A NOUN-BASED FEATURE LOCATION APPROACH SUPPORTED BY TIME-AWARE TERM-WEIGHTING TECHNIQUE FOR FACILITATING SOFTWARE MAINTENANCE

SIMA ZAMANI

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2016

A NOUN-BASED FEATURE LOCATION APPROACH SUPPORTED BY TIME-AWARE TERM-WEIGHTING TECHNIQUE FOR FACILITATING SOFTWARE MAINTENANCE

SIMA ZAMANI

THESIS SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2016

UNIVERSITI MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: **Sima Zamani**

Registration/Matrix No.:**WHA100046**

Name of Degree: **DOCTOR OF PHILOSOPHY**

Title of Project Paper/Research Report/Dissertation/Thesis ("this Work"): **A Noun-Based Feature Location Approach Supported by Time-Aware Term-Weighting Technique for Facilitating Software Maintenance**

Field of Study: **Software Engineering (Computer Science)**

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature   Date

Subscribed and solemnly declared before,

Witness's Signature   Date

Name:
Designation:

ABSTRACT

Feature location is one of the frequent software maintenance activities that aims to identify a source code location pertinent to a software feature. Most of the proposed feature location approaches are based, at least in part, on text analysis to determine the similarity of a new feature with the source code data. However, the text analysis methods used in feature location originate from the natural language context. Unlike the typical context in which these methods are applied, text documents in software repositories, such as source code files, have a corresponding set of metadata including such items as time-stamps, developer identifiers, and commit comments. Furthermore, the history of changes of the source code is recorded in the repositories that leads to a larger dataset size. Due to these differences between the contexts in software repositories and natural language, the text analysis does not utilize its possible potential for accurately locating software features. Accordingly, the goal of this thesis is to improve feature location by addressing the specific characteristics of the repositories' text data, i.e. incorporation of the data with metadata and larger dataset size, within the text analysis process.

In this thesis, a new feature location approach is proposed that considers the metadata of time and developer, and uses only the nouns. The proposed approach analyzes and weights the data from the aspect of time when the data was recorded and the aspect of developer who recorded the data in the repository. In this approach, first, a time- and developer-based corpus is created from the nouns extracted from the repository's data. Then, the nouns are weighted using two term-weighting techniques including a time-aware term-weighting technique and a developers-based time-aware term-weighting technique. Next, the calculated weights for each noun are combined to obtain the total noun's weight. Finally, the source code files were ranked based on the summation of the total weights of the nouns that appeared in both the given software feature and the source code files.

The empirical evaluation of the proposed approach on a set of open-source projects indicates remarkable improvements over the feature location baseline approaches that utilize VSM (Vector Space Model) and SUM (Smoothed Unigram Model). The proposed approach outperforms the accuracy, effectiveness and performance of the feature location baseline approaches as much as 62%, 43% and 30%, respectively. In this approach, the time-based analysis and weighting of the data make an improvement over the baseline approaches up to 38%, 35% and 19%, respectively; whereas the developer-based analysis and weighting of the data make an improvement up to 55%, 39% and 29%, respectively. Furthermore, the use of nouns-only, instead of using all types of terms, improves the accuracy, effectiveness and performance as much as 26%, 49% and 23%, respectively and reduces the dataset size up to 60%. The statistical analysis of the experimental results demonstrates the significance of the improvement in all aspects. In general, consideration of time-metadata and developer-metadata in analyzing and weighting the data, along with the use of only the nouns, makes significant improvements to feature location.

ABSTRAK

Ciri lokasi merupakan salah satu daripada aktiviti yang kerap dilakukan dalam penyelenggaraan perisian dengan tujuan untuk mengenal pasti lokasi permulaan dalam kod sumber yang berkaitan dengan sesuatu fungsi perisian. Kebanyakan daripada pendekatan ciri lokasi yang dicadangkan adalah berdasarkan sekurang-kurangnya sebahagian kepada analisis teks bagi mengenalpasti persamaan fungsi yang baharu dengan data kod sumber. Walaubagaimanapun, kaedah analisis teks yang diguna pakai dalam ciri lokasi berasal daripada konteks bahasa semula jadi. Berbeza daripada konteks umum di mana kaedah-kaedah tersebut diaplikasikan, dokumen teks dalam repositori perisian seperti fail-fail kod sumber, mengandungi satu set metadata yang sepadan termasuk item-item seperti setem masa, identiti pembangun dan komen komit. Tambahan pula, sejarah perubahan kod sumber juga direkodkan di dalam repositori di mana ia menyebabkan saiz set data menjadi lebih besar. Disebabkan oleh perbezaan tersebut antara konteks dalam repositori perisian dan bahasa semula jadi, analisis teks tidak dapat menggunakan potensinya untuk mengesan fungsi-fungsi perisian dengan tepat. Dengan wajarnya, matlamat utama thesis ini adalah untuk menambah baik ciri lokasi dengan menangani sifat khusus data teks repositori, iaitu penglibatan data dengan metadata beserta saiz set data yang lebih besar dalam proces analisis teks.

Dalam thesis ini, satu pendekatan ciri lokasi yang baru dicadangkan di mana ia mengambil kira metadata masa dan pembangun, dan juga menggunakan hanya kata nama. Pendekatan yang dicadangkan menganalisis dan menghitung data dari aspek masa data direkodkan dan juga aspek pembangun yang direkodkan di dalam repositori. Dalam pendekatan ini, satu korpus berdasarkan masa dan pembangun dicipta dari kata nama yang diekstrakkan daripada data repositori. Kemudian, kata nama tersebut dinilai dengan menggunakan dua teknik hitung-istilah (term-weighting technique) yang terdiri daripada satu teknik sedar-masa hitung-istilah (time-aware term-weighting technique) dan satu teknik sedar-masa hitung-istilah berdasarkan pembangun (developer-based time-aware term-weighting technique). Seterusnya, keberatan yang dikirakan bagi setiap kata nama digabungkan bagi mendapatkan jumlah keberatan bagi kata-kata nama tersebut. Akhir sekali, fail-fail kod sumber tersebut diatur mengikut kedudukan berdasarkan kepada jumlah keseluruhan daripada jumlah keberatan kata-kata nama tersebut yang muncul dalam sesuatu fungsi perisian dan fail-fail kod sumber.

Penilaian empirikal untuk pendekatan dicadangkan ke atas satu set projek sumber-terbuka menunjukkan peningkatan yang luar biasa ke atas pendekatan-pendekatan ciri lokasi asas yang menggunakan VSM (Vector Space Model) dan SUM (Smoothed Unigram Model). Pendekatan dicadangkan ini memberikan keputusan yang lebih baik dari segi ketepatan, berkesanan, dan prestasi dalam pendekatan-pendekatan ciri lokasi asas, sebanyak 62%, 43%, dan 30% masing-masing. Dengan pendekatan ini, analisis berdasarkan masa dan penghitungan data memberi peningkatan sehingga 38%, 35%, dan 19% masing-masing berbanding dengan pendekatan-pendekatan asas; manakala analisis berdasarkan pembangun dan penghitungan data memberi peningkatan sehingga 55%, 39%, dan 29% masing-masing. Tambahan pula, penggunaan hanya kata nama, selain daripada penggunaan semua jenis istilah, meningkatkan ketepatan, keberkesanan dan prestasi sebanyak 26%, 49%, dan 23% masing-masing. Pendekatan yang dicadangkan juga berjaya mengurankan saiz dataset sehingga 60%. Analisis statistik daripada keputusan eksperimen menunjukkan penambahbaikan dalam semua aspek. Secara umum, penimbangan metadata masa dan metadata pembangun dalam menganalisis dan menghitung data, bersamaan dengan penggunaan hanya kata nama memberi peningkatan yang ketara ke atas ciri lokasi.

ACKNOWLEDGEMENTS

I would like to thank all those people who have helped me to finish this thesis and supported me throughout my whole study.

First, I am greatly indebted to my supervisor Prof. Lee Sai Peck for her support and guidance throughout the years. She has taught me many valuable lessons about being a good researcher. I sincerely appreciate her advice concerning my presentations and writings.

A special thanks to my husband, Ramin Shokripour, for his endless help, care, kindness and patience. I would like to thank my family. I am forever indebted to my parents and my parents in law for their unconditional love and support.

I really appreciate the time and effort that Dr. John Anvik put into reading my papers and his valuable comments. Also, I am thankful to Dr. Hadi Saboohi for proofreading this thesis and helping me to improve the quality of my writing.

I would like to thank many great people working with me in the same research lab for their friendship and their willingness to listen and help me on presenting my ideas. In particular, I would like to thank Ehab Nabiel Mohammed, Amineh Amini, and Saifur Rehman Khan.

The work conducted in this thesis uses several open-source projects to provide the required dataset for experimental evaluation of the proposed methods and approach. I am grateful for the contributions of developers of these open-source projects for helping me to collect their repositories data. The proposed methods and approach in this thesis were developed in TraceLab framework. I am thankful to the developers of this project in helping us to implement our components in TraceLab.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

TiNoFeLo	Time-aspect analysis of data in a Noun-based Feature Location
DeNoFeLo	Developer-aspect analysis of data in a Noun-based Feature Location
TiDeNoFeLo	Combination of Time-aspect and Developer-aspect analysis of data in a Noun-based Feature Location approach
TATW	Time-Aware Term-Weighting technique
TADTW	Time-Aware Developers' expertise Term Weighting technique
TF	Term Frequency
TF-IDF	Term Frequency-Inverse Document Frequency
PM	Pattern Matching
IR	Information Retrieval
NLP	Natural Language Processing
VCS	Version Control System
ITS	Issue Tracking System
MSR	Mining Software Repositories
UM	Unigram Model
SUM	Smoothed Unigram Model
VSM	Vector Space Model
rVSM	revised Vector Space Model
LSI	Latent Semantic Indexing
LSA	Latent Semantic Analysis
LDA	Latent Dirichlet Allocation
SVD	Singular Value Decomposition
MRR	Mean Reciprocal Rank

