ONTOLOGY-BASED SOURCE CODE RETRIEVAL MODEL TO SUPPORT PROGRAM COMPREHENSION

ROZITA BINTI KADAR

UNIVERSITI SAINS MALAYSIA

2019

ONTOLOGY-BASED SOURCE CODE RETRIEVAL MODEL TO SUPPORT PROGRAM COMPREHENSION

by

ROZITA BINTI KADAR

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

November 2019

ACKNOWLEDGEMENT

In the name of Allah, the Most Gracious and the Most Merciful.

Alhamdulillah, praise to the almighty who has given me the strength and patience to keep on holding in the journey to acquire knowledge.

I wish to express my most profound thanks to my supervisors, Professor Dr. Putra Sumari, Dr. Sharifah Mashita Syed Mohamad, and Associate Professor Dr. Nur'Aini Abdul Rashid for their invaluable advice, guidance, inspiration and moral support.

In addition, I would like to thank the Malaysia Ministry of Higher Learning, and Universiti Teknologi MARA (UiTM) for their outstanding management, to provide financial support and study leave.

To my husband, Mohd Zamri, my children, Luqman, Rijal, Najwa, Harith and Raudhah, thank you for making Ummi complete. My special thanks also goes to my mother and my mother in law, my siblings, to my relatives, and dearest friends for their love, encouragements, patience and always being with me. This research will not be completed without you.

Last but not least , this PhD is dedicated to my late father Allahyarham Kadar Hj Zain and late father in law Allahyarham Udin Mohamed.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF SYMBOLS

=	Equals
{ }	Set - a collection of elements
[1 … *n*]	Close interval
( )	Parentheses
∪	OR, UNION
∩	AND, JOIN
∋	Such as
| |	Absolute value
$\sqrt{\ }$	Square root
Σ	Summation – Sum of all values in range of series
SD	Standard deviation
α	Alpha - significance level
$\bar{x}$	Mean

LIST OF ABBREVIATIONS

AOIG	Action-Oriented Identifier Graph
ASDG	Abstract System Dependence Graph
CL	Concept Location
CR	Change Request
CSM	Cosine Similarity Measurement
FCA	Formal Concept Analysis
FEAT	Feature Exploration and Analysis
FN	False Negative
FP	False Positive
GATE	General Architecture for Text Engineering
grepOF	Grasp Ontology Fragment
ICA	Independent Component Analysis
IEC	The International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IR	Information Retrieval
IROF	Information Retrieval Ontology Fragment
ISO	The International Organization for Standardization
JPDA	Java Platform Debugger Architecture
KLOC	Kilo Line of Code
LDA	Latent Dirichlet Allocation
LM	Language Model
LSI	Latent Semantic Indexing
NLP	Natural Language Processing
OntoBased-SR	Ontology-Based Source Code Retrieval
OOP	Object-oriented Programming
OWL	Ontology Web Language
PDA	Prune Dependency Graph
RDF	Resource Description Framework
SDG	System Dependence Graph

SDV	Single Value Decomposition
SEMERU	Software Engineering Maintenance and Evolution Research Unit
SITIR	Single Trace and Text Retrieval
SVD	Singular Value Decomposition
TF	Term Frequency
TFM	Term Frequency Matrix
TN	True Negative
TP	True Positive
TPTP	Test and Performance Tools Platform
UML	Unified Modeling Language
VSM	Vector Space Model
XML	Exchange Markup Language

MODEL CAPAIAN KOD SUMBER BERASASKAN ONTOLOGI UNTUK MENYOKONG PEMAHAMAN PROGRAM

ABSTRAK

Mengenal pasti bahagian-bahagian kod sumber sepadan dengan fungsi tertentu adalah salah satu aktiviti yang biasa dilakukan oleh penyelenggara perisian semasa menjalankan tugas penyelenggaraan. Tugas ini merupakan cabaran utama kepada mereka kerana mereka perlu memahami sistem perisian terlebih dahulu. Pelbagai teknik telah dicadangkan untuk membantu penyelenggara perisian mengurangkan usaha mereka untuk mencari lokasi kod sumber yang akan diberi perhatian. Konsep lokasi adalah salah satu teknik yang dapat memberikan penyelesaian kepada permasalahan ini. Teknik yang paling umum untuk menyokong konsep lokasi adalah capaian teks. Oleh itu, kajian ini memberi tumpuan kepada penggunaan kod sumber untuk teknik capaian teks dalam mengenal pasti lokasi yang relevan untuk diselenggarakan. Kandungan sumber data yang digunakan sebagai maklumat dalam proses capaian kod sumber adalah salah satu elemen yang dapat meningkatkan prestasi teknik ini. Kebanyakan teknik yang sedia ada tidak mengambilkira kepentingan maklumat pengetahuan kod sumber. Memandangkan pentingnya maklumat pengetahuan digunakan sebagai sumber data, kajian ini meneroka dan menemui perwakilan yang boleh mewakili maklumat pengetahuan kod sumber dan mencadangkan teknik baru untuk perwakilan maklumat pengenalan kod sumber berkonsepkan sistem perisian pada perspektif ontologi. Kajian ini menghasilkan model yang digunakan dalam teknik capaian kod sumber, dikenali sebagai **Model Capaian Kod Sumber Berasaskan Ontologi (OntoBased-SR)**. Idea asas kerja yang dicadangkan ini bertujuan untuk memperkaya maklumat yang akan digunakan sebagai

