing two class diagrams 93

Figure 3- 16: Possible Nest Encoding for Similarity Assessment of Two Sets of Sequence Diagrams 94

Figure 3- 17: Possible Nest encoding for Comparing Sequence Diagrams of Figure 3-6 94

Figure 4- 1: Convergence Characteristics of CSA with HGA and CSA only...... 103

Figure 4- 2: Percentage of Time when CSA with HGA and CSA produce Fitter Fitness Values than each other 104

Figure 4- 3: Convergence Characteristics of CSA, GA and PSO 105

Figure 4- 4: Percentage of Time when CSA and GA produce better Fitness Values 105

Figure 4- 5: Examples of Sequence Diagrams in Repository formed from the Query 109

Figure 4- 6: Comparison of MAP of three Retrieval Methods............................ 110

Figure 4- 7: Time to search Repository for Three Methods................................ 111

Figure 4- 8 : Mean Average Precision for the three different method of diagrams retrieval 117

Figure 4- 9: Effects of different weight values on the retrieval methods............ 117

Figure 4- 10: Correlation with reuse efforts for all the methods........................... 118

Figure 4- 11: Time to search repository by each method...................................... 118

Figure 4- 12: Mean of MAP for single similarity assessment method and 3 similarity assessment methods 120

Figure 4- 13: Mean of MAP of two methods and 3 methods................................ 121

Figure 4- 14: Relationships between MAP and number of Projects returned after Pre-filtering 123

Figure 4- 15: Effects of Pre-filtering on MAP for different Retrieval Methods 124

LIST OF ABBREVIATIONS

UML	Unified Modelling Language
GA	Genetic Algorithm
CSA	Cuckoo Search Algorithm
PSO	Particle Swamp Optimization
SA	Simulated Annealing
HC	Hill Climbing
HGA	Hungarian Algorithm
IR	Information Retrieval
CBR	Case-based Reasoning
RSIM	Relationship Similarity
CSIM	Concept Similarity
MOOG	Message Object Order Graph
LD	Lavenshtein Distance
LCSMM	Longest Common Subsequence of Matching Messages
FUNC_Sim	Functional Similarity
STRUC_Sim	Structural Similarity
BEHV_Sim	Behavioral Similarity
AGR_Sim	Aggregation Similarity
MBSim	Metric-based Similarity

DAPATAN SEMULA BERBILANG-PANDANGAN BAGI SPESIFIKASI REKABENTUK PERISIAN MENGGUNAKAN PENDEKATAN PENILAIAN PERSAMAAN

ABSTRAK

Kajian ini mengkaji bagaimana reka bentuk perisian yang dimodelkan menggunakan Bahasa Pemodelan Seragam (UML) boleh diguna semula. Satu pemahaman yang jelas adalah bahawa UML memodelkan sistem perisian dari perspektif yang berbeza tetapi berkaitan. Isu utama yang timbul apabila mengguna semula reka bentuk ini ialah, bagaimana persamaan antara artifak UML boleh dikira dari perspektif yang berbagai. Bagaimanapun, tiada pendekatan muktamad yang mengira persamaan antara artifak UML merentasi pandangan sambil memelihara konsistensi merentasi pandangan-pandangan ini. Sehubungan itu, tesis ini mencadangkan satu pendekatan penilaian persamaan baru yang memudahkan pengiraan persamaan antara artifak UML dari perspektif yang berbagai. Pendekatan utama adalah untuk mengira persamaan artifak UML dari tiga perspektif bebas iaitu perspektif struktur, perspektif fungsian dan perspektif kelakuan. Persamaan berbilang pandangan dikira sebagai hasil tambah berwajaran perspektif bebas dan kemudian keputusannya diskalakan oleh faktor yang dipanggil penalti tak-konsisten. Penalti tak-konsisten menangani pemetaan berkonflik antara gambarajah struktur dengan gambarajah-gambarajah fungsian dan antara gambarajah struktur dengan gambarajah-gambarajah kelakuan. Sebagai tambahan, satu teknik pra-penapisan untuk menapis jumlah model-model gudang sebelum peringkat dapatan semula juga diperkenalkan. Keputusan eksperimen menunjukkan bahawa teknik dapatan semula berbilang pandangan yang dicadangkan mengatasi teknik dapatan semula pandangan tunggal dalam mendapatkan semula projek perisian yang paling relevan dari gudang dengan

ketepatan purata bermakna (*Mean Average Precision*) sehingga 92%, dan korelasi dengan usaha penggunaan semula sehingga 83.9%.Tambahan pula, cadangan teknik pra-penapisan telah membawa kepada pengurangan masa dapatan semula dengan kira-kira satu faktor 10. Oleh itu, pendekatan berbilang pandangan dicadangkan untuk digunakan semasa penggunaan semula reka bentuk perisian.