MAP	Mean Average Precision
DFR	Divergence From Randomness
LM	Language Modeling
WWW	World Wide Web
ID	Identification Code
JDT	Eclipse Java development tools
AOIG	Action-Oriented Identifier Graph
GrepOF	grep with Ontology Fragments
ASDG	Abstract System Dependence Graph
Verb-DO	Verbs and their corresponding Direct Objects
FCA	Formal Concept Analysis
AST	Abstract Syntax Tree
BRCG	Branch Reserving Call Graph
SPR	Scenario-based Probabilistic Ranking technique
HITS	Hyperlinked-Induced Topic Search
PROMESIR	Probabilistic Ranking of Methods based on Execution Scenarios and Information Retrieval
LOBSTER	LOcating Bugs using Stack Traces and tExt Retrieval
TYRION	TraceabilitY link Recovery using Information retrieval and code OwNership
POS	Part-Of-Speech
JS	Jenson-Shannon method
NER	Named Entity Recognition
AOP	Aspect-Oriented Programming
IDE	Integrated development environment

ANNIE	A Nearly-New Information Extraction System
JAPE	Java Annotation Patterns Engine
LSD	Least Significant Difference
CVS	Concurrent Versions System
CVSANALY	Analyzing CVS Repositories
JELDoclet	Java Export Language doclet
NN	Noun, singular or mass
NNP	Proper noun, singular
GATE	General Architecture of Text Engineering

CHAPTER 1: INTRODUCTION

Software repositories are great sources of knowledge mainly related to the domain of a software project. The repositories, in which artifacts of software systems are stored, are produced from the early stage of software development (Kagdi, Collard, & Maletic, 2007; Witte, Li, Zhang, & Rilling, 2007; Rilling, Witte, Gasevic, & Pan, 2008; Witte, Li, Zhang, & Rilling, 2008). They are commonly used as record-keeping sources to store information and documents related to the whole lifecycle of a software system in order to help the developers and managers in developing, managing, maintaining and publishing the software system (Hassan, 2006). The software repositories involve the data such as the source code data recorded in version control systems, bugs or defects recorded in issue tracking systems, and the communication between project personnel recorded in communication archives (Hassan, 2006; Kagdi, Collard, & Maletic, 2007).

The data recorded in the repositories is typically changed and revisited multiple times during the software lifetime. The history of the changes as well as the metadata about the changes (e.g., developer who made the change, time when the change was done, and why the change was made) are recorded in the repositories (Kagdi, Collard, & Maletic, 2007). The action of analyzing the historical data recorded in the software repositories is known as Mining Software Repositories (MSR[1]) (Kagdi, Collard, & Maletic, 2007; Gethers, Kagdi, Dit, & Poshyvanyk, 2011). The aim of mining software repositories is discovering and extracting the interesting information about the software and its evolution (Hassan, 2008; Hassan & Xie, 2010) in order to enhance the software development activities (Hassan, 2006; Kagdi, Collard, & Maletic, 2007). Software maintenance is one of the software development phases which is basically enhanced by the MSR research works.

[1]The term MSR has been used to explain a broad class of investigations into the examination of repositories of software (e.g., Subversion and Bugzilla).

One of the frequent maintenance activities is feature (or concept) location which aims at identifying an initial location in the source code of a software system that needs to be modified in order to satisfy a particular software feature (Biggerstaff, Mitbander, & Webster, 1993; Rajlich & Wilde, 2002). A software feature expresses a functionality defined by software requirements (Dit, Revelle, Gethers, & Poshyvanyk, 2013) and it can be formulated as a change request for adding a new feature, removing a bug or defect, or improving an existing functionality (Yang & Pedersen, 1997; Bohner, 1996; Chen & Rajlich, 2000; Dit et al., 2013). A sample of a change request which was reported to the Eclipse JDT project is presented in Appendix A. More effective support for change requests is needed to obtain sustainable, high-quality evolution of software systems.

Typically, in order to satisfy a change request in the source code of a project, a set of source code files needs to be modified. Thus, one of the key issues in addressing a change request is finding relevant locations in the source code of the project requiring modification to address the change request (Sillito, Murphy, & De Volder, 2008; Dit et al., 2013). The possible starting point of the changes in the source code is identified by feature location[2] (Chen & Rajlich, 2010). Feature location is also a frequent software engineering activity that directly enhances software maintenance and software evolution tasks such as reverse engineering and incremental change (Marcus, Sergeyev, Rajlich, & Maletic, 2004). A sample of how feature location process is performed in a software project is explained in Appendix A. In a medium or large-scale software project with thousands of source code files, manually performing the feature location process is extremely challenging and time-consuming.

[2]The complete extension of the changes in the source code is accomplished using impact analysis. The methodology of performing impact analysis is different from feature location. In fact, feature location is known as the starting point of the impact analysis process.

1.1 Background

To facilitate the feature location process, many approaches have been proposed which are classified into three main categories based on their applied techniques. These categories involve dynamic analysis, static analysis, and text analysis (Dit et al., 2013). Dynamic analysis uses the information obtained from the execution of software to simulate the requested feature, and it is often used when features can be invoked and observed during run time (Eisenberg & De Volder, 2005; Liu, Marcus, Poshyvanyk, & Rajlich, 2007; Poshyvanyk, Gueheneuc, Marcus, Antoniol, & Rajlich, 2007; Dit et al., 2013). Unlike dynamic analysis, static analysis techniques does not require a working software system. Instead, static analysis deals with structural information such as control or data flow dependencies (Chen & Rajlich, 2000; Lukins, Kraft, & Etzkorn, 2010). Finally, text analysis investigates the text data recorded in the historical software repositories and analyzes them to extract useful information (Petrenko, Rajlich, & Vanciu, 2008; Cleary, Exton, Buckley, & English, 2009).

A recent survey on feature location literature (Dit et al., 2013) discovered that more than 51% of the published literature in feature location research area is based, at least in part, on text analysis. The use of text analysis for feature location is based on the assumption that identifiers, comments, and other text data found in software repositories contain domain knowledge that can be used for locating software features (Marcus et al., 2004; Dit et al., 2013). Text analysis process is taken into account as a fundamental step in most of the feature location approaches. The primary text analysis methods that were reported in the literature are Pattern Matching (PM) (Ratiu & Deissenboeck, 2007; Petrenko et al., 2008), Information Retrieval (IR) (Poshyvanyk et al., 2007; Dit et al., 2013) and Natural Language Processing (NLP) (Shepherd, Fry, Hill, Pollock, & Vijay-Shanker, 2007; Hill, Pollock, & Vijay-Shanker, 2009).

PM usually involves a text-based technique to search the source code using a utility such as regular expression matching tool, *grep* (Ratiu & Deissenboeck, 2007; Petrenko et al., 2008) being an example. This technique could be considered as the simplest text analysis technique that is used for feature location. The second class of text analysis methods consists of IR methods, which are based on statistical methods. IR methods typically analyze the documents of a corpus and retrieve similar documents to the context of a given change request (Poshyvanyk et al., 2007; Cleary et al., 2009; Lukins et al., 2010). IR considers the text resources as a collection of terms that co-occur frequently in the documents of the corpus (Manning, Raghavan, & Schutze, 2008). NLP, the last category of text analysis based feature location methods, considers the grammatical category of a term in the text to analyze and extract the required information from the documents (Shepherd et al., 2007; Hill et al., 2009). The NLP-based feature location methods apply NLP techniques to improve the accuracy of feature location process. NLP methods are query based and originate from a natural language context such as the newspaper articles (Bassett & Kraft, 2013).

As mentioned above, according to the research of Dit et al. (Dit et al., 2013), text analysis is the basis of most feature location approaches. Since text analysis process can be considered a fundamental step in most of the feature location approaches, the overall accuracy of feature location would be directly affected by the accuracy of the text analysis process. Investigation of the feature location literature indicates that the origination of the applied text analysis methods for feature location mostly comes from a natural language context (Bassett & Kraft, 2013). With respect to the origination of most of the text analysis methods that are applied for feature location, it is likely that text analysis may not utilize all possible potential for identifying the correct source code location pertinent to a change request. In order to address this issue, there is a need for investigating the characteristics of the text data recorded in the software repositories in comparing to the text data in the natural language context.

The comparison of the text data recorded in the software repository and the text data in a natural language context indicates the existence of some differences between these two types of text data (Bassett and Kraft 2013). As mentioned previously, unlike the typical context in the natural language, the text data in the repository is associated to a set of metadata, which is not found in simple text documents (Kagdi, Maletic, & Sharif, 2007; Ratanotayanon, Choi, & Sim, 2010). Incorporation of the data with metadata provides the ability of analyzing the data from different aspects such as the aspect of the developer who changed the data or the aspect of time at which the data was created or changed.

On the other hand, the text documents in the natural language context such as the newspaper articles are less structured than those found in the software repository such as the source code files (Bassett & Kraft, 2013). Furthermore, unlike the text data in the natural language context, the text data recorded in the repository are incrementally changed in the project lifetime and the history of the changes is recorded in the repository in order to keep track of the evolution and progress of the software development (Kagdi, Collard, & Maletic, 2007). Due to the storage of the history of changes, the size of data recorded for a text document is larger than that of the typical text documents in the natural language context. These characteristics of the repository's text data, i.e. incorporation of the data with metadata of time and developer, and larger dataset size, differentiate the text data recorded in the repository from the text data in the natural language context.

1.2 Problem Statement

The characteristics related to incorporation of the data with metadata and dataset size, which differentiate the text data in the repository from the text data in the natural language context, motivate the researcher to question whether the existing text analysis methods are efficient enough to be applied for feature location. According to these characteristics, there is a need for different techniques and methods to analyze the text data in the repository; or at least, the techniques and methods customary used for analyzing the text data need to be modified to consider the specific characteristics of the text data in the repository. Accordingly, the problem statement of this research is identified as follows:

> *The text analysis does not utilize its possible potential for accurately locating the software features* since the origination of the applied text analysis methods for feature location mainly comes from a natural language context.

Low accuracy of the text analysis is due to the lack of considering the specific characteristics of repository's text data, i.e. incorporation of the data with metadata and large dataset size, in analyzing the historical text data. Addressing these characteristics in analyzing the text data enhances the location identification process. Improving the text analysis process, which is foundational to the most of existing feature location approaches, will result in increasing the overall accuracy of feature location.

1.3 Research Objectives

With regards to the identified research problem, the main research question that needs to be addressed in this research is formulated as follows:

> *Does consideration of specific characteristics of the text data recorded in the software repository within the text analysis process make an improvement in feature location accuracy?*

To answer this research question, the following research objectives are formulated that need to be addressed in this research. The main research objective is:

To improve feature location by considering the specific characteristics of the text data recorded in the software repository within the text analysis process.