sumber maklumat dalam proses capaian, memanfaatkan penggunaan maklumat leksikal dan pengetahuan dalam kod sumber. Kajian ini menyiasat secara terperinci bagaimana maklumat pengetahuan dapat disepadukan dengan maklumat leksikal untuk menghasilkan maklumat yang lebih baik. Sumbangan utama kajian ini adalah mencadangkan model untuk menghasilkan sumber maklumat baru untuk teknik capaian kod sumber yang dihasilkan daripada penyepaduan maklumat pengetahuan dan maklumat leksikal. Eksperimen telah dijalankan dan keputusan dari hasil kerja yang dicadangkan menggunakan sumber data yang mengandungi pengetahuan dan maklumat leksikal telah dibandingkan dengan penanda aras. Penanda aras hanya menggunakan maklumat leksikal sebagai sumber data. Metrik ukur digunakan untuk menganalisis data untuk mengukur keberkesanan proses capaian semula. Keputusan menunjukkan bahawa model yang dicadangkan lebih berkesan dalam mencapai dokumen yang relevan berbanding dengan penanda aras. Keputusan lain adalah untuk menguji keupayaan hasil kerja yang dicadangkan untuk menyusun kedudukan dokumen yang relevan dengan pertanyaan yang diberikan. Keputusan menunjukkan bahawa hasil kerja yang dicadangkan dapat menyusun kedudukan dokumen yang relevan lebih baik berbanding dengan penanda aras. Oleh itu, dapat disimpulkan bahawa, penyelenggara perisian tidak memerlukan banyak usaha untuk mencari dokumen yang relevan untuk membuat perubahan jika kerja yang dicadangkan ini digunakan untuk menyokong tugas penyelenggaraan.

ONTOLOGY-BASED SOURCE CODE RETRIEVAL MODEL TO SUPPORT PROGRAM COMPREHENSION

ABSTRACT

Identifying the parts of source code corresponding to a specific functionality is one of the most common activities undertaken by software maintainers while performing maintenance tasks. This task is a key challenge to them since they need to comprehend a software system in advance. Various techniques have been proposed to help software maintainers to reduce their effort in finding location of source code concern. Concept location is one of the techniques that able to provide a solution to the problem. The most common technique to support concept location is text retrieval. Therefore, this study focuses on the use of source code for text retrieval technique in identifying relevant location to be maintained. The contents of data source that use as an information in source code retrieval process is one of elements that able to improve the performance of the technique. Most of the existing works did not consider the importance of knowledge information of source code. Considering the important of knowledge information use as a data source, this study explores and find the acceptable representation of source code to represent knowledge information and proposes a novel technique for source code retrieval process by conceptualising a software system on an ontological perspective. The work produce a model applied in the source code retrieval technique, namely **Ontology-Based Source Code Retrieval Model (OntoBased-SR)**. The basic idea of the proposed work intended to enrich the information that will be used as a source of information in retrieval process, leveraging the lexical and knowledge information in the source code. This work investigates in more detail how knowledge information can be best integrated with the lexical

information in order to generate better information. The main contribution of this study is in proposing a model for producing a new source of information for source code retrieval technique resulted from the integration of knowledge and lexical information. Experiments have been conducted and the result of the proposed work uses data source that contain knowledge and lexical information has been compared with a benchmark. The benchmark uses lexical information only as data source. The measurement metrics were used to analyse the data to measure the effectiveness of retrieval process. The result shows that the propose model is more effective in retrieving the relevant documents compared to the benchmark. Another result is to test the capability of the proposed work to rank the relevant documents response to the query. Finding shows that the proposed work can perform better in term of ranking the relevant documents compared to benchmark. It can therefore be concluded that, software maintainers do not required much effort to find relevant documents to make changes if the proposed work used to support maintenance tasks.

CHAPTER 1

INTRODUCTION

1.1 Background of the Study

When a software system is in operation, it needs to be maintained to improve its usefulness. Software maintenance is important in software engineering as its goal is to satisfy users' expectation. The term 'maintenance' refers to an evolution, which is a process of continuous development activities from a low, simple and worst system to a higher, more complex and better state (Jin & Cordy, 2005). Software maintenance is required in providing cost-effective support since cost and time are the major constraints to software maintenance process (Abran *et al*.,2004; Fakhoury *et al*., 2018; Koushik & Selvarani, 2012; Saleem *et al*., 2009; Salvaneschi *et al*., 2017).

Program comprehension is necessary in performing maintenance tasks and mainly takes place before changing any process. Software maintainers must be familiar and comprehend the parts of source code in the program to be maintained (Alhindawi *et al*., 2014; Wang, 2017). However, most of software maintainers face a problem in comprehending a software system when implementing maintenance tasks (Alhindawi *et al*., 2014; Krüger *et al*., 2019).

In maintaining the software system that needs to be changed, software maintainers must firstly identify the locations in the source code that are the most relevant to the intended changes (Dit *et al*., 2013; Marcus & Haiduc, 2013; Poshyvanyk *et al*., 2012; Schankin *et al*., 2018). Thus, the task of identifying the source code for a particular function is the key requirement in maintaining the software

(Alhindawi *et al*., 2014; Dit *et al*., 2013; Poshyvanyk *et al*., 2012; V. Rajlich & Gosavi, 2004). Hence, this present study attempts to overcome the problem in software maintenance by proposing a suitable technique to improve the identification of source code location before performing maintenance tasks.