MULTI-VIEW RETRIEVAL OF SOFTWARE DESIGN SPECIFICATIONS USING SIMILARITY ASSESSMENT APPROACH

ABSTRACT

This study examines how software designs that are modelled using Unified Modelling Language (UML) can be reused. A notable understanding is that UML model software systems from different but related perspectives. The main issues that arise when reusing these designs is how the similarity between the UML artifacts can be computed from multiple perspectives. However, there is no definitive approach that computes the similarity between the UML artifacts across the views while maintaining the consistency across these views. Consequently, this thesis proposes a new similarity assessment approach that facilitates the computation of similarity between UML artifacts from multiple perspectives. The primary approach is to compute the similarity of UML artifacts from three independent perspectives of structural, functional, and behavioural perspectives. The Multiview similarity is computed as weighted sum of the independent perspectives and then scaled by the result of factor called inconsistency penalty. The inconsistency penalty handles the conflicting mapping between structured diagram and functional diagrams and structured diagram with behavioural diagrams. Additionally, a pre-filtering technique to sieve out the number of repository models prior to retrieval stage is proposed. The experimental results show that the proposed Multiview retrieval approach outperformed the single view retrieval approach in retrieving the most relevant software projects from repository with Mean Average Precision of up to 92% and correlation with reuse effort of 83.9%. Furthermore, the proposed pre-filtering technique leads to significant reduction in

retrieval time by approximately a factor of 10. Therefore, it is recommended to use Multiview approach during software design reuse.

CHAPTER 1

INTRODUCTION

This chapter briefly introduces the concepts of reuse by highlighting the terminologies and concepts that would be used in the remaining chapters. The chapter discusses the problem that initiated this research, defines the sets of objectives and the scope of the thesis. Furthermore, the chapter covers the methodology and contribution, as well as the organisation of the thesis.

1.1 Introduction

Software reuse is the creation of software system using previously developed software rather than development from the scratch (Frakes and Kyo, 2005). It helps to prevent the *reinvention of the wheel* during the software development. The benefit of software reuse includes accelerated software development, risk reduction process, effective use of specialists, reduction of development time, improvement of productivity and increase in the overall quality of software products (Al-Badareen et al., 2010). However, these advantages do not come without any drawbacks. According to Salami and Ahmed (2014c), some of the challenges of software reuse include increased effort to create and maintain components library, effort to find and adapt reusable components, lack of tool supports and increase in maintenance cost.

According to Kotonya et al. (2011) every year, more than $5 billion worth of software projects are cancelled or abandoned worldwide. Many of these projects are dropped not because their software failed but because the project objectives and

assumptions changed. Usually, the failed software projects are locked in potentially reusable software components. If we can find efficient ways to salvage and reuse these components, significant amount of the original investment can be recovered and new software can be developed rapidly at low-cost.

There are two types of software reuse: systematic and opportunistic (Kulkarni, 2013). In systematic reuse, software is particularly developed to be used in the future. This results in robust, well documented, and thoroughly tested artifacts. However, according to Salami and Ahmed (2014c), Keswani et al. (2014) these types of reuse requires time, effort and additional cost of making components reusable. Meanwhile, many organisations are unwilling to sacrifice since there is no guarantee that such components can be reused in the future. However, in opportunistic reuse, developers come to the conclusions that a component is reusable when they realise that the previously developed component can be used in the new software products. However, according to Salami and Ahmed (2014c) the components might not be in their best form of reuse. The UML retrieval techniques reported in this thesis can be utilised in both situations of software reuse mentioned above.

Software reuse can be carried out in four phases: representation, retrieval, adaptation, and incorporation (Park and Bae, 2011). During the representation phase, the fragment (i.e. query) of the software to be developed is presented. In the retrieval phase, the software components that are similar to the query with minimal adaptation cost are selected from the repository. During the adaptation, the components are modified to suite the need for the current software under development. Finally, in the incorporation phase, the new software components are integrated back to the repository for future reuse.

1.2 Problem Statement

There are different types of software artifacts that may be reused during software development. These artifacts include software requirements specifications, analysis, software design, source code, test cases, and documentations. These artifacts can be divided into early-stage and later-stage artifacts. The first three artifacts listed above are referred to as early-stage artifacts while the other artifacts are referred to as later-stage artifacts (Ahmed, 2011). The benefits of reusing early-stage has long being recognised in maximising the benefit of software reuse, because it leads to the reuse of corresponding later-stage artifacts (Rufai, 2003).

Early-stage artifacts such as software design artifacts are described utilising sets of models using Unified Modelling Languages (UML) diagrams. The UML is a *de facto* modelling language used by software developers during the initial stages of software development. Reusing of these models is challenging due to different reasons like the multi-dimensional nature of the modelling process, the variety of models to be designed, and the multiple perspectives of software systems which should be modelled (Lucas et al., 2009, Paydar and Kahani, 2015). For example, a structural perspective may describe static relationship between various software elements, while a behavioural perspective may describe the behaviour of software system.

The problem of reusing software design artifacts modelled using UML diagrams is the necessity to take into account the collective information contained in the multiple perspectives representation of software systems (Lucas et al., 2009). These perspectives describe a single system. They contain highly related and overlapping information. Therefore, similarity of software systems should be evaluated in a consistent manner by simultaneously considering the different perspectives of the

software system, rather than simply aggregating similarity values obtained from independent perspectives. Many of the researches investigating model reuse have focused on single view during retrieval, thus creating inconsistency between the UML models. For example, this is proven by the work of Park and Bae (2011) that compare class diagrams in one stage, and compare sequence diagrams in another stage. Another work by Salami and Ahmed (2014a) did not explicitly mention how the similarity of software system across multiple views can be computed. These inconsistencies among different models of a system may be a source of numerous errors for the software to be developed (Muskens et al., 2005).