With respect to the main objective, a set of objectives are derived as follows:

- **Objective 1:** To study the existing text analysis methods applied for feature location and identify the current problems in the existing text analysis methods for feature location.
- **Objective 2:** To propose a feature location approach that considers differences between the text data recorded in the software repository and the text data in the natural language context to make an improvement in text analysis process of feature location.
- **Objective 3:** To evaluate the impacts of considering the specific characteristics of the text data recorded in the repository within the text analysis process of the proposed feature location approach.

In order to address the second objective, the characteristics of text data recorded in the software repository were investigated. As mentioned previously, incorporation of the data with metadata and large dataset size are two characteristics that differentiate the text data recorded in the repository from the text data in the natural language context. The metadata of time and developer are two important pieces of metadata that have the potential to make an improvement in the accuracy of MSR activities (Sisman & Kak, 2012; T. Zhang & Lee, 2013). More details on the rational of selecting these pieces of metadata are explained in Section 1.5. In relation with these metadata, it is possible to analyze the text data recorded in the repository from the aspect of (i) the aspect of time at which the data was recorded in the repository, (ii) the aspect of developers who recorded the data in the repository.

On the other hand, due to the storage of the history of changes applied to the repository's data (Kagdi, Collard, & Maletic, 2007), a larger size of data is stored in the repository compared to the natural language context. Although dataset reduction is one of the challenges in text analysis methods (Crain, Zhou, Yang, & Zha, 2012), the urgency of reducing the dataset size for the data recorded in the repository is more than that of in the natural language context due to the larger size of data in the repository. Thus, the characteristics of the repository's text data that are addressed in this research are: incorporation of data with metadata of time and developer, and the larger size of dataset. With respect to these characteristics, a set of sub-objectives are derived from the second objective of this research which are formulated as follows.

- **Objective 2.1**: To improve the accuracy of feature location by analyzing the text data recorded in the repository from the aspect of time at which the data was recorded in the repository.
- **Objective 2.2**: To improve the accuracy of feature location by analyzing the text data recorded in the repository from the aspect of developer who recorded the data in the repository.
- **Objective 2.3**: To reduce the size of dataset used for feature location.

Addressing these sub-objectives will result in addressing the second objective.

1.4 Research Approach

The first objective of this research (Objective 1) is addressed by conducting a comprehensive review on feature location literature with more focus on the feature location approaches that use text analysis methods. The main aim of reviewing the feature location

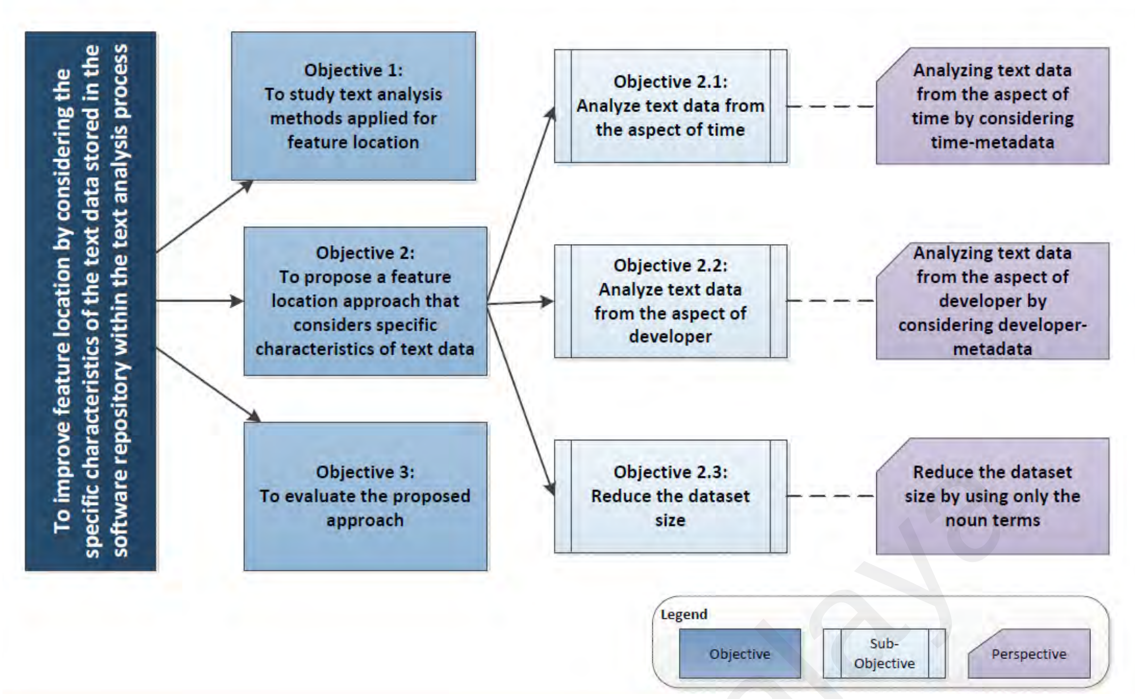

To improve feature location by considering the specific characteristics of the text data stored in the software repository within the text analysis process
Objective 1: To study text analysis methods applied for feature location
Objective 2: To propose a feature location approach that considers specific characteristics of text data
Objective 3: To evaluate the proposed approach
Objective 2.1: Analyze text data from the aspect of time
Objective 2.2: Analyze text data from the aspect of developer
Objective 2.3: Reduce the dataset size
Analyzing text data from the aspect of time by considering time-metadata
Analyzing text data from the aspect of developer by considering developer-metadata
Reduce the dataset size by using only the noun terms
Legend
Objective
Sub-Objective
Perspective

Figure 1.1: Three perspectives of considering specific characteristics of repository's text data in relation with the research objectives

literature is to identify the current problems of applying the text analysis methods for locating the software features in the source code of software systems.

As shown in Figure 1.1, to tackle the second objective (Objective 2), all identified sub-objectives have to be addressed. According to the sub-objectives, the identified characteristics of repository's text data should be taken into account in the text analysis process. These characteristics are considered in text analysis process in three perspectives. In the first perspective, the text data need to be analyzed from the aspect of time at which the data was recorded in the repository in order to more accurately locating the software features. In the next perspective, the text data is analyzed from the aspect of developers who recorded the data in the repository in order to more accurately locating the software features. The last perspective deals with reducing the size of dataset used for feature location. The investigation of the literature in other research areas such as traceability shows that the use of only the noun terms simply reduces the size of dataset (Capobianco, Lucia, Oliveto, Panichella, & Panichella, 2012). Accordingly, the third perspective aims at reducing the dataset size by using only the noun terms that exist in the text data recorded in the repository. As shown in Figure 1.1, each of these perspectives is corresponding to a different sub-objective. Addressing these perspectives in text analysis process results in satisfying the corresponding sub-objectives, and consequently, satisfying the second objective (Objective 2). The rationales behind identifying these perspectives are described in Section 1.5.

Finally, in order to address the last objective of this research (Objective 3), the proposed approach is empirically evaluated using a set of open-source projects. The results obtained from the experiments are compared with the baseline feature location approaches to approve the improvement which is obtained from considering specific characteristics of the text data recorded in the repository.

1.5 Rationales behind the Various Perspectives

As mentioned above, three perspectives were identified in order to satisfy the corresponding sub-objectives. These perspectives deal with the consideration of time-metadata, developer-metadata, and the use of only the noun terms. In the rest of this section, the reasons of identifying each one of these perspectives are briefly explained. More explanation on these reasons is presented in Chapter 4.

- **Considering Time-metadata:**

  The consideration of time-metadata in analyzing the text data for feature location is based on two main principles:

  - Defect localization: It is known that the most recent modifications to a software project are most likely the cause of future bugs or defects (Zimmermann, Weisgerber, Diehl, & Zeller, 2005; Hassan & Holt, 2005; Kim, Whitehead, & Zhang, 2008). By considering recent modifications in the source code, this would lead to finding relevant locations that are the cause of a new change request (Sisman & Kak, 2012).

  – Software evolution: Each software project has different goals and requirements in different periods of the project's lifetime (Gómez, Kellens, Brichau, & D'Hondt, 2009). For a given change request, the requested modification to the source code in the same time period of the project's lifecycle would likely have the similar goals or requirements. This principle bears further elaboration in Section 4.1.

- **Considering Developer-metadata:**

  The consideration of developer-metadata in analyzing the text data for feature location is based on the following principles:

  – There is an undeniable correlation between the data in the source code files and the project developers who are working on the source code files. Having one of these items (the name of source code file or name of developer) can lead to deriving the other (Kagdi, Gethers, Poshyvanyk, & Hammad, 2012). This correlation can be re-established by investigating the vocabularies and terms used by the developer in the source code file, thus revealing the developer's expertise (Schuler & Zimmermann, 2008).

  – In general, developers work on specific and related subjects and functionalities during software evolution (Bird, Pattison, D'Souza, Filkov, & Devanbu, 2008).

  – For a source code file, centralization of the developers' activities on a specific subject indicates the importance of the subject for the source code location from the developer's point of view. The relevancy of the subject, which is determined as an important subject from the developer aspect, increases the potential of the corresponding location to be the correct location of the change request. This principle is further elaborated in Section 4.1.

- **Using Noun Terms:**

  The use of only the noun terms for feature location is based on the following main reasons:

  - Reducing the general dataset size and reducing the amount of noises exist in the extracted entities from the data-sources (Sarawagi, 2008; Capobianco et al., 2012), thereby enhancing the effectiveness of improvements made by other means.
  - The noun usage in traceability (Capobianco et al., 2012) and bug assignment (Shokripour, Anvik, Kasirun, & Zamani, 2014) leads to the improvement of traceability and bug assignment results, respectively.
  - Using only nouns leads to an independence of the approach from dimensionality reduction methods, which is one of the challenges in IR methods (Crain et al., 2012).

1.6 Scope of Research

The abstract research area of this thesis is Software Engineering, concentrated on software evolution, program analysis, and empirical studies. In particular, this thesis focuses on identifying a source code location pertinent to a software feature which is known as feature location. Since the most commonly used data analysis method for feature location is text analysis (Dit et al., 2013), the core of this research is based on the use of text analysis for feature location.