1.1.1 Program Comprehension in Software Maintenance

Program comprehension is a very important activity in software maintenance since software maintainers need to understand the program to be maintained or to implement changes. Studies on program comprehension have been carried out since the early 1970s and revealed its importance as a major activity during software maintenance (Hu *et al*., 2018; Maletic & Kagdi, 2008; Sasirekha & Hemalatha, 2011; Xia *et al*., 2017).

The effectiveness of program comprehension activities depends on the speed of software engineers in understanding the program, thereby maintaining the system software efficiently. Previous studies explored the techniques used to overcome the problems of program comprehension such as impact analysis, exploring the program structures and its representation, program design as well as identifying the location of source codes concerned. Although there are lots of studies done to improve program comprehension, some programmers neglected them due to complex ways in the proposed ideas as they failed to expose the programmers to the real world environment (Lahtinen *et al*., 2007; Maletic & Kagdi, 2008; Schankin *et al*., 2018; Vainio & Sajaniemi, 2007).

Source code is an essential artefact for software maintainers to become familiar with a software system (Carvalho, 2013; Corley *et al*., 2012; Cornelissen *et al*., 2011; Sharafi, 2011; Tiarks *et al*., 2013; Yazdanshenas & Moonen, 2012). Nowadays, the expansion in size and difficulty of software system leads to difficulties in maintaining the system. Thereby, software maintainers have to keep up a huge size of source code that needs to be comprehend (Haiduc *et al*., 2010; Ishio *et al*., 2012) and to identify a corresponding piece of code that needs to be maintained (Carvalho, 2013; Roehm *et al*., 2012; Ying & Robillard, 2011). Among their tasks is to identify the source code location and understand it before implementing maintenance tasks (Carvalho, 2013; Roehm *et al*., 2012; Ying & Robillard, 2011).

Due to the increase in the size and complexity of software system as well as lack of proper documentation and knowledge expert, maintenance tasks can be very time-consuming, thus requiring more effort to maintain a system (Guzzi *et al*., 2011; Normantas & Vasilecas, 2013; Roongruangsuwan & Daengdej, 2010). In the maintenance phase requires a very high effort to get to know and understand the source code (Bouwers *et al*., 2010; Meng *et al*., 2006.). Supported by Norman and Vasilecas (2013) and Wang (2017), it has been stated that software engineers have to spend 41.8% of total effort in reading and finding a relevant source code program to make changes.

Besides, problems in providing the related documentation on a system as well as the lack of experts contribute to the major problems in carrying out maintenance tasks. Expert knowledge is needed especially for novices to help them in performing maintenance tasks. Problems usually exist when the developers who developed the software system abandoned their project (Xu, 2005). The absence of the original

programmer will subsequently affect the understanding of a system and leave a negative impact in performing the maintenance tasks. Furthermore, existing information is still insufficient and needs to be enriched. One way to do this is by refining the information into the integration of information, which combines different information sources (Poshyvanyk *et al*., 2012).

The study on program comprehension remains incomplete and should be continued to produce the best techniques to improve program comprehension (Corritore & Wiedenbeck, 2001; Maletic & Kagdi, 2008; Xu, 2005). This problem is yet to be discussed in detail to support software maintainers in performing maintenance tasks.

1.1.2 Identification of Source Code Location for Program Comprehension

The process of identifying the location of source code concern is called concept location. It is primarily a human activity defined as "the activity of identifying the initial location in the source code that implements the functionality in a software system" (Dit *et al*., 2013). Several techniques have been introduced to automate some or all of the process of concept location. Those techniques rely heavily on code comprehension as it is considered prerequisite when maintaining any software system.

One of the most commonly used techniques to support concept location is based on text search in source code where the software maintainers write a query and the search engine returns a list of source code elements relevant to the query (Poshyvanyk *et al*., 2012). Considering all the tasks in concept location, many software maintainers can benefit from this technique to reduce their efforts and save their time to comprehend unfamiliar system and to improve performance of maintenance tasks.

1.2 Research Problem

It is known that one of the important elements that effect the performance of the text retrieval technique is the information contained in a data source called corpus. Most of the existing works on text retrieval technique (Akbar & Kak, 2019; Bohnet *et al.*, 2008; Hill *et al*., 2009; Karnalim, 2018; Lapeña *et al*., 2016; Rahman & Roy, 2017; Rahman *et al*., 2017; Sachdev *et al*., 2018; Shepherd *et al*., 2006; Swathi & Anju, 2019; Vinayakarao *et al*., 2017) did not consider the importance of knowledge information which they did not provide information about the relationships between concepts within a domain. Thus, the existing works unable to describe the whole structure of source code and lack of informtaion to represent significant domain knowledge information of system software.

Since ontology-based approach provides a foundational perspective for knowledge representation, this approach should be considered as a contribution to text retrieval technique. The purpose of using ontology is to extract the terms and concepts in certain domain in the source code as well as to find the correct relationships between different concepts (Anikin & Sychev, 2019). Thus, representing source code in a form of ontology is able to describe the whole structure of source code at any level of granularity and can present more information in developing corpus as a data source for source code retrieval.