The UML models consist of two type of information: (i) structural information which represents the structural representation of software system (for example, relationship between classes in class diagrams) and (ii) lexical information which represents the internal information of UML models (for example, class name, and attribute names). Matching of UML entities requires matching of both structural and lexical information of UML diagrams. Existing works on UML matching techniques can be categorised into four, which are; information retrieval (IR), case-based reasoning, ontology-based, and graph-based technique.

Traditionally, information retrieval technique is applied in web search engines. The IR provides techniques for comparing text documents and can be applied to all UML artifacts that contain a reasonable amount of text. Traditional IR techniques consider software artifacts equal if they contain the same words in the same frequency. The ambiguity problem emerges when two artifacts representation are similar but the actual meaning of the artifacts is different. For example, the words *customer* in one requirement specification and the word *client* in another requirement may be

considered different by IR even though their actual meaning are the same. Furthermore, IR does not take into account the structural information of UML artifacts, therefore two UML diagrams with the same words frequency but with opposite structural representation are considered equal by IR.

Ontology-based techniques such as WordNet specify concepts and the relationship among those concepts especially those that are in the same or similar domain. It defines concepts based on the notion of synset (synonyms) built on their length in the WordNet graph. Consequently, two concepts with opposite meaning are considered equal if there is short distance in their path length.

The graph-based technique, on the other hand relies on the structural representation of UML artifacts. It measures the similarity of two artifacts by comparing the vertices and arcs of their equivalent graph representation. The similarity of UML artifacts is computed by comparing the subgraph using taxonomic comparison of elements and their relationship to other elements. The drawback of this technique is that only structural information of UML artifacts are considered during similarity computation neglecting the lexical information inside the diagrams.

In the process of exploring large repositories, there are many competing constraints that need to be fulfilled due to the large number of models in the repository, thus widening the search space. In exploring large repository, the search space can be exponential since huge number of candidate solutions need to be analysed. Accordingly, finding mapping that produces optimal similarity of UML artifacts represents an NP-hard problem. It would thereby cause the retrieval stage to be computationally expensive, especially when the size of the projects in the repository are large. Few existing works such as the work of Channarukul et al. (2005) and Gomes

et al. (2003) performed pre-filtering using common class diagram names between the diagrams and using WordNet respectively. However, since WordNet is utilised during pre-filtering of the repository diagrams, many of the diagrams which have similar names in meaning are likely to be returned, thereby making the retrieval stage computationally expensive. Recently Salami (2015) proposed a pre-filtering, using software metrics that describe some properties of software system based on class and sequence diagrams. However, the number of repository returned at the end of pre-filtering are fixed, thereby defeating the aim of pre-filtering stage if the number of repository projects returned at the end of the pre-filtering are large.

1.3 Research Questions

Considering the research problem as outlined previously, the research questions of this thesis emerge as follows:

1. What are the suitable measures for determining the similarity of UML artifacts from multiple perspectives?
2. What are the appropriate matching techniques that can be employed during UML artifacts retrieval?
3. How can we pre-filter repository models when the size is large, and what among the UML artifacts information (e.g. metric, lexical) can best be used during pre-filtering?
4. What is the suitable proportion of software artifacts that can be returned after pre-filtering?

1.4 Research Objectives

This research aims to study the retrieval of early-stage software artifacts modelled using UML diagrams. In more detail, it seeks to fulfil the following research objectives:

1. To design an efficient technique for determining the similarity between UML software artefacts from multiple perspectives by comparing their lexical and structural properties.

2. To devise and design a pre-filtering technique for improving the efficiency of UML artifacts retrieval from large software repository.

3. To determine the suitable proportion of repository projects to be returned after pre-filtering.

1.5 Scope and Limitation

This research focuses on computing the similarity between software projects containing class diagrams, sequence diagrams and state machine diagrams. These three diagrams represent the structural, functional, and behavioural views of the software systems. The information derived from these diagrams represent the different perspectives of a software system.

The structural perspectives of software system are usually presented using the following diagrams: class, components, objects, deployment, package, composite, and profile. However, according to Ahmed (2011), Al-Khiaty and Ahmed (2016) only class diagrams are used during the requirement engineering to represent the structure of the system. Other diagrams are mostly used to explain the small piece of classes