The main objective of this thesis is improving the feature location accuracy by considering the specific characteristics of repository's text data, i.e. incorporation of the data with metadata and larger size of dataset. With respect to these identified characteristics, three sub-objectives are formulated that deal with, (i) analyzing the text data from the aspect of time, (ii) analyzing the text data from the aspect of the developer, and (iii) reducing the dataset size. Addressing these objectives in the text analysis process of feature location leads to improving the overall accuracy and effectiveness of feature location process.

Since the aim of feature location is identifying the source code entities pertinent to change requests, the main text data which is basically used for feature location is the source code of software systems. As the source code and its relevant documents are recorded and managed in Version Control System (VCS), the main software repository used for feature location is VCS. Therefore, the expression of software repository in this thesis is mainly refer to VCS. According to the most recent survey on feature location (Dit et al., 2013), most of the existing feature location approaches which are based on text analysis do not deal with the historical information that were recorded in the VCS. The existing feature location approaches mostly used a specific release of a software project as the data source. However, consideration of history of changes of the source code has a great potential to improve the accuracy of feature location. Thus, this research uses the whole history of changes occurred on the source code of the software project in order to more accurately perform the location identification process. Thus, in this research, all the data recorded in the VCS is collected and used to prepare the dataset for feature location process.

Since the data recorded in the repositories of the open-source projects is freely available, these software projects are used as the datasets in this research. Availability of the data of open-source projects supplies the access to rich repositories of large-scale projects that are used by thousands of researchers for MSR activities and provides a reality evaluation (Hassan, 2008). Thus, required data in this research is collected from the VCSs of a set of open-source software projects. To prepare the datasets, all the data recorded in the VCSs of these projects is collected and preprocessed to be used in location identification process.

1.7 Significance of Research

As mentioned in earlier, feature location is one of the frequent activities of software maintenance which is an important, time-consuming and costly phase of software development. Effective support for maintenance activities, such as feature location, leads to the improvement of software quality and reduction of software development cost and effort. In addition, feature location would be considered as one of the traceability activities that aims to link source code entities to software features (Marcus & Haiduc, 2013). The ability of recovering traceability links between software artifacts is an important factor for the effective development and maintenance of a software system (Y. Zhang, Witte, Rilling, & Haarslev, 2006; Winkler & von Pilgrim, 2010), which explicitly helps to improve the quality of software development companies (Winkler & von Pilgrim, 2010).

Moreover, feature location, in addition to enhancing the addressing of change requests, is used for some other aspects such as searching in the source code (Hill et al., 2009), reverse engineering (Marcus & Haiduc, 2013) and bug assignment (Kagdi et al., 2012). Due to the incrementally changing and continuous growth of software projects, identifying the related source code elements to a specific feature or task among thousands of source code files becomes increasingly difficult for either the new and former developers of the project. Furthermore, as the developers who developed the software project are different from the maintainers, there is an essential need for feature location approaches for searching in the source code context. Thus, searching the source code of a software project helps the project's developers, especially the developers who are new in some or all parts of the project, to have a better understanding of the system.

Due to the significance of the feature location activity, more effective support for this activity is needed to obtain a sustainable, high-quality evolution of a software system. Considering metadata, such as time and developer, and reducing the dataset size can enhance the feature location approaches which are based on the text analysis process. As mentioned previously, text analysis is the most commonly used analysis technique for feature location which is the basis for different categories of feature location techniques (Dit et al., 2013), thus improving this process leads to improvement in the overall accuracy of feature location activity.

1.8 Thesis Organization

The following chapters of this thesis are organized as follows:

In Chapter 2, an overview of the related works on the feature location literature, with focus on the researches that used text analysis is given.

Chapter 3 explains the methodology which is applied in this research in order to address the identified objectives. This chapter presents the details of the research methodology steps, including literature review, problem statement and research objective formulations, the proposed approach, and experimental evaluation.

In Chapter 4, the feature location approach which is proposed in order to satisfy the research objectives is described in detail. In order to propose the main approach, two intermediate methods are proposed that consider the metadata of time and developer in the text analysis process. In relation with the metadata, in these proposed methods, the data are analyzed from the aspect of time at which the data was recorded and the aspect of the developer who recorded the data in the source code of software project. These two methods embody the proposed approach that analyzes the data from both the aspect of time and the aspect of the developer. The proposed methods and approach use only the noun terms to identify the related source code location with the desired change request.

Next, the proposed approach is evaluated through several subject systems with different scales of project using different metrics. Chapter 5 explains the setup for the experimental evaluation of the proposed approach.

The results of the experiments are reported and analyzed in Chapter 6. The results are assessed in compared to the most commonly used text analysis-based methods that have been used in feature location.

Finally, Chapter 7 concludes the whole research work that presented in this thesis. In this chapter, the achievement of the objectives and the contributions of the research are highlighted. Finally, the limitations of the study and future directions of the research are described.

CHAPTER 2: LITERATURE REVIEW

Feature location is known as the process of identifying an initial location in the source code of a software project pertinent to a specific software feature[1]. Feature location is a research area closely related to the research areas of concept location and bug localization, and is also one of the most commonly used activities performed by project developers during software development and maintenance. This activity is an essential part of the incremental change process (Dit et al., 2013) and is a prerequisite to some of Mining Software Repositories (MSR) activities such as program comprehension (Poshyvanyk, Gueheneuc, Marcus, Antoniol, & Rajlich, 2006).

Feature location is an active research area that has attracted a lot of research effort resulted in proposing many automatic and semi-automatic feature location approaches. This chapter reviews the feature location literature highlighting the most related studies. The research study presented in this thesis covers the consideration of time-metadata, and developer-metadata as well as the use of only the noun terms for locating the software features in the source code of software systems. Thus, the research area of feature location and some other related research areas such as traceability recovery and bug assignment are investigated to find the related studies. Figure 2.1 shows the structure that was followed to review the related studies in this chapter.

In this chapter, first of all, an overview of the studies conducted in feature location research area is presented in Section 2.1. In this section, the main categories of data analysis methods that have been used for feature location are introduced highlighting the most commonly used category, which is text analysis. Next, in Section 2.2, the existing feature location approaches that rely on using text analysis methods are described. In

[1]A feature can be considered as a special concept associated with the functionality of the system which is visible for the end user.

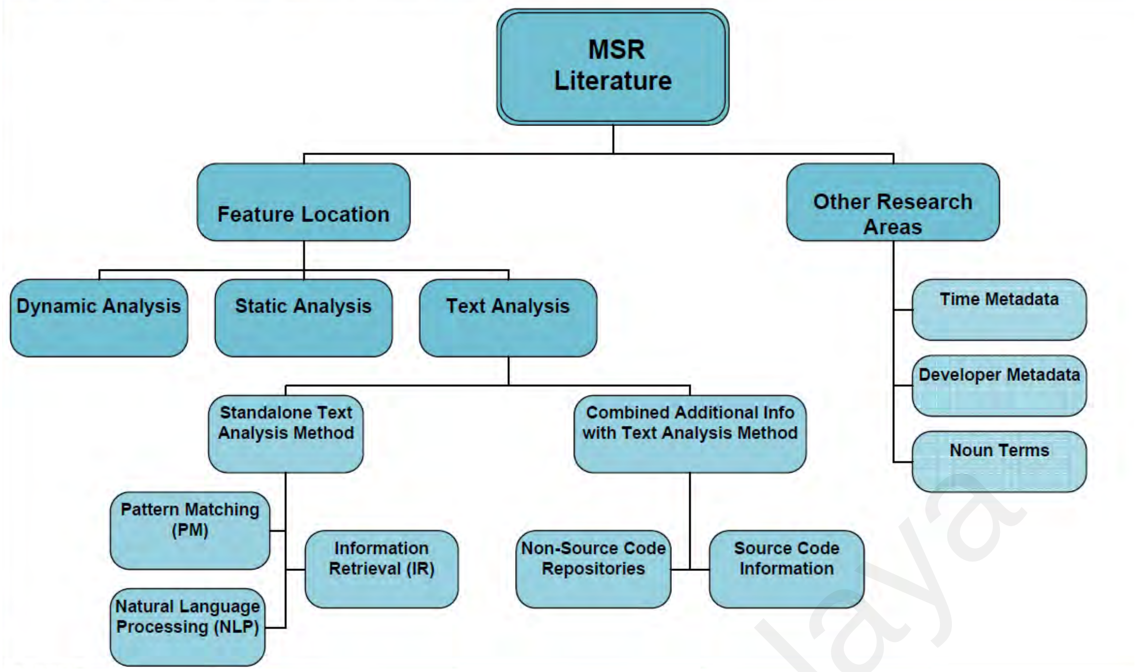


Figure 2.1: Hierarchy of topics reviewed in literature review chapter

Section 2.3, the approaches that improved the text analysis by combining the text data with additional information are investigated. Since this research aims to consider the metadata of time and developer, and use noun terms to improve the feature location accuracy, the literature in the area of feature location and other related research areas is further investigated to identify the related studies in Section 2.4. Finally, a summary of this chapter and the inferences are explained in Section 2.5.

2.1 Overview of Feature Location

Wilde et al. (Wilde & Scully, 1995) and Biggerstaff et al. (Biggerstaff et al., 1993) were the first researchers who focused their efforts on addressing the problem of feature location. Biggerstaff et al. (Biggerstaff et al., 1993) implemented a tool to extract the identifiers in the source code of the project, and then cluster the extracted identifiers to support identification of concepts. Wilde and his colleagues (Wilde & Scully, 1995) used the Software Reconnaissance method that analyzes the dynamic information to address the problem of feature location. Since then, feature location has attracted many research efforts resulted in the proposal of many automatic and semi-automatic feature location approaches that use different types of methods for data analyses. According to the most recent survey on feature location (Dit et al., 2013), the existing feature location approaches are classified into three main categories, based on the applied method for data analysis method:

- Dynamic analysis
- Static analysis
- Text analysis

Dynamic analysis uses the information obtained from the execution of software to simulate the requested feature, and it is often used when features can be invoked and observed during run time (Cornelissen, Zaidman, Van Deursen, Moonen, & Koschke, 2009; Eisenberg & De Volder, 2005). Instead, as its name implies, static analysis deals with the static properties of software project like class structure, dependency graphs and control/data flow dependencies (Chen & Rajlich, 2000; Lukins et al., 2010; Marcus, Maletic, & Sergeyev, 2005; Robillard, 2008). Finally, text analysis investigates the text data stored in the historical project repositories and analyzes them to extract useful information (Cleary et al., 2009; Gay, Haiduc, Marcus, & Menzies, 2009; Lukins et al., 2010; Poshyvanyk et al., 2007). The properties of these categories as well as their strengths and weaknesses are presented in Table 2.1.