Apart from that, the lack of integration of information in corpus also invites problems in source code retrieval technique. Previous works (see Hu *et al*., 2018; Schankin *et al.*, 2018; Marcus *et al*., 2004; Poshyvanyk *et al*., 2006; Poshyvanyk & Marcus, 2007; Gay *et al.*, 2009; Bohnet *et al.*, 2008; Hill, 2009) had only use lexical information in constructing the corpus. Lexical information only provided the

information of source code which are extracted from identifiers and comments (Hu *et al*., 2018; Schankin *et al*., 2018). This makes the content in the corpus to be limited to represent the entire information in the source code. By integrating lexical information and knowledge information produced by ontology, it will enrich the information contained in the corpus (Song *et al*., 2019; Yang *et al*., 2019). This will also contribute to enhance the performance of source code retrieval technique.

Therefore, this study explores the important of new information that able to improve the source code retrieval technique. It proposed the use of ontological as knowledge representation by extracting information from source code. Ontology has been used in many fields to represent knowledge about certain concepts and can be applied to improve maintenance tasks (Karnalim, 2018; Swathi & Anju, 2019; Wilson, 2010). Furthermore, this study integrated the knowledge and lexical information as an addition of relevant information. This approach has a great potential to reduce the effort and increase the performance of software maintainers to perform maintenance tasks.

1.3 Research Questions

The research questions are as follow:

i. How to propose source code representation feasible to represent significant domain knowledge information of system software compared to original source code? (RQ-1).

ii. How to enrich the contents data source that used in source code retrieval process by leveraging the knowledge information of source code? (RQ-2).

iii. How to construct a model to improve the source code retrieval technique while performing concept location? (RQ-3).

1.4 Research Objectives

The aims of this study is to improve the source code retrieval technique in concept location approach to facilitate software maintainers in finding the location of source code prior to the software maintenance phase. To improve the effectiveness of identifying out the source code location relevant to the topic of interest means to return many relevant documents and few non-relevant documents to support user exploration of the program and task completion.

The following objectives were aimed to achieve the above goal.

i. To construct a new ontology as a source code representation to represent domain knowledge information of system software (RO-1).

ii. To propose a new corpus development by integrating and enriching the information in a new corpus to evaluate the performance of the proposed model (RO-2).

iii. To produce a model based on the use of ontology representation as an enhancement of source code retrieval technique (RO-3).

The first objective *(RO-1)* was derived from the first research question *(RQ-1)* listed in Subchapter 1.3 where the aim is to explore and find the acceptable representation to represent knowledge information for source code by conceptualising a software system on an ontological perspective. Meanwhile, the second objective *(RO-2)* was defined to answer the second research question *(RQ-2)*, which involves the integration and enrichment of the data source that use in retrival process by utilising the lexical and knowledge information. The third objective (*RO-3)* derived from research question *(RQ-3)* involves constructing a model for source code retrieval technique to improve concept location.

1.5 Expected Contribution

This study is expected to contribute to:

i. **Introducing a new source code representation** – Explore and find acceptable representation to represent domain knowledge information for program source code.

ii. **Developing a new corpus** – The corpus as a data source to improve text retrieval technique by integrating and enriching the lexical and knowledge information.

iii. **Introducing a new Ontology-Based Source Code Retrieval (OntoBased-SR) technique** - The technique is to improve the accuracy of retrieve and rank the relevant information based on query submitted.

1.6 Scope and Limitation

This study focuses on source code retrieval approach for source code concept location technique in the context of software maintenance, which occurs in the presence of source code modification request. Although the underlying challenges associated with the concept location are indexing process, data sources, query formulation and similarity measure, this study focused on the importance of information represented as a source for information retrieval. It deals with a source code as a software artefact focusing on text documentation

For indexing process and similarity measure, this study deployed the existing information retrieval model, which is Latent Semantic Indexing (LSI) model. For similarity measure, the cosine similarity was used to measure relevant documents to the input query with the result displayed in the ranked list of documents. This study selected Object-Oriented Programming (OOP) source code and JAVA language as a

source of input due to their characteristics, which are simple, modular, modifiable, extensible, maintainable and re-usable.

This study only extracted the method-level granularity from the source code as the target source. This preference can be explained by several factors. First, methods locate the concepts and features in more detail than classes and files. Second, most of the existing technique consider the structural and dynamic information in the text retrieval techniques, which often use information at the level of granularity of methods. Third, as more and more approaches made use of the method-level granularity, researchers had little choice in the cases when a comparison to previous approaches is desired.

This study make used the roles of ontology in terms of source representation in representing the knowledge information and the information often used in source code concept location technique, which is lexical information. Taking into account the importance of the knowledge information and lexical information, both were integrated in this study with hope that this approach can enhance the ability of the retrieval process. The experiment was conducted to test the performance of the proposed work by applying it to the real open source system.

1.7 Thesis Organisation

The organisation of the chapters in this thesis is as follows:

In **Chapter 2** discusses the literature review related to this study. It begins with the tasks and activities of software maintenance and their roles in software engineering. It continue to explores previous studies regarding program comprehension that considers different works, ideas and opinions of the researchers.

Furthermore, the following section presents the knowledge-based management that discusses the roles of ontology in knowledge representation as well as the importance of ontology in software engineering. After that, the concept location technique suitable in program comprehension activity is demonstrated followed by the section that briefly explains the text retrieval model for concept location. The next section is about the types of source code representation used in concept location. The following section describes the cosine similarity measurement used in this study. At the end of the chapter is the chapter summary.