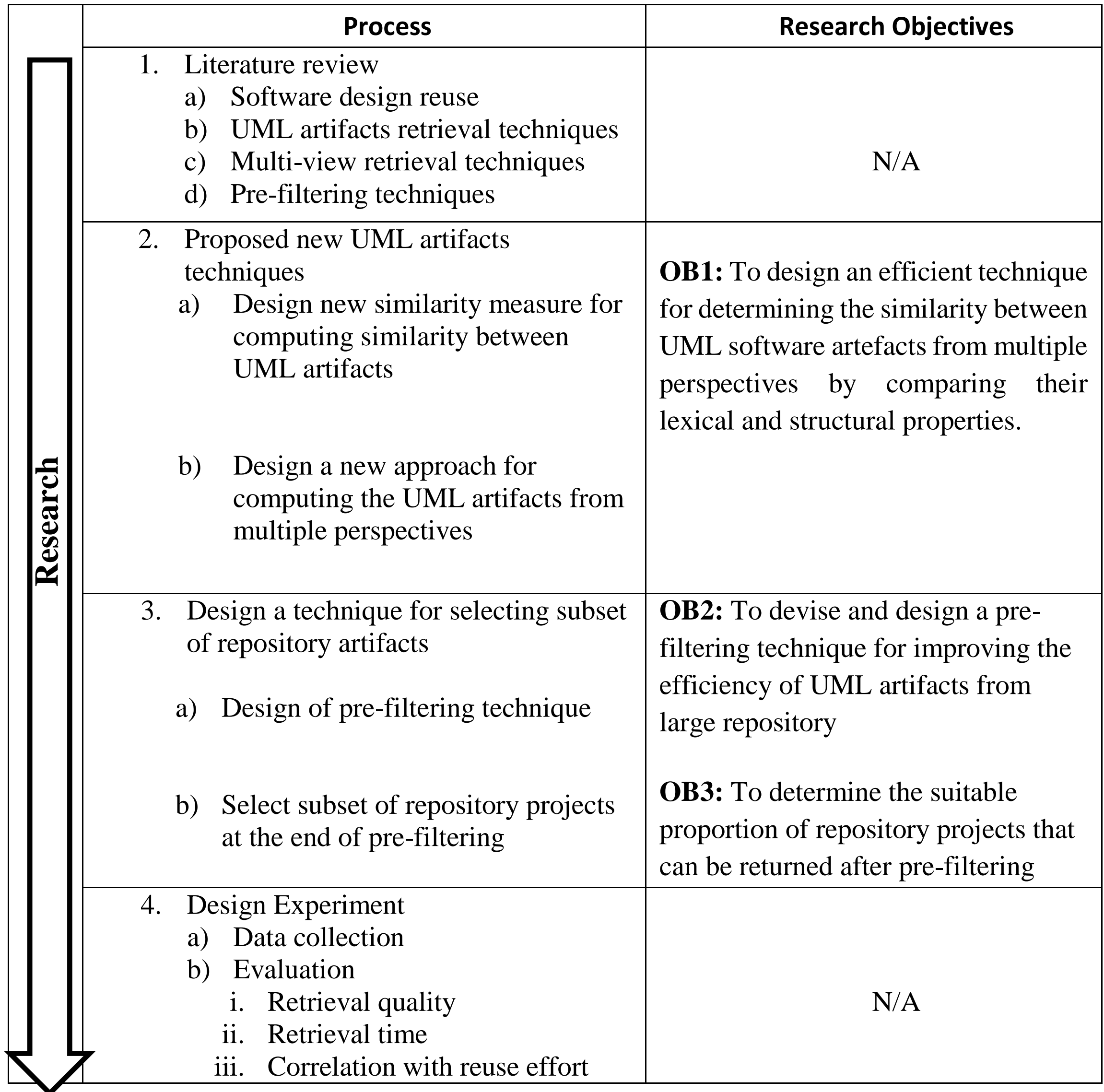
with complicated relationship. Considering this, our structural similarity assessment relies only on class diagrams.

Use case diagrams are usually employed to capture the functionalities of a software system. During requirement phase, each of the use case can be represented by one or more sequence diagrams which depicts how objects interact and work together to provide service (Ahmed, 2011). Considering this, the functional similarity assessment method relies only on sequence diagrams.

Behavioural views of the software system are mostly captured using state machine diagrams. The diagram represents the system from two different levels: system level and object level. In the system level, the state machine diagrams are used to show the system behaviour in response to user actions, while at the object level they show the dynamic behaviour of objects. Other behaviour diagrams include activity diagrams, and interaction diagrams. However, according to Ahmed (2011), these diagrams are mostly used during the architectural and design phases to express artifacts at different design phases. Only state machine diagrams are used during requirements to show the flow of event within or between objects. Hence, this study's behavioural similarity assessment method relies only on state machine diagrams.

1.6 Research Methodology

In order to achieve the research objectives stated in section 1.4, the research methodology is divided into four main phases as illustrated in Figure 1-1.



Process	Research Objectives
1. Literature review a) Software design reuse b) UML artifacts retrieval techniques c) Multi-view retrieval techniques d) Pre-filtering techniques	N/A
2. Proposed new UML artifacts techniques a) Design new similarity measure for computing similarity between UML artifacts b) Design a new approach for computing the UML artifacts from multiple perspectives	**OB1:** To design an efficient technique for determining the similarity between UML software artefacts from multiple perspectives by comparing their lexical and structural properties.
3. Design a technique for selecting subset of repository artifacts a) Design of pre-filtering technique b) Select subset of repository projects at the end of pre-filtering	**OB2:** To devise and design a pre-filtering technique for improving the efficiency of UML artifacts from large repository **OB3:** To determine the suitable proportion of repository projects that can be returned after pre-filtering
4. Design Experiment a) Data collection b) Evaluation i. Retrieval quality ii. Retrieval time iii. Correlation with reuse effort	N/A

Figure 1- 1: Research Methodology Flow

Phase 1 - In this phase, all relevant literatures had been reviewed to gain an extensive idea on how different UML software artifacts were matched and retrieved from software repositories. The literature studied surrounded software reuse, software design reuse, software retrieval, UML artifacts retrieval, the application of metaheuristics algorithms in software engineering problems. The IEEE explorer, ACM Library, Science Direct, Google Scholar, Springer were used as the main sources of knowledge. Some preliminary work in terms of discussion and literature survey was

conducted to gain more information on how software engineers (especially software developers) reuse previous software designs.