According to the itemized strengths and weaknesses in Table 2.1, dynamic and static analysis may not always be available in real-world scenarios. Furthermore, these types of data analysis impose considerable overhead to the system, due to extracting dynamic and static information (Dit et al., 2013). On the other hand, most of the data recorded in software repositories, i.e. software requirements or source code, are in text format

Table 2.1: Main properties of dynamic, static, and text analyses

	Dynamic Analysis	**Static Analysis**	**Text Analysis**
Prerequisites	- Needs compilable source code () - Preferred when the requested feature is able to be exercised during runtime	- Needs compilable source code	- Needs neither an executable software system, nor a compilable source code
Input	- Scenario	- Program Elements	- Query
Properties	- Deals with the program execution behavior - Develops a set of scenarios that exercises only the requested feature and then extracts the information of the execution traces - Works based on finding differences between the control flows of the passing and failing of a set of test cases that exercises the requested feature	- Deals with the static properties of software project like class structure, dependency graphs and control/data flow dependencies - Close to the real activity of the project developers that performed to search a code related to a software feature	- Deals with the text data recorded in the historical project's repositories to extract beneficial information - Relies on the assumption that identifiers, comments, and other text data found in software repositories contain domain knowledge that can be used for locating software features
Strengths	- Mitigates the dataset size that needs to be investigated to identify the related source code locations (due to developing a bounded set of scenarios that exercises only the requested feature) (Cornelissen et al., 2009). - Provides worthy information to correctly locate the software feature that may not be obtained from other types of data analysis (Poshyvanyk et al., 2007).	- Requires neither a working software system, nor a test case that exercises the requested feature (Dit et al., 2013). - Provides useful information for identifying the source code elements related to a software feature that may not be obtained from other types of data analysis (Marcus et al., 2005).	- Always available in real-world systems (Dit et al., 2013). - Most of data recorded in software repositories, i.e. software requirements or source code, are in text format and also change requests that in many systems are considered as the only available sources for conducting the feature location process are written in text format (Kagdi et al., 2012).
Weaknesses	- Imposes considerable overhead to the system, due to developing and executing collection of traces (Lukins et al., 2010; Dit et al., 2013). - Deficiency of developing a concise set of scenarios to invoke all parts of the source code which are related to the implementation of the requested feature (Chen & Rajlich, 2000). - Difficulty of formulating a set of scenarios that only invokes the desired feature (this phase may result in executing irrelevant code that imposes further overhead to the system) (Chen & Rajlich, 2000). - Dependent on the quality of the identified test cases (Gay et al., 2009; Lukins et al., 2010). - High cost in distributed and time-sensitive systems (Robillard, 2008). - Unable to deal with the non-executable files, like the files used for configuration or documentation (Dit et al., 2013).	- Often results in returning many false positives which is due to overestimating the source code elements recognized to be pertinent to the desired feature (Eisenberg & De Volder, 2005). - Imposes overhead to the system, due to extracting structured information (Dit et al., 2013). - Static information which is derived from structural information, such as call graphs, is language specific (Robillard, 2008). - Unable to deal with the non-executable files, like the files used for configuration or documentation (Dit et al., 2013).	- Applied text analysis methods for examining the text data for feature location mostly originate from a natural language context (Bassett & Kraft, 2013). - Considers the text data in a source code file as a collection of terms that co-occur frequently in the documents of corpus (Bassett & Kraft, 2013). - Ignores the differences of text data recorded in software repositories and text data in a natural language context within text analysis process (Bassett & Kraft, 2013).

(Karahasanović, Levine, & Thomas, 2007; Noice, 2013). Moreover, change requests that in many systems are considered as the only available sources for conducting the feature location process are written in natural language text (Gethers et al., 2011). Due to the enumerated reasons, more effective support for the analysis of text-based sources is needed to obtain a high-quality feature location process that more accurately locates the software change requests. In addition, improvement in the accuracy of text analysis process of feature location will affect the overall accuracy of feature location that often works as a starting point in many MSR activities such as impact analysis.

The research of Dit and his colleagues (Dit et al., 2013) shows that the most commonly used category of feature location method is text analysis, since more than 51% of the published literature in this research area is based, at least in part, on text analysis. Thus, it is reasonable to assume that text-based sources are the most important sources for feature location and text analysis plays the most significant roles in locating features in the source code. These reasons encouraged the researcher to focus on improving text analysis process of feature location to locate software features more accurately. In the rest of this section, some of the text analysis based feature location approaches are investigated to find their strengths and weaknesses. Since this thesis mainly focuses on the use of metadata in the text analysis process, the feature location literature is investigated to find the most related works. Accordingly, first, the main categories of the text analysis methods used for feature location are reviewed. Then the use of metadata in text analysis based feature location approaches is investigated.

2.2 Text Analysis based Feature Location

The text analysis methods that have been used for feature location are mostly classified into three main categories (Dit et al., 2013):

- Pattern Matching (PM)
- Information Retrieval (IR)
- Natural Language Processing (NLP)

Table 2.2 presents some of the properties of these categories of text analysis methods highlighting their strengths and weaknesses. In the follows, some of related studies that use these categories of text analysis are reviewed.

2.2.1 Related Studies on Pattern Matching (PM) Methods

PM, which is the first category of text analysis methods, is considered as the simplest text analysis feature location method. It usually involves a text-based technique to search the source code using a utility such as regular expression matching tools, "grep" being an example. One of the studies that utilized PM for feature location was performed by Petrenko et al. (Petrenko et al., 2008). They combined the use of *grep* with ontology fragments to develop a feature location approach. In this study, the ontology fragments were used to record partial domain knowledge about software system features in order to enhance developers to formulate suitable queries and also help them to investigate the retrieved results. The ontology used in this study was generated manually by the developer and it can be refined and expanded during the feature location process due to more knowledge about the system gained by the developer.

The work of Petrenko et al. was extended by Wilson (Wilson, 2010). The author presented an approach that systematically formulates queries based on the obtained knowledge from ontology fragments. In this approach, developers are able to formulate desired queries based on the terms recommended in the ontology fragments and then input the formulated queries to *grep* to locate software features in the source code. Furthermore, many modern development environments, such as Microsoft Visual Studio and Eclipse,

Table 2.2: Main properties of PM, IR, and NLP methods

	PM	IR	NLP
Properties	- Query based - Searches the source code using a text-based technique such as regular expression matching tools - Looks for exact matching	- Query based - Works based on statistical methods - Deals with the frequency of terms/words in documents to determine the amount of similarity between documents and queries - Ranks the documents of corpus according the determined similarity scores that indicate their relevance with the given query	- Query based - Analyzes the parts of speech of the words used in source code
Strengths	- Simplest category of text analysis methods (Dit et al., 2013) - Extended and utilized in many modern development environments (Poshyvanyk, 2008) - Relatively robust but not very precise (Wilson, 2010)	- Made a significant improvement over PM (Dit et al., 2013) - Accepts more general queries (Cleary et al., 2009) - Identifies the source code locations more accurately compared to PM (Dit et al., 2013) - Handles multiple-word queries (Marcus et al., 2004) - IR is not as fragile as PM and it does not return un-ranked result sets (Marcus et al., 2004)	- Considers grammatical categories of terms within the text to analyze and extract the required information from the documents (Shepherd, Pollock, & Vijay-Shanker, 2006) - Typically uses ontology/glossary/dictionary to consider other forms of terms appearing in a given query (Abebe & Tonella, 2010; Petrenko et al., 2008) - more precise than PM and IR (Dit et al., 2013) - Can be used to extend either source code terms and query terms (Dit et al., 2013) - No indexing is necessary (Shepherd et al., 2006) - Can be used beside PM and IR methods to address feature location problem more accurately (Hill et al., 2009)
Weaknesses	- Chances of a programmer choosing query terms that match the vocabulary of unfamiliar source code are relatively low (Marcus et al., 2004) - Regular expression matching typically used for PM is extremely fragile (Wilson, 2010) - Features such as morphology changes, synonyms, line breaks, and reordered terms will cause regular expression queries to fail (Petrenko et al., 2008) - Low recall caused by using regular expression queries (Abebe & Tonella, 2010) - Commonly occurring sub-strings in the code cause even mildly broad search terms to return large result sets, leading to low precision searches (Marcus et al., 2004) - Typically fails to handling searches for two interacting words that occur near to each other (caused by line breaks and word ordering changes break most simple regular expression searches) (Petrenko et al., 2008)	- Typically uses all terms in the document after common preprocessing steps (Rao & Kak, 2011) - Considers a document as a collection of terms that co-occur frequently (Bassett & Kraft, 2013) - Returns considerable false positives because it does not account for sentence structure (Dit et al., 2013) - Rarely handles natural language issues such as morphology or synonym that lead to low recall (Lukins et al., 2010)	- More difficult and more expensive than PM and IR (Dit et al., 2013) - Inability to address ill-formed sentences (Dit et al., 2013) - Inability to address syntactic ambiguities caused by structure (Hill et al., 2009)

provide a lot of useful add-ons based on simple pattern matching to handle the activities like referencing to class and method names (Poshyvanyk, 2008).

However, pattern matching methods are simple methods, they may not be highly accurate due to some weaknesses which are summarized in Table 2.2. To address this issue, the researchers (Marcus et al., 2004; Poshyvanyk et al., 2006) made a significant improvement over pattern matching by proposing Information Retrieval (IR) based approaches that accept more general queries and identify the source code locations more accurately.

2.2.2 Related Studies on Information Retrieval (IR) Methods

The main properties of IR models were outlined in Table 2.2. The IR methods analyze the source code documents and determine the amount of similarity between the source code documents and a query which is typically formulated by a software developer in natural language. The source code documents are finally retrieved as a list that are ranked based on their relevancy to the desired query (Cleary et al., 2009; Gay et al., 2009; Poshyvanyk et al., 2007).