In **Chapter 3,** research methodology that discusses about the procedures taken since the beginning until the end of conducting study is demonstrated. The first section describes the research methodology design followed by the proposed work model. The next subsection presents the process of conducting the evaluation on the proposed work. This is followed by the discussion on the pre-processing of the dataset and the information on the gold standard as well as the preparation for the experiment. The following sections explain the preparation of benchmark; the experiment conducted well as the statistical evaluation. Finally, summary is presented in the last section.

Chapter 4 covers the implementation of the first phase of the proposed work that briefly discusses the construction of ontology, corpus creation and the integration and enrichment of the new corpus. The second phase of the proposed work is discussed in **Chapter 5.** It provides a step-by-step explanation on the procedures conducted for source retrieval process in concept location. The last section describes the summary of the chapter.

The evaluation of the proposed work is explained in **Chapter 6**, which is presented in detail regarding the procedures conducted in the experiment, the analysis

and results of the evaluation. The threats to validity of experiments are also presented in this chapter. The discussion on the findings is presented in the next section. The final section states the chapter summary.

Chapter 7 is the last chapter comprising the conclusion of the study by drawing the summary of the thesis. Besides, it includes the contributions of the study determined based on the findings. The chapter also provides suggestions of possible improvements for further work, which are stated at the end of the chapter. At the end of the chapter is the conclusion of the chapter.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Nowadays, comprehending a software system has become a challenge because of the size and complexity of the program. Program comprehension is very important and must be in place prior applying maintenance task. Over the past decades, many studies in program comprehension have been conducted suggesting many ideas, techniques and tools that can help software developers to comprehend a program. Nevertheless, the study of program comprehension remains a challenge.

Recognising the location of source code is one of the activities while implementing maintenance task and it is the fundamental in program comprehension process. Software developers need to match their understanding on the program domain to its representation in the source code. Moreover, among the things to be concern during the process of identifying the relevant source code are; the characteristics of the source code structure and the nature of the problem domain such as internal comments, external documentations, variable names and annotations. This constitutes the problems in program comprehension.

This chapter discusses the issues of concept location related to program comprehension during software maintenance. It is divided into five main discussions: the overview of software maintenance; the importance of program comprehension; the role of concept location; the application of source code retrieval technique in concept

location; and the role of ontology to provide the information while comprehending a program. Then, summary is made at the end of the chapter.

2.2 Software Maintenance

Software maintenance is defines as a process of modification a software system to correct faults and to improve performance (IEC/ISO, 2008; Standard, 2008). Software maintenance tasks involve four categories: corrective, preventive, adaptive, and perfective. These tasks are then grouped into two major activities: correction and enhancement (as in Table 2.1).

Table 2.1 Software Maintenance Tasks and Its Activities

Software Maintenance Tasks	**Tasks Description**	**Activities**
Corrective	To repair and fixing existing faults that cause the system to fail.	Correction - Understand system, evaluate hypotheses concerning problem, repair code, regression tests.
Preventive	Preventing failures by detecting and fixing before failure.	
Adaptive	Making change in existing software to accommodate a changing environment or to fulfill the customer-based requirements.	Enhancement - Understand system, define the requirement for improvement/adaptation, and develop preliminary and detailed adaptation/perfective design, code changes, debug, and regression tests.
Perfective	Making improvement to the existing systems and to increase the users' satisfaction without affecting end-user functionality.	

The key challenge faced by developers is to comprehend the software system being maintained. Program comprehension is time-consuming because this activity is an integral activity in each software maintenance task (as shows in Table 2.1). The next section discusses in detail on several issues related to program comprehension.

2.3 Program Comprehension

Software system needs to be maintained after its operation. The process of changes and modifications applied in maintenance task is to improve the quality, speed and accuracy of the system. Program comprehension is important for a successful evolution of software as the tasks in software maintenance are required to understand the software to be maintained (Fakhoury *et al*., 2018; Maletic & Kagdi, 2008; Salvaneschi *et al*., 2017; Sasirekha & Hemalatha, 2011; Soh, 2011). Before the software system can be correctly maintained, the specific location of existing code that will be modified has to be identified to make changes (Alhindawi *et al*., 2014). Software maintainers must first find the relevant parts of the code corresponding to a particular change and this task is related to concept location. Therefore, this section explores the importance of program comprehension in software maintenance and further discusses the roles of concept location to improve program comprehension.

Program comprehension is "the process of taking source code and understanding it" (Deimel & Naveda, 1990) or the process of using the existing knowledge to acquire new knowledge (Aljunid, Zin, & Shukur, 2012; Xia *et al*., 2017). The understanding on program is related to execution behaviour and relationship of variables involved in the program (Hu *et al.*, 2018; Sasirekha & Hemalatha, 2011). The study of program comprehension can be explained as the process occurs in the software engineers' mind when they understand a program (Feigenspan & Siegmund, 2012). This section discusses the issues arising in program comprehension followed by the existing approaches and models on program comprehension.

2.3.1 Issues on Program Comprehension

Source code is a more trusted source of data compared to composed documentation primarily since documentation is regularly non-existed or obsolate (Maalej *et al*., 2014). However, the problems still exist if the source code is used as reference to a system. The activity in reviewing and understanding a source code is not the same as reviewing ordinary documents and many problems in program comprehension arise due to the use of textual representation as the primary source of information. In fact, programs are often in the form of a hierarchical structure, but the actual behaviour of a program cannot be reflected as it is represented in textual forms. Although many methods and tools have been proposed to represent source code, experience have shown that textual presentation is the most suitable to represent the software system (Krinke, 2004).