The outcomes of the literature study guided the researcher on the strengths and weaknesses of the existing approaches. At the end of the literature, this study found that the existing works lack standard techniques to consistently map different UML diagrams from multiple perspectives during retrieval of software project designs from repositories. Therefore, the inference of this phase is the design of a new approach for retrieving of UML software artifacts from multiple perspectives.

Phase 2– Based on the study done in the first phase, several similarity measures were designed to enhance the similarity assessment between UML diagrams artifacts. The similarity measures compute the similarity between UML entities (e.g. class names). It measured the presence or absence of similar features between two UML artifacts.

The proposed similarity measure are based on:

(i) Substring similarity assessment method which relied on the use of Levenshtein distance to compute the similarity between concepts in UML diagrams.

(ii) Longest common subsequence (LCS) which compute the similarity between sequence diagrams as the length of common subsequence of matching messages between the two sequence diagrams.

(iii) Graph-based approach which computes the similarity of two UML diagrams by comparing the vertices and the edges of the graph.

At the end of the similarity assessment, a ranked list of requirement specifications are returned to the reuser. Requirement specifications at the top list are the most similar

to the new requirement specifications. Thus, the adaptation of the corresponding artifacts (for example design, code, and documentation) from the repository should require the least time and effort. During the similarity assessment, heuristics algorithm is employed to aid the matching and retrieval of UML artifacts from repository.

Furthermore, in this phase an approach for computing the similarity of UML artifacts from multiple perspectives is presented since software systems are modelled using multiple UML diagrams. This approach is referred to as Multiview similarity assessment method, in which the similarity between UML artifacts is calculated as an aggregation of independent perspectives of the UML artifacts. The independent perspectives are:

(i) Structural perspective, which relied on the information contained in class diagram.

(ii) Functional perspective, which relied on the information contained in sequence diagrams.

(iii) Behavioural perspective, which relied on the information contained in state machine diagrams.

The Multiview similarity is scaled with an inconsistency penalty factor which handles the conflicting mapping between structure diagram and functional diagram as well as between structural diagram and behavioural diagram. Details of the proposed similarity assessment methods is discussed in Chapter 3.

Phase 3 – In this phase, the approach of selecting subset of repository models prior to retrieval is proposed. The phase consisted of designing of pre-filtering technique and selecting of subset of repository projects at the end of pre-filtering.

The aim of pre-filtering stage was to minimise the retrieval time by selecting first set of repository artifacts in a computationally inexpensive stage prior to retrieval stage. This stage is particularly important when the repository contain many projects. In this stage, metadata of the new requirement specification is compared with the metadata of the repository projects. The metadata collected at this stage is the metric data such as total number of classes in a class diagram, number of messages exchanged by objects in sequence diagrams, and the number of attributes and operations of classes in class diagrams.

To ensure this stage is computationally inexpensive, the metadata are obtained from requirements specifications when new projects are stored in the repository for the first time. The metadata of the repository are updated whenever changes were made. However, the metadata of the new software are obtained in the pre-filtering stage, since it only becomes available at this stage. At the end of the pre-filtering stage, subset of repository are selected and returned for subsequent comparison in the retrieval stage.

Phase 4 – Experiment was carried out in the final phase. Evaluation of similarity assessments approach were conducted to assess the effectiveness of the proposed retrieval approach. The evaluation was based on three criteria: (i) retrieval quality (ii) retrieval time and (iii) correlation with reuse efforts. In order to perform the evaluation, data were collected for the experiments. The output of this phase lead us to some conclusions regarding this research (see chapter 4). Details are discussed in due cause.

According to Zhang (2006), data scarcity is a common problem to most software engineering research. Since there were no available software reuse repositories containing UML diagrams, this study relied on reverse engineered class and sequence

diagrams using AltovaUModel®[1]. Previous researchers also relied on the reverse engineering, for example Assunçao and Vergilio (2013) used ObjectAid UML Explorer[2] to reverse source code to class diagram. A repository containing different families of open source software was created. Each of the family of software contained different versions of the software family. It could be argued that different releases of the same software were more similar to themselves than other software. The UML diagrams in the repository (see Table 4-1) used for several experiments had 11-66 number of classifiers and 15-254 sequence diagrams containing 172-92921 messages. Similarly, other datasets contained UML diagrams of different sizes which belong to several domains.

Retrieval quality referred to the number of projects retrieved after similarity assessment. This study relied on the standard measure used to measure the information retrieval system to evaluate the quality of the artifacts retrieved from the repository. Mean Average Precision (MAP) is widely used for evaluating ranked retrieval systems.