The popular IR models that are typically used for feature location, i.e. Vector Space Model (VSM), Latent Semantic Indexing (LSI), Latent Dirichlet Allocation (LDA), and Smoothed Unigram Model (SUM), are explained in Appendix B. These methods are applied in many feature location approaches to locate software features in source code.

The research study of Marcus et al. (Marcus et al., 2004) is one of the earliest feature location studies that apply IR for concept location. This study introduced the use of LSI (Deerwester, Dumais, Furnas, Landauer, & Harshman, 1990), which is an extension of VSM, to identify the source code elements related to a concept written in natural language. In this study, first, the source code elements were indexed based on the extracted terms from the identifiers and comments as the documents of corpus. Next, Singular Value Decomposition (SVD) technique was used to map the documents into the LSI space. Then, the requested queries were also mapped into the LSI space using the same process performed for the source code elements. Finally, the similarity between the source code elements and the requested queries was determined using VSM. Another IR-based approach was proposed by Lukins et al. (Lukins, Kraft, & Etzkorn, 2008; Lukins et al., 2010). This study is the first that utilized LDA (Blei, Ng, & Jordan, 2003) as an alternative model to LSI for bug localization.

One of the recent studies on the use of IR models for feature location is the study of Wang et al. (S. Wang, Lo, & Lawall, 2014). This study investigated different schemes of VSM to optimize the feature location problem. The multiple VSM variants with different weighting schemes have different performance for software systems. This study considered 15 VSM variants with a different TF-IDF[2] weighting scheme and proposed a genetic algorithm based composite model. Using genetic algorithm, the space of possible compositions of various VSM variants is explored. The proposed approach in this study is search-based compositional bug localization that includes two main phases involving training phase and deployment phase. In the first phase, the approach is trained using a set of bug reports which are associated with the source code files that were modified to satisfy the bugs. To discover near-optimal compositions with the best performance on the training set, search heuristics are employed to identify promising compositions by traversing the search space. In the second phase, deployment, the composite model is applied to locate new bug reports.

The use of popular IR models, i.e. VSM, LSI, LDA, and SUM, was examined for the purposes of feature location in the studies of Rao and Kak (Rao & Kak, 2011), Zhou et al. (Zhou, Zhang, & Lo, 2012), and Wang et al. (S. Wang, Lo, Xing, & Jiang, 2011).

[2]Term Frequency-Inverse Document Frequency

According to the research of Dit et al. (Dit et al., 2013), IR is the most commonly used category of text analysis methods applied for feature location. However, feature location approaches that use IR methods have some weaknesses that are briefly mentioned in Table 2.2. IR-based feature location approaches typically use all the terms in the document after common preprocessing steps. Also, IR considers the text resources as a collection of terms that co-occur frequently in the documents of the corpus (Manning et al., 2008). As mentioned earlier, text data in the software repositories is associated with additional data, such as metadata that records the evolution of the text, not found in simple text documents that can be considered in text analysis process in order to improve the feature location accuracy.

2.2.3 Related Studies on Natural Language Processing (NLP) Methods

The last category of text analysis feature location approach, Natural Language Processing (NLP), utilizes the NLP techniques to provide the ability of dealing with the role of terms in sentences to locate the software features more accurately. NLP-based feature location approaches address some of weaknesses of IR-based approaches such as considering morphology or synonym. Furthermore, NLP-based approaches consider different categories of terms appeared in the requested query or source code elements to locate the features in the source code. The other properties of NLP-based feature location approaches are presented in Table 2.2. In this category of approaches, NLP techniques are typically used to extract desired information from source code elements in order to enhance and facilitate the process of queries extension and refinement.

The approach applying NLP for feature location in the study of Shepherd et al. (Shepherd et al., 2006) provided a natural language representation of the source code elements, named AOIG (Action-Oriented Identifier Graph) deal with the verbs and their corresponding direct objects (verb-DO) appeared in source code identifiers. AOIG enhances the query formulation process by suggesting a set of direct objects for the verbs requested from the developer.

The other study on applying NLP for feature location was performed by Hill et al. (Hill et al., 2009). They proposed a NLP-based feature location approach that extended and refined the requested query to improve location identification process. The approach proposed by Hill and his colleagues extracted the noun, verb, and prepositional phrases from method and field names and also generated additional phrases by analyzing the methods' parameters. Then, a hierarchy of all the extracted phrases, which are linked to their corresponding source code elements, was created based on partial phrase matching. Once a user has formulated a query, the approach inspected the phrases for matches and returned a hierarchy of phrases and their corresponding source code elements which are related to the formulated query.

Moreover, Abebe and Tonella (Abebe & Tonella, 2010) introduced a feature location approach that uses NLP techniques to extract concepts from source code identifiers, and forms candidate sentences in which those identifiers are used. The formulated sentences act as an input of an ontology that involves the identifiers (represent the concepts) and their relations (extracted from the formulated sentences). The functionality of this approach is similar to the one proposed by Petrenko et al. (Petrenko et al., 2008) with one main difference related to the way of creating the ontologies. Opposed to the Petrenko's approach, the ontology is automatically generated in the approach of Abebe and Tonella.

2.2.4 Summary of Text Analysis based Feature Location

These three categories of text analysis methods which are originated from the natural language context (Bassett & Kraft, 2013) can be used in feature location approaches as the standalone methods to perform the feature location activity. The text analysis based approaches mentioned in this section are some examples of using text analysis methods as the standalone methods for feature location. Some of important properties of these approaches are summarized in Table 2.3. Since these approaches used PM, IR, and NLP as the standalone methods for feature location without considering additional information, their main strengths and weaknesses are related to their corresponding text analysis methods mentioned in Table 2.2.

Table 2.3: Properties of related feature location approaches that used text analysis as the standalone method for feature location

Approach	**Text Analyzer**	**How to Locate**	**Evaluation Metric**	**Subject Sys.**	**Object Sys.**	**Comparison Sys.**
(Petrenko et al., 2008)	PM	Combine regular expression matching and manually created NLP ontology	F-Measure; Precision and recall; TopN Ranked; Time	Eclipse and Mozilla	10 bug reports: 5 from each Subj. Sys.	lexical-based (ELex), IR-based (GES)
(Wilson, 2010)	PM	Combine grep with ontology fragments called grepOF	Statistical test of ranked list of source code elements	Eclipse and Mozilla	8 bug reports: 4 from each Subj. Sys.	*grep*
(S. Wang et al., 2014)	IR (VSM)	Considered 15 VSM variants with a different TF-IDF weighting scheme and proposed a genetic algorithm based composite model	Hit, MAP, and MRR	AspectJ, Eclipse, and SWT.	3,459 bug reports	15 various VSM variants
(Marcus et al., 2004)	IR (LSI)	First use of LSI for feature location	Precision and recall	NCSA Mosaic	8 queries written by programmers	grep and ASDGs
(Lukins et al., 2008, 2010)	IR (LDA)	First use of LDA for feature location	Effectiveness; TopN Ranked; Average of ranks	Eclipse, Mozilla, and Rhino	5 of Mozilla; 219 of Eclipse; 106 of Rhino	LSI
(Shepherd et al., 2006)	NLP	Define an Action-Oriented Identifier Graph (AOIG) and connect code segments through the actions performed by the programmers	Precision and recall; Time and space costs	JHotDraw	–	Manual process
(Hill et al., 2009)	NLP	Information Extraction	F-Measure	Rhino	19 change requests	Shepherd approach (verb-DO)
(Abebe & Tonella, 2010)	NLP	Use natural language parsing to extract data from program identifiers in order to automatically create NLP ontology	F-Measure; Precision and recall	WinMerge	7 bug reports	Manual process with and without using ontology

The focus on using only the text data extracted from source code elements such as identifiers may not be highly accurate in identifying the correct location of software features.

2.3 Combining Additional Information with Text Analysis

In order to improve the accuracy of feature location approaches which are based on text analysis process, many research studies focused on using additional data such as structural information besides text data for feature location. Some of the related studies that combined text analysis methods with additional information is outlined in Table 2.4. This table indicates important properties of related feature location approaches that combine text analysis with additional information such as static, dynamic or historical data to improve accuracy of feature location process. The third column in this table explains the additional information which is used in the corresponding text analysis based feature location approach. The next column briefly describes how the additional information is used besides text data to locate a software feature. The way that the approaches located the software features in the source code of software projects is explained in more details in the rest of this section. The fifth column shows the metrics that were used in the experimental evaluation of the feature location approach. The next two columns respectively point to the software projects that were used as the case study to evaluate the approach, called subject systems, and the set of samples, e.g. bug report or change request, which were selected from the subject systems for the evaluation, called object systems. To show the improvement made by the desired approach in feature location process, the last column in Table 2.4 indicates the related feature location approaches that were used to assess the improvement made by the approach in feature location. The approaches outlined in this table and the way they used additional information for feature location are explained in the rest of this section.