Current software systems are difficult to be comprehended because of the size and complexity of the program, thereby leads to difficulties in understanding and maintaining the system. This is due to the source code itself. Most of the problems due to tricky codes, different programming styles, poor naming conversion, program representation, insufficient comments, architectures, components, design and identifier style. Many studies have proposed the factors to improve source code comprehension such as cross referencing, developers' program domain knowledge, syntax highlighting and tools, comments, dependence graph, slicing, ripple analysis and program decomposition.

Another issue is on the human activities. Many software developers work in team on large software projects. The implication of this is that they need to be able to quickly understand each other's code writing styles to be more productive. Software

maintainers need to be able to understand the developers' code. Software maintenance is important as it may account for 65% to 75% or as much as 80% the total lifetime of a software (Greevy & Zaidman, 2005; Normantas & Vasilecas, 2013; Roongruangsuwan & Daengdej, 2010; Schankin *et al*., 2018; Siegmund, 2016; Xu, 2005). In addition, sometimes the source code documentation was never written, out-of-date or lost. This suggests that software maintainers need to give more efforts in the maintenance phase.

Despite many studies carried out in finding the different strategies and techniques to overcome program comprehension problems, most researchers still have yet to discuss on how to help the software maintainers to comprehend a program.

2.3.2 Elements of Program Comprehension Process

Frequently, most of the studies on program comprehension consider three basic elements that complement to the comprehension process. These elements are knowledge based that appear in a programmer’s mind; external representation and; the assimilation process (in Figure 2.1). The process flow explains how developers understand a program using their existing knowledge through the assimilation process supported by external representation to obtain new knowledge.

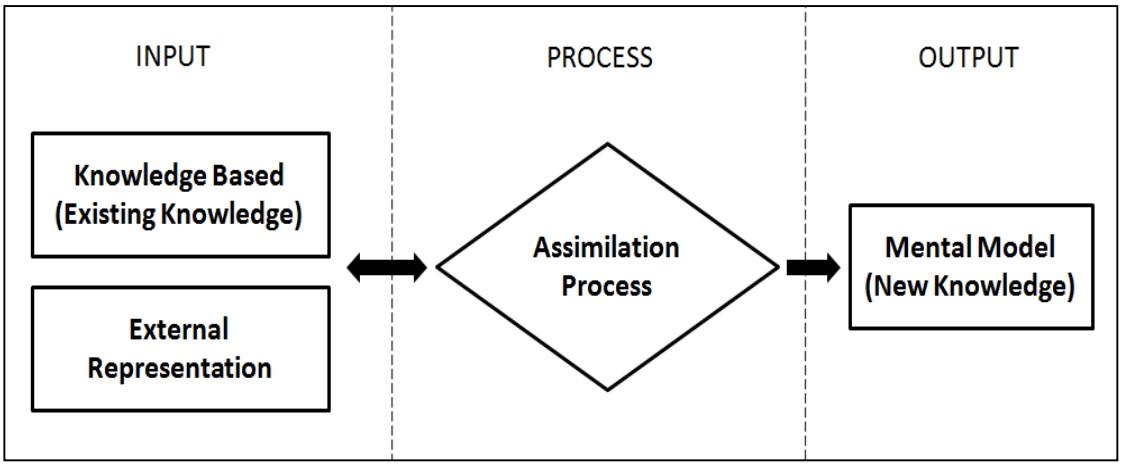

Figure 2.1 The Elements in Program Comprehension Process

Knowledge-based

Knowledge based is an experience or existing knowledge on a program contained by a programmer. It can determine the programmers' ability to comprehend a program. Table 2.2 shows the types of knowledge and researchers discovering the ideas.

Table 2.2 Types of Knowledge

Authors	Types of knowledge
Brooks, 1983; Carvalho, 2013; Rugaber, 2000; von Mayrhauser & Vans, 1997	Domain Knowledge
Soloway & Ehrlich, 1984; Wiedenbeck, 1986	Plans and Rules of discourse
Brooks, 1983; Rist, 1986; Weiss & Mockus, 2013	Beacons and Chunks
Gellenbeck & Cook, 1991; Shneiderman & Mayer, 1979; Soloway & Ehrlich, 1984	Syntactic and Semantic
Busjahn *et al*., 2014; Détienne, 2002; El-sheikh *et al*., 2013; Letovsky, 1987; Piaget, 2013)	Schemas and Abstraction

Domain knowledge consists of three domains, which are task/problem domains, intermediate domain and program domain (Brooks, 1983). During the comprehension process, the task domain is mapped to the intermediate domain and produces the program domain. Moreover, hypotheses can be constructed using domain knowledge by predicting the program with reference to the existing knowledge. Another types of knowledge is the plan and rules of discourse, which is used for developing and validating expectations, interpretations and inferences, includes causal knowledge on information flow and the relationships among parts of a program (Soloway & Ehrlich, 1984; Wiedenbeck, 1986).

Beacons are the familiar feature in the source code serving as a cue indexed into existing knowledge to present certain structure of plans (Brooks, 1983; Rist, 1986). On top of that, beacons are utilised to predict hypotheses. Another type of knowledge is schema. According to Piaget's theory, schemas are the way of organising knowledge to become as a unit. Each knowledge is related to aspects including the object, action and abstract concepts (Piaget, 2013).