Average precision (AP) for a given query is obtained using precision values calculated at each point whenever a new projects is retrieved (i.e. precision = 0 for each of the relevant project that is not retrieved). The Mean Average Precision for a set of query is the mean of the AP scores for each query, also referred to as *mean precision at seen relevant projects* (Teufel, 2007). The formula is given in Equation 1-1 as follows:

---

[1] http://www.altova.com/
[2] http://www.objectaid.com/

$$MAP = \frac{1}{N}\sum_{j=1}^{N}\frac{1}{Q_j}\sum_{i=1}^{Q_j}P(rel = i) \tag{1-1}$$

N is the number of queries, Q_j is the number of relevant documents for query j and $P(rel=i)$ is the precision at the ith relevant document.

One of the expected gains of software reuse is the decrease in the software development time. A retrieval strategy with great precision and recall yet with unsuitable long retrieval time may not be used by the reuser. Subsequently, the retrieval time of our reuse approach is gauged as the time taken to retrieve similar projects from repository.

A reuse system might have the capacity to retrieve relevant projects from repository with high MAP. It is possible that the similarity scores returned by the system might be meaningless. The system may just be great in ranking the repository projects. In order to overcome this problem, the degree of correlation between similarity scores returned by the reuse system and estimated modification (reuse) effort would be analysed. The significant amount of reuse effort is dedicated to programming. Code-based sizing metrics would be used to estimate reuse effort. The estimated reuse effort would be calculated using formula in Equation 1-2 for predicting software maintenance effort (Basili et al., 1996).

$$Estimated\ reuse\ effort = (0.36 * effective\ SLOC) + 1040 \tag{1-2}$$

The SLOC (source lines of codes) is the sum of added, deleted, and modified SLOC. It is computed using Unified Coding tool[3]. The strong correlation between similarity score and estimated reuse efforts indicated the similarity score returned by the reuse system could provide a reuser with rough estimates of the amount of effort required to adapt the retrieved software projects to suit the need of the new system to be developed.

1.7 Research Contributions

This section summarises the main contributions of this thesis by describing the approach introduced.

1. Software systems are typically modelled from different viewpoints rather than single view. An approach for computing the similarity of software system from multiple perspectives is presented. The Multiview similarity of software systems is computed as an aggregation of structural, functional, and behavioural views of software systems:
    i. Structural perspectives: The structural perspective relies on the information contained in class diagrams. Some of the contributions of this thesis in the area of class diagram based retrieval of software include: development of a similarity measure for computing the similarity between software projects using the concepts names in class diagrams; identification of suitable features for computing the similarity between classifiers in class diagrams; and determination of suitable approach for matching of classifiers in a class diagram.

[3] http://sunset.usc.edu/ucc_wp/

ii. Functional perspectives: the functional similarity assessment of software systems is based on the information contained in sequence diagrams. The similarity computation between sequence diagrams can be split in to two: (i) the similarity of sequence diagrams is computed from their longest common matching messages. (ii) Since software systems are hardly modelled using single sequence diagrams, an approach of assessing the similarity of set of sequence diagrams is also presented.

iii. Behavioural perspectives: The behaviour of software systems are usually manifested using state machine diagrams. An approach of assessing the similarity of state machine diagram by converting state machine into equivalent directed graph is presented. The similarity of two state machine diagrams is computed by comparing the node and edges of the graph. Additionally, a method computing the similarity of sets of state machine diagrams is presented since software system are usually modelled using multiple state machine diagrams.

2. Usually, a repository contains many software models. Retrieval time may be very high and this can out weight the benefit of reuse. A fast way of identifying subset of repository projects that are potentially similar to the query is proposed. A Multiview pre-filtering approach based on structural, functional, and behavioural perspectives of software systems. Furthermore, the proposed pre-filtering approach automatically determined the proportion of the repository projects to be returned after the pre-filtering. The shortlisted projects are then compared in a more computationally demanding retrieval stage.

1.8 Thesis Structure

The rest of this thesis is organised as follows:

Chapter 2 reviews previous researches and discusses preliminary knowledge related to this thesis. Literature review on existing techniques that are currently available to address early-stage artifacts reuse are presented. It also includes a brief description of background knowledge on software retrieval and metaheuristics algorithms.

Chapter 3 describes the proposed work. It presents the techniques for retrieving software system from repository. An approach for pre-filtering the set of repository projects prior to retrieval is also presented. It describes the proposed similarity assessment techniques for comparing UML diagrams. It presents several similarity measures for assessing the similarity between class diagram, sequence diagrams, and state machine diagrams. In addition, an aggregation similarity method is presented to compute the similarity of software projects from multiple perspectives.

Chapter 4 describes several experiments conducted to evaluate the UML retrieval approach proposed in this thesis. Finally, **Chapter 5** concludes and provides some possible suggestions for improvements of future work associated with the research.

CHAPTER 2

LITERATURE REVIEW

This chapter provides background knowledge on software reuse and reviews existing works on early-stage reuse. The chapter is divided into four main sections. Section 2.1 discusses software designs and reuse. Section 2.2 presents discussion on software retrieval and UML retrieval techniques. Section 2.3 presents discussion on metaheuristics search algorithm. Section 2.4 presents the summary of the chapter.