Table 2.4: Properties of related feature location approaches that combined text analysis with additional information

Approach	**Text Analyzer**	**Additional Information**	**How to Locate**	**Evaluation Metric**	**Subject Sys.**	**Object Sys.**	**Comparison Sys.**
(Zhao, Zhang, Liu, Sun, & Yang, 2006)	VSM	Static Information extracted from structural correlation between source code elements	Used VSM to rank source code elements. Then, used Branch Reserving Call Graphs (BRCG) to further recover the relevant elements for requested feature	Precision and Recall	GNU DC and UnRTF	27 features: 21 of DC and 6 of UnRTF	VSM, and a dynamic approach
(Scanniello & Marcus, 2011)	VSM	Static information extracted from clustering the similar source code elements based on combining both structural and textual analysis	Indexed source code elements using VSM and then compared the indexed elements with the neighboring elements in a dependency graph and create new weighted graph. Finally, clustered the source code elements using BorderFlow	Effectiveness	Atunes; Art of Illusion; Eclipse; jEdit	198 bug reports: 12 for Art of Illusion, 30 for Atunes, 114 Eclipse, 41 jEdit	VSM
(Alhindawi, Dragan, Collard, & Maletic, 2013)	LSI	Static information extracted from stereotype information	Identified method's stereotype by analyzing static and structural information of the method elements and then added a comment before the method defining the method's stereotype.	Precision and Recall; Rank of the first, and the last relevant elements	HippoDraw, Qt	22 features: 11 for each Subj. Sys.	LSI with and without the added stereotype information
(Saha, Lease, Khurshid, & Perry, 2013)	Indri	Static information extracted from Structural correlation between source code elements	Built the abstract syntax tree (AST) for each source code elements. Then, created structural view of elements and indexed them using Indri.	TopN Ranked; MRR; MAP	Eclipse, AspectJ, SWT, Zxing	3400 bugs	BugLocator
(Ye, Bunescu, & Liu, 2014)	VSM	API descriptions and previously fixed bug reports	Given a bug report, the ranking score of each source file is computed as a weighted combination of an array of features encoding domain knowledge, where the weights are trained automatically on previously solved bug reports using a learning-to-rank technique.	Accuracy, MAP, and MRR	AspectJ, Birt, Eclipse, JDT, SWT, and Tomcat	22,000 bugs	VSM, Usual Suspects, BugLocator, and BugScout
(S. Wang & Lo, 2014)	DFR and LM	Version history, related bug reports, and structure	Integrated a bug prediction technique used in Google, with a bug localization technique called BugLocator, and the bug localization technique BLUiR.	MAP	AspectJ, Eclipse, SWT, and ZXing	3,000 bugs	bug localization solution of Sisman and Kak, BugLocator, BLUiR
(Poshyvanyk et al., 2007)	LSI	Dynamic information extracted from execution traces	Combined the results of LSI (as a text analysis method) and the results of SPR (as a dynamic analysis method)	Effectiveness	Mozilla, Eclipse	3 bugs of Eclipse and 5 bugs of Mozila	LSI and SPR

(Revelle, Dit, & Poshyvanyk, 2010)	LSI	Dynamic and static information extracted from structural relation between source code elements and execution traces that are analyzed using web mining	Ranked source code elements, which were obtained from dynamic analysis, using LSI. Then, used web mining algorithms, i.e. HITS and PageRank to filter irrelevant source code elements	Effectiveness	Eclipse and Rhino,	45 features of Eclipse and 241 features of Rhino	LSI, Web mining, and execution traces as a standalone approach
(Le, Oentaryo, & Lo, 2015)	VSM	Dynamic info. extracted from analysis of program spectra	Used three components to score the words appeared in the textual description of bugs, program spectra, and both of them. Then, combined these three scores to determine total score.	Accuracy (Top N), and MAP	AspectJ, Ant, Lucene, and Rhino	157 bugs	IR$_{LSI}$Dyn$_{bin}$, PROMESIR, LR and MULTRIC
(Sisman & Kak, 2012)	DFR and LM	Information extracted from version histories, i.e. number of bugs, number of changes to the code element, and time	Defined a coefficient of probability of defectiveness of the source code elements by analyzing version histories of software systems, and added this coefficient to DFR and LM methods	MAP	AspectJ	iBugs dataset for AspectJ	Probabilistic IR models, i.e. DFR and LM
(Zhou et al., 2012)	VSM	Length of source code elements and the information of the similar bugs	Added a coefficient to VSM to score the source code elements based on the file length. Furthermore, used previously fixed bugs to index source code elements	TopN Rank; MRR; MAP	Eclipse, AspectJ, SWT and Zxing	3000 bug reports	VSM, LDA, SUM, LSI
(J. Wang, Peng, Xing, & Zhao, 2013)	TF-IDF	Package Structure, Inheritance Hierarchy, Intent, and called method info.	Allows developers to begin with an initial query and gradually refine it based on previous feedbacks	Time and F-measure	JEdit	20 developers performing four feature location tasks	Grouped developers in two groups, experimental group and control group, each one includes 10 developers
(Scanniello, Marcus, & Pascale, 2015)	VSM	Link analysis	Augmented text retrieval based approaches by using link analysis techniques to rank the source code elements during text retrieval	effectiveness	Art of Illusion, aTunes, Eclipse, jEdit, Cocoon, Derby, Lucene, and OpenJPA	175 bug reports	VSM and CLC (Concept Location using Clustering)
(Hill, Pollock, & Vijay-Shanker, 2007)	TF-IDF	Information extracted from structural correlation between source code elements and position of appearance the terms in the source code element	Combined text analysis with structural representations of code and scored the terms based on the frequency of appearance in the corresponding source code element and where the query terms appear in the source code element.	Precision and Recall; F-Measure	Gantt, jBidWatcher, Freemind	8 features: 1 for Gantt, 4 for jBidWatcher, 3 for Freemind	Suade, boolean-AND, and boolean-OR
(Bassett & Kraft, 2013)	LDA	Information extracted from structural correlation between source code elements	Assessed different configuration for LDA to weight the terms based on the importance of terms derived from a method's name and from the names of the methods it calls.	Effectiveness and MRR	ArgoUML, Eclipse, JabRef, jEdit, muCommander	400 features and bugs in five open-source Java systems	Compare different configurations of LDA

(Poshyvanyk & Marcus, 2007)	LSI	Formal Concept Analysis information extracted from shared terms between source code elements	Ranked source code elements using LSI, and then clustered the ranked results using FCA	Effectiveness	Eclipse	2 bug reports of Eclipse	LSI
(Cleary & Exton, 2007)	LM (cog-nitive assign-ment)	Information derived from non-source code repositories extracted from relation be-tween terms co-occurrence in non-source code repositories	Defined a model of relationships between terms recorded in source code elements, then when two terms are compared, if there is no direct corre-spondence, consult the relationship model to see if an indirect match can be made.	Precision and Recall; Average precision	CHIVE	4 concerns	Classic LM, de-pendency based LM, and LSI
(Moreno, Tread-way, Marcus, & Shen, 2014)	Lucene (VSM)	Stack traces extracted from bug reports and dependency graph	Measured the textual similarity of source code el-ements with the requested query, and then com-bined the scores with structural information col-lected from stack traces found in bug reports, and dependency graphs of software system	Effectiveness; MRR; MAP	ArgoUML, BookKeeper, Derby, Hiber-nate, JabRef, JEdit, Lucene, Mahout, Um-Commander, OpenJPA, Pig, Solr, Tika, ZooKeeper	155 bug reports from 14 Subj. Sys.	Lucene (VSM)

The information used to improve the text analysis based feature location approaches is derived from the source code repository or non-source code repositories. The information derived from source code can be extracted from compiled or executed source code, such as the dynamic and static information. Moreover, this information can be related to the source code entities such as the specific properties of the entities or relationships among the entities. Examples of these types of extracted information are explained through the feature location approaches which are listed in Table 2.4.

2.3.1 Information Extracted from Compiled or Executed Source Code

The static and dynamic information are derived from the dependency graphs and execution traces which are extracted from the compiled and executed source code. The first study that combined static analysis with text analysis for the purpose of feature location is the study of Zhao et al. (Zhao et al., 2006). This study presented a static, non-interactive feature location approach, called SNIAFL that used Branch Reserving Call Graphs (BRCG)[3] (Tao, Lu, Zhiying, Dan, & Jiasu, 2003) to provide additional information for feature location. This approach concentrates on determining both the specific source code elements and relevant source code elements. Specific source code elements were definitely used to implement a specific feature but have not been used for any other features. Relevant source code elements involve all source code elements that contribute to the implementation of the desired feature. Using IR models, especially VSM, the initial connections between features and specific source code elements were prepared. In this step, for each feature, a list of source code elements ranked based on the amount of similarity with the desired feature was produced. To shorten the list, only the elements with high similarity values (higher that a determined threshold) were used as the initial set of specific source code elements for the desired feature.

[3]BRCG is an extension of call graph that involves branches and sequential information.

After this step, the BRCG which was extracted from the source code is pruned for each feature based on the obtained initial set of specific elements. Then, the branches which were found to be irrelevant to the initial set were pruned. The remaining elements in the pruned BRCG were considered as the set of relevant source code elements to the feature. In the last step, the BRCG was traversed again to specify the features which were found to be relevant to any other feature. The output of this approach is a pseudo execution trace for each feature. Overall, this study investigated the impacts of using static analysis in text analysis based feature location approaches and indicated the importance of structural information in improving the accuracy of these approaches.

Similar to Zhao et al. (Zhao et al., 2006) that combined text analysis with structural representations of source code elements, the research study of Hill et al. (Hill et al., 2007) presented an approach, called Dora, to facilitate software maintenance tasks such as feature location and impact analysis. Dora received a formulated query and the identified seed method set by the user to find the relevant neighborhood source code elements to the query. In Dora, the relevancy of each source code element to the formulated query is calculated using the combination of TF-IDF[4] scoring and method features such as the locations of relevant terms in the source code element. For example, Dora gave more value to the terms that appeared in the method name and the number of statements that contain a query term. Furthermore, Dora traversed the call graph edges from the seed set to identify more relevant source code elements to the requested query. Finally, the relevant source code elements as the "neighborhood" to the requested query were presented to the user. Generally, this study shows the way of using structural information to identify relevant source code elements.

Scanniello and Marcus (Scanniello & Marcus, 2011) also combined the textual sim-

[4]Term Frequency-Inverse Document Frequency

ilarities and the structural dependencies. The contribution of the approach of Scanniello and Marcus is clustering the source code elements based on the similarities that obtained from both the text analysis and static analysis. In this approach, the text analysis information, obtained from applying an IR model to index the source code elements, was combined with the obtained information from dependency graph[5] in order to create a new weighted graph. Then, the related source code elements of the new graph were clustered using a BorderFlow algorithm (A.-C. Ngomo, 2010; A.-C. N. Ngomo & Schumacher, 2009). Next, for a formulated query, the textual similarities of the source code elements and the similarity of each cluster with the query were computed. Finally, the similar clusters with the requested query were retrieved. The source code elements in the retrieved clusters were ranked based on the amount of similarities in a descending order.

The combination of text analysis and static analysis was also used in the study of Alhindawi et al. (Alhindawi et al., 2013) that utilized the structural information to identify a stereotype for each source code element. The stereotype includes the terms that explained the abstract role of a source code element, i.e. a method. Alhindawi et al. produced a feature location tool named StereoCode. In this tool, the stereotypes are identified automatically by analyzing structural and static information. Then, the identified stereotypes are added to the corresponding source code element as a comment. It means that the source code elements are re-documented to record the stereotypes in the source code elements. Then, an IR model, i.e. LSI, is used to index the source code elements in order to find the most related source code elements to a requested query.