External representations

External representations are any materials available as an aid to support programmers while comprehending a program. The materials can be represented in different ways and formats. The external support may be in a form of system documentation, source code, manual, book or expert advises as well as techniques and tools.

Assimilation

Assimilation is a process comprehending a program and considering incorporated and constructed with existing knowledge. In particular sign, the characteristics of programmers while comprehending a program is important since they use all their senses and capabilities to understand a program.

2.3.3 Program Comprehension Approach in Cognitive Perspective

Cognitive model is used to represent the processes involved in developing and building the programmers' mental model or acquire new knowledge from existing knowledge (Storey, 2006). Developers use their existing knowledge such as programming expertise, programming language, computing environments, programming principles, architectural model, algorithm and solution approve as well as domain-specific information or problem-domain, which will after that go through the assimilation process supported by external representation. This process is continued to obtain new knowledge like functionality, architecture, algorithm implementation, control flow and data flow.

Previous studies on cognitive model provide explanations on the short-, long-,and working-memories used (Brooks, 1983; Pennington, 1987; Soloway & Ehrlich, 1984; von Mayrhauser & Vans, 1997). Other authors theorised that cognitive internal representation of knowledge is produced through the concept of frames, plans, and chunks (Minsky,1974; Rich & Waters, 1990; Soloway, 1984). Gagne (1985) also proposed cognitive strategies and believed that environment can influence the comprehension process. He stated that to stabilise the cognitive strategies, people must have certain techniques of thinking, ways of analysing problems and having approaches in solving a problem. People use cognitive strategies in thinking about the

things they learnt and in solving problems. These are the ways in managing the processes of learning, remembering and thinking. Bloom (1956) discovered the ideas of learning domain called Bloom's Taxonomy and adjusted by Anderson *et al.* (2001). The taxonomy focuses on three domains with one of them devoted on cognitive domain that emphasises things that learners to know during learning. It involves knowledge and the ability to develop intellectual skills.

Bandura (1994) argued that people can gain new knowledge through viewing or observing. He stated the steps involved in learning process, which are attention, retention, reproduction and motivation. The first learning step proposed is to pay attention to new things. Learners have to pay full attention to grasp a new knowledge. Then, they must have the ability to store the information (retention) they obtained. This internal mental state is important as an essential part in the learning process. The next step is the reproduction where learners are able to use the knowledge they grasp and to be successful in their learning, they have to be motivated to apply the new knowledge modelled.

The next is the discussion on the three predominant approaches of program comprehensions, which are top-down, bottom-up and integrated meta-model. These models are the foundation in creating the new model of program comprehension (Meng *et al.*, 2006; O'brien, 2003).

2.3.3(a) Top-down

Typically, top-down approach is adopted when the developers become familiar with the source code (Soloway & Ehrlich, 1984). This approach is goal-oriented and hypothesis-driven contains a hierarchy of goals and plans. It is the dynamic process strategy of reconstructing knowledge to formulate hypotheses regarding the domain of the program and mapping this knowledge to the source code and use the strategic plan to implementation plan (Brooks, 1983; Storey, 2005; Von Mayrhauser & Vans, 1995). However, the limitation of this approach is that it does not consider novices' capabilities as they are inexperienced in the domain and lack of knowledge to formulate hypotheses in the first place

2.3.3(b) Bottom-up

Bottom-up approach is introduced by Letovsky (1987) focusing on novice developers since it does not require higher level knowledge structures such as design or application-domain knowledge. The developer firstly read the code statements and then mentally chunk or group these lines of code into higher-level abstractions to form the abstract concepts supported by beacons (Letovsky, 1987; Von Mayrhauser & Vans, 1995). This approach is suitable for developers who are unfamiliar with the source code.

2.3.3(c) Integrated Meta-model

Von Mayrhauser & Vans (1995) introduced the integrated meta-model by integrating the top-down and bottom-up approaches. The proposed approach is based on the observations. They found that neither top-down nor bottom-up is the best approach in the assimilation process (von Mayrhauser & Vans, 1993). Supported by

Storey (2005), the paper mentioned that developers can choose to invoke top-down or bottom-up model as a starting point for formulating hypotheses when the code is familiar.

2.3.4 Program Comprehension Models

This section discusses the existing models supporting program comprehension. The discussion is based on selected papers focusing on the strategies, approaches and the process taken to assimilate the existing knowledge to yield new knowledge. Figure 2.2 shows the evolution of program comprehension discussed by Schulte *et al.* (2010). In previous studies, most of the researchers used the cognitive model as a strategy to propose a program comprehension model. They believe that this is the way of managing the processes of learning, remembering and thinking.

The first model proposed by Shneiderman & Mayer (1979) uses the bottom-up approach focusing on novice users. The model involves the short-term memory and long-term memory as well as the internal semantic knowledge to develop mental model. It involves a process of chunking in which users are mentally making a chunk out of a program guided by the beacons. Pennington (1987) proposed a model with bottom-up approach guided by beacons, plans and text structure to perform chunking process. The work integrated the domain and program model to depict situation model. Burkhardt *et al.* (2002) in their study use bottom-up approach to comprehend Object-Oriented Program compared to Shneiderman & Mayer (1979); and Pennington (1987) that focuses on structured program.