2.1 Software Designs

Software development process comprises of three important phases: the requirement and analysis phase, the system design phase, and the implementation phase. This work focused on software design phase. According to Gomes (2004) and Robles et al. (2012), the design phase is important in the software life-cycle because most of the decisions made at this phase have great influence over the other phases. It is also a task that is more complex than the analysis phase because it requires more expertise and know-how from the developers. In addition, the knowledge at the design stage describes the fundamental of software system abstraction and their relationships, and these knowledge are more abstract and less formal than knowledge in the coding phase. Therefore, if software development companies could store and retrieve their knowledge effectively at the early-stage of the software lifecycle, it could be possible to improve the software development process.

Software system design phase can be divided into two levels: architectural design, and detail design. The architectural design is related to the conceptual designs of the system in which the problems and their solutions are analysed. The system entities and the subsystems that comprise the system models are defined. The concern of this level is more to the requirement analysis phase rather than the implementation phase. The output of this phase is the conceptual design that identifies the software architecture, so that it was able to satisfy the specification produced in the analysis phase. The detailed design on the other hand is related to the implementation and coding phases. The detailed design describes how codes are organised. The output of this phase is data structures and algorithms for coding purposes (Tawosi et al., 2015). In both cases, the software design phase is closely tied to other software system phase.

2.1.1 Software Design Reuse

Reuse has long been recognised as the hope for the software engineering community since it started, with the main expectation of reducing the development cost and time (Ahmed, 2011). The reuse of source code are largely been used. However, it takes a small portion of reuse since it is performed at lower level, neglecting the advantage of reuse of bigger software construct. Most of the widespread existing reuse tools for indexing and searching in the market are quite generic and are only based on code search or component search, which are usually based on keywords. Specialised tools for retrieving the software designs are lacking, because it was difficult to abstract and represent the knowledge produced in this phase (Robles et al., 2012).

According to Gomes (2003), in the last two decades, some forms of design reuse have emerged in the literature. This form includes frameworks reuse, design pattern

reuse, and the software product line reuse. These types of reuse are much more promising than the usual form of code reuse and they are getting more significant in the software reuse community. Typically reusable artifacts are divided into two: early stage and later stage artifacts. The early-stage artifacts include requirement specification, analysis, and designs while the later-stage artifacts include implementation, test cases, and documentations (Rufai, 2003). Figure 2-1 shows the taxonomy of software reuse artifacts.

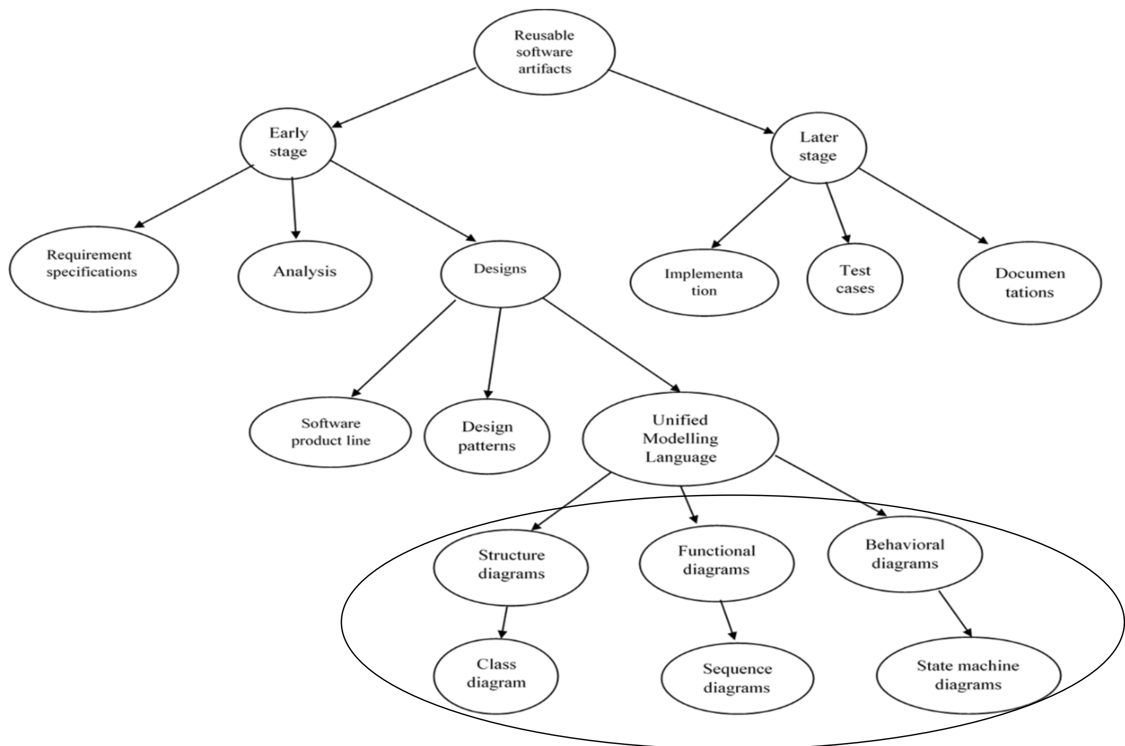


Figure 2- 1: Taxonomy of Reusable Software Artifacts (Rufai, 2003, Ahmed, 2011)

2.1.1(a) Design Pattern Reuse

Design Patterns are defined as descriptions of problems and their solutions for common design problems (Tsantalis et al., 2006). Design patterns provide a technique to document solutions for recurring problems and sharing those solutions in an application-independent fashion (Bayley and Zhu, 2010). Patterns provide high level

form of reuse as they function at the architectural level and detail designs. They can be viewed as guidelines for providing ways of assembling entities in the form of classes and interfaces.

Patterns consist of four essential elements: name, problem, solutions, and statement. The name identifies the design pattern and the name should be meaningful for reference to the pattern. The problem defines the problem area of the design patterns and the situation that the patterns intend to solve. The solution describes the parts of the design solutions, the relationship, and their responsibilities. A statement is the outcome of the patterns describing the consequences of the pattern applications. A statement assists the designers in understanding whether or not a pattern can be used in a particular situation or not (Hsueh et al., 2008). Design patterns support software reusability. However, according to Hasheminejad and Jalili (2012), it is difficult to find the right patterns for reuse. Finding the right design patterns to a given problem heavily relies on the expertise of the software developers, and it is extremely difficult for the novice developers that are not familiar with design pattern to find the right pattern for reuse.

2.1.1(b) Software Product Lines

Software product lines are sets of software systems that share common architecture and share components. Each of the software is specialised to reflect different requirements. The core value of software product line is the design of the software systems that suite the needs of different customers. Software product lines promote software reuse by building pre-planned family of software product that are in the same domain (Shatnawi et al., 2016). Software product lines usually occur in existing

software systems. This happens when organisations develop an application and similar application is needed to be developed for different customers. A portion of the previous application code is informally used to develop the new application. As more applications are developed, the changes tend to corrupt the original application structure which increasingly makes it difficult to create new versions. Software product lines often reflected a general, application-specific architectural style or patterns (Sommerville, 2011). While acknowledging the importance of software product line in software reuse, this study is constrained with the limitation of domain specific of the software product line, since the research focused on the reuse of software system from multiple domains.

2.1.3(c) Unified Modelling Language

Unified modelling language (UML) is a general purpose modelling language that graphically represents systems requirements and designs and was accepted by the International Organisation for Standardisation (ISO) as a standard specification. The UML provides diagrams for visualising, specifying and documenting software systems (Torres et al., 2011). The UML comprises a set of diagrams that can be used to model a software system. The diagrams are categorised into two: structural diagrams which document static structure of system objects, and behavioural diagrams which shows the behaviour of system objects (Salami and Ahmed, 2014c). Each of the category represents a particular aspect of software systems to be developed. Collectively, they provide complete software systems. The UML taxonomy of diagrams consider only structure and behaviour diagrams, without any category for the functional diagrams. However, according to Ahmed (2011), use case and sequence diagrams could be

interpreted as a mean of specifying the functionality of a system. Each use case diagram can be represented by one or more sequence diagrams which depict how objects interact and work together to provide services. Hence, this thesis considered sequence diagrams as the representative of functional perspectives of software systems. Subsequently, the thesis briefly discusses class diagrams, sequence diagrams, and state machine diagrams which represent the structural, functional, and behavioural views of software systems.

A class diagram is a blue print of an object that shares the same attribute and methods. It depicts the structure of a system by showing the system's classes and the relationships among the classes. Class diagram has three properties: the class name, the attributes which are the variables within the class, and the methods which define the actions a class can perform.

A sequence diagram captures the behaviour of a use case by showing the interaction between objects arranged in time order. The vertical dimension in a sequence diagram represents time, while the horizontal dimension represents the objects participating in an interaction. The directed arrow represents the messages on sequence diagrams (Rumbaugh et al., 2004).

A state diagram describes the system behaviour by showing how objects respond to events according to its current state, and how it enters new states (Rumbaugh et al., 2004). The common use of this diagram is to show how an object behaves during its lifetime. The basic notational elements of state machine diagrams are rounded rectangle which represents state; an arrow representing the transitions between the state; a filled cycle denoting the initial state; and a hollow circle containing filled circle denoting the final state.

2.2 Software Retrieval

Retrieval of relevant software from repository is an important task in software reuse (Assunçao and Vergilio, 2013). Typically, software project in repository will have several UML diagrams, and can have different interpretation depending on the software systems' goals and domains. The relevancy of a project with the problem at hand is generally defined based on the similarity between the projects and the problem specification.

During retrieval, matching and similarity scoring are employed to asses and rank shortlisted repository projects. Matching refers to the mapping of entities in one model to other entities in the same or similar models to be compared. It also defines the conditions under which models are selected from repository (Park and Bae, 2011). Thus, matching is a combinatorial optimisation problem, and one of the heuristic search technique described in section 2.3 is employed to aid the matching of model entities. The similarity scoring on the other hand, focused on measuring the semantic relatedness of different concepts (Sun et al., 2013). Usually, projects are ranked using a similarity metric, which assesses the degree of similarity between the target problem and the repository projects to be ranked.

An example of Similarity Function is shown in Equation 2.1, where Q is the target problem, R is the repository projects, w_i are the weight associated with the concepts ($\sum w_i = 1$), *CSim* is the concept similarity, q_i is a problem concept, r_i is repository projects concepts and n is the number of projects in the repository. Weights are a way of assigning different importance to concepts.