Another study that leverages structural information in source code was conducted by Saha et al. (Saha et al., 2013) that dealt with the structured information retrieval based on code constructs, such as the names of classes and methods to improve bug

[5]This approach focused on static method references as the static dependencies.

localization accuracy. This study presented an automatic bug localization tool, named BLUiR built on Indri[6] (Strohman, Metzler, Turtle, & Croft, 2005). BLUiR builds an abstract syntax tree (AST) for the source code elements. Then, the source code elements were inputted to Indri for preprocessing, creating structured documents, and indexing. This study differentiates between two main parts of a reported bug, i.e. summary and description, and also distinguished between four different documents' fields, i.e. class, method, variable, and comments. Then, BLUiR performed separate searches for each of the eight different combinations (combination of query represents and document fields) and weighted the terms appeared in each of these combinations differently. Finally, the value of a document was obtained based on the summation of the calculated terms' weights across all eight searches.

Another related study that combined the text information and static information is the study of Ye et al. (Ye et al., 2014). This study introduced an adaptive ranking approach that leverages domain knowledge through functional decompositions of source code files into methods, API descriptions of library components used in the code, the bug-fixing history, and the code change history. Given a bug report, the ranking score of each source file was computed as a weighted combination of an array of features encoding domain knowledge, where the weights were trained automatically on previously solved bug reports using a learning-to-rank technique. The ranking function was defined as a weighted combination of features, where the features draw heavily on knowledge specific to the software engineering domain in order to measure relevant relationships between the bug report and the source code file. This study used project specific API documentation to connect natural language terms in the bug report with programming language constructs in the code. Furthermore, previously fixed bugs were also used as training examples for

[6]Indri is an open-source information retrieval toolkit.

the proposed ranking model in conjunction with a learning-to-rank technique.

The other interesting study that dealt with the combination of text information and static information is the study of Wang et al. (S. Wang & Lo, 2014). This study proposed a bug localization approach called AmaLgam used to locate relevant buggy files. The proposed approach dealt with version history, similar reports, and structure as the dataset. To combine these three together, AmaLgam integrates a bug prediction technique used in Google (analyzes version history), with a bug localization technique called BugLocator (analyzes similar reports from Issue Tracking system), and the state-of-the-art bug localization technique BLUiR (considers structural information for bug localization). The approach assigns weights to a source code file based on the combination of probability of the file to be buggy (computed by the bug prediction technique) and the similarity of the given bug report to the file (computed by integrating BugLocator and BLUiR).

Another type of additional information which was used to improve the text analysis process of feature location is derived from executed source code known as the dynamic analysis information. One of the studies that combined dynamic information and lexical information was conducted by Poshyvanyk et al. (Poshyvanyk et al., 2007). In this study, Poshyvanyk and his colleagues introduced a feature location approach called PROMESIR (Probabilistic Ranking of Methods based on Execution Scenarios and Information Retrieval). This approach combined two previously developed techniques for feature location including an IR-based technique, i.e. LSI (Marcus et al., 2004) and a scenario-based probabilistic ranking technique, i.e. SPR (Antoniol & Gueheneuc, 2006). In this approach, the obtained results from the LSI-based and SPR-based techniques were considered as the judgments of two independent experts, which provide their expertise to address the problem of identifying a feature. Then, the obtained results from these two experts, i.e. LSI-based and SPR-based techniques were combined using an inspiration from the recommended method by Jacobs (Jacobs, 1995).

The study of Revelle et al. (Revelle et al., 2010) also combined text analysis and dynamic analysis to improve feature location process. This study presented a feature location approach based on the idea of data fusion. Data fusion aims at combining multiple sources of information in order to obtain better results compared to the results of using the data sources individually. The approach by Revelle et al. applied data fusion on the information obtained from text analysis, execution traces, and web mining. In this approach, the source code elements that were obtained from analyzing execution traces (collected by dynamic analysis) were ranked using LSI (used for text analysis). Then, web mining algorithms, i.e. HITS[7] (Kleinberg, 1999) and PageRank[8] (Brin & Page, 2012), were used to filter the results obtained from the combination of dynamic and text analysis in order to eliminate source code elements which are irrelevant.

Furthermore, the dynamic information is used to improve feature location accuracy in the study of Le et al. (Le et al., 2015). This study combined the IR-based techniques and spectrum-based techniques in order to overcome the limitation of existing techniques. In this study a multi-modal technique was proposed that dealt with both bug reports and program spectra to locate the bug reports. The proposed technique involved three main components including AML$_{Text}$, AML$_{Spectra}$, and AML$_{SuspWord}$. The first component considered only the description appeared in bug reports which in text format. The second component, AMLSpectra only took into account the program spectra. Finally, the third component, AMLSuspWord dealt with the words which are learned from analyzing both the textual description and program spectra in the two previous components. This

[7]HITS (Hyperlinked-Induced Topic Search) basically identifies hubs and authorities rely on the links of an element to multiple relevant pages.

[8]PageRank scores elements rely on their relative importance with other elements.

component associated a program element to a set of words obtained from both the previous components. In this component, the score of a word was computed based on the frequency of appearances of the corresponding program elements in failing or correct execution traces. Each of these three components adjusted a score to each program element, and finally the total score was calculated based on the summation of these scores.

However, the use of static and dynamic information significantly improves the accuracy of feature location process, these types of information may not always be available in real-world scenarios. Furthermore, the use of static and dynamic analysis imposes considerable overhead to the system, due to extracting static and dynamic information and analyzing the large amount of data (Dit et al., 2013).

2.3.2 Information Extracted from Source Code Entities

The other set of information is also derived from the source code which is related to the properties of the source code entities or their internal correlations. This set of information are extracted from a source code that may or may not contain errors preventing it to be compliable. The study of Sisman and Kak (Sisman & Kak, 2012) is an example of the studies that deals with the overall properties of source code elements for the purpose of bug localization which is a closely related research area to feature location. Sisman and Kak believed that if a source code element recently modified meaning that it is not stable and depend on how recently it modified, the probability of being defective for the file will increase. The estimated defectiveness probability is added to the existing probabilistic IR models, i.e. DFR framework[9] and LM[10], as a coefficient to improve the accuracy of

[9]DFR (Divergence From Randomness) is an information theoretic framework that examines the relevancy of a document with a query based on the divergence of the probabilities of document feature from pure non-discriminative random distributions. This approach aims at modeling the noise in the data with simple probability distributions to specify the discriminatory features of document from the background noise.

[10]LM (Language Modeling) is a probabilistic approach that ranks the documents in a corpus based on the likelihood of relevancy of the documents to a given query.

location identification process. This study indicates that the use of information recorded in the version histories of software projects leads to improving the accuracy of location identification process. In the Sisman's study, the histories of changes occurred in a software project was used to predict the probability of being buggy for the source code elements. Two main factors are investigated for each source code element to estimate the probability of defectiveness of the element. The first factor considers the prior defect and modification probabilities for the corresponding source code element that analyzes the number of modification of the source code committed in the VCS and the number of bug fixing commits. The second factor is temporal or time decay which is time required for a file to be stabilized and bug-free after implementing a change to the file.

The study of Zhou et al. (Zhou et al., 2012) is the other related study that deals with additional information for the purpose of bug localization. They proposed a bug localization approach called BugLocator which is an IR-based approach for locating the relevant source code elements for fixing a bug. BugLocator indexed all source code elements based on their textual similarity with a reported bug using a revised Vector Space Model (rVSM). The rVSM considers the length of source code elements as well as the information about similar bug reports that have been fixed previously to identify the related source code locations to the given bug report. In general, this study assumed that the bigger source code files are most likely to be buggy. Based on this assumption, a coefficient is determined to be added to the VSM method to consider the length of source code elements for bug localization. Furthermore, rVSM deals with the previously fixed bugs as extra information to improve the accuracy of location identification process. This approach indexed source code elements in two paths. In the first path, the source code elements were indexed based on the textual similarity of the element with the reported bug by considering the length of the element. In the second path, BugLocator analyzed the similar bugs that have been fixed before in order to adjust the rankings of the relevant source code elements. Finally, the ranked lists of source code elements obtained from those two paths were combined to obtain the final ranked list of source code elements.

Furthermore, another related study was conducted by Wang et al. (J. Wang et al., 2013). In this research study, first, the Wang and his colleagues investigated the feature location process and identified this process as a human-oriented and information-intensive process. They considered the software source code space as a multi-dimensional information space that software developer need to interactively seek, browse and navigate in order to locate a query. In this information space, each dimension explains source code elements from a specific semantic or syntactic facet. The proposed approach in this study which is implemented as a web-based tool, called MFIE, is semi-automatic approach. MFIE allows developers to start with an initial query and then gradually refine and adjust the query based on the feedbacks. In each stage, the MFIE used static program analysis and data mining techniques to mines multiple facets information extracted from source code elements. The multiple facets information space includes package structure, inheritance hierarchy, usage dependencies, and intent.

The other study that dealt with the overall properties of source code elements is the study of Scanniello et al. (Scanniello et al., 2015). This study improves the feature location accuracy by considering dependencies between source code elements. The authors found that two most common kinds of information typically used in the static techniques are lexical information, such as the text recorded in source code, and structural information, such as dependencies among source code elements. The existing static techniques mainly focus on dealing with one of these kinds of information and only a few techniques deal with the combination of these two. To address this issue, this study proposed a static concept location approach based on text retrieval or search, and by the work on web document retrieval. Particularly, the proposed approach augmented text retrieval based approaches by using link analysis techniques to rank the source code elements during text retrieval. The PageRank algorithm, which is a link analysis algorithm and used in web document retrieval applications, was used in this study to provide the statistical information.

In addition to the overall properties of source code elements which are used in the studies such as of Sisman and Kak (Sisman & Kak, 2012), Zhou et al. (Zhou et al., 2012), and Wang et al. (J. Wang et al., 2013) consideration of the specific properties of source code entities improves the text analysis based feature location approaches. For instance, the study of Hill et al. (Hill et al., 2007) [11], in addition to static information, considers the place of appearance of source code entities to improve the feature location accuracy. Another example of considering the place of appearance of source code entities for the purpose of feature location is the study of Bassett and Kraft (Bassett