Instead of cognitive models, the combination of other models were applied in the studies such as text comprehension model (Pennington, 1987), constructivist model (Exton, 2002; Václav *et al.*, 2002), vision model (Ali *et al.*, 2011) and problem solving model (Douce, 2008). Learning Model proposed by Rajlich & Wilde (2002) interprets programmes based on constructivist theory where the developer divides program comprehension process into assimilation and adaptation. From his perspective, assimilation is the process of adding new facts to mental model, otherwise adaptation is the process of organising the existing knowledge to absorb new knowledge. Xu (2005) extends the Learning Model, namely Multi-Dimensional Model integrating the Bloom's Taxonomy, cognitive model and learning model. This study looks at the activities of assimilation and accommodation in the learning process. Although this study focuses on experts to make a hypothesis, it is also suitable for novices as it combines top-down and the bottom-up approaches.

Meng *et al*. (2006) introduced the Comprehension Process Model that utilises ontology and the description logic to constitute the content of mental model. The ontology based on story-drive is used to model the sources of information that describes the behaviour of a program. Store Model proposed by Douce (2008) is the heuristic model combining the elements in the working memory model and other knowledge such as strategic, semantic and plan. Frey *et al*. (2011) worked on categorisation and separation of concern to build a mental model. Their study took an element in programmers' knowledge to understand the program of concern. The process makes use of prediction or hypothesis using prior knowledge and verification on the prediction will update the knowledge about the concern. The classification of the models was made as shown in Table 2.3.

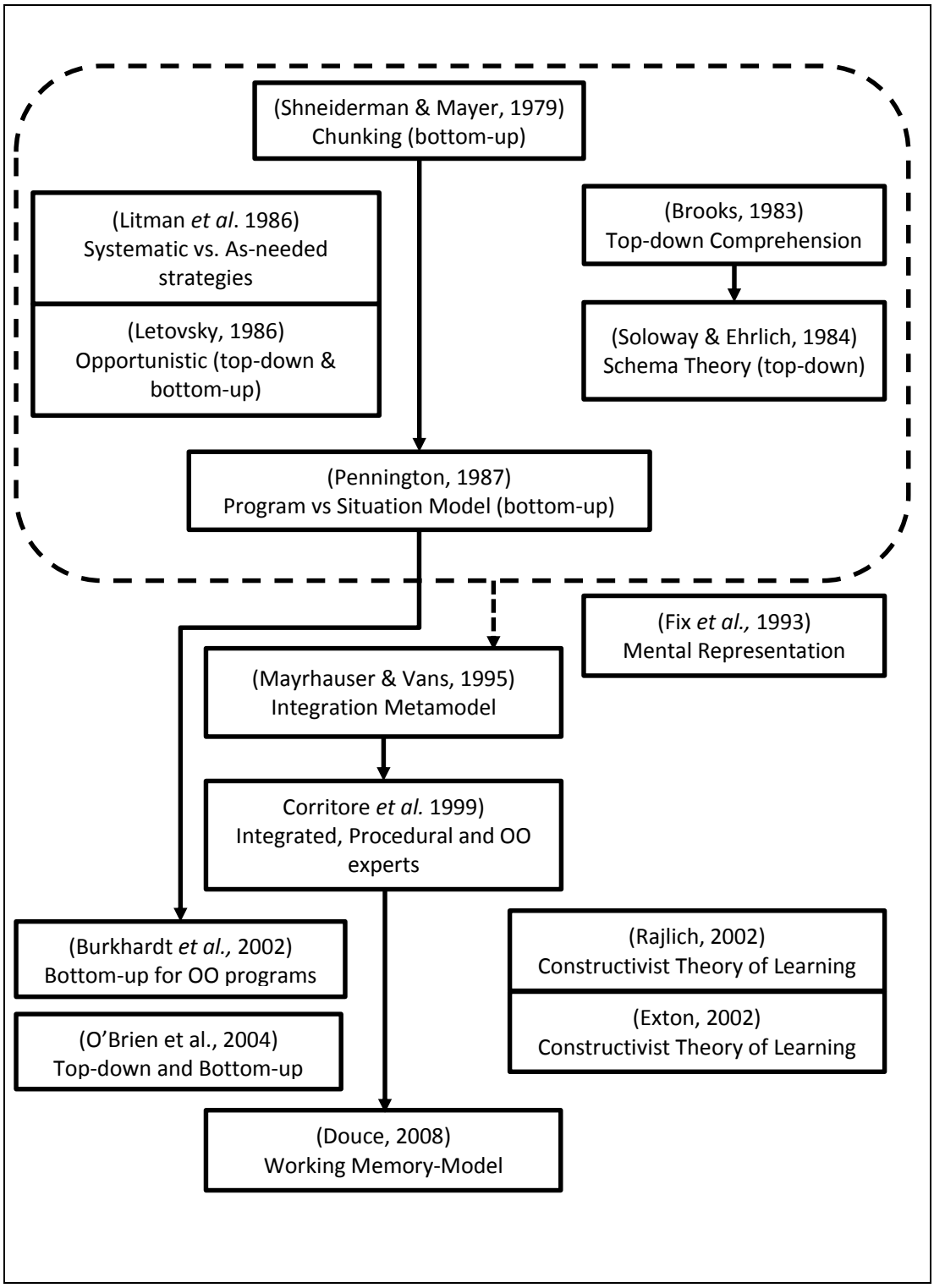

Figure 2.2 Evolution of Program Comprehension Model (Schulte *et al.*, 2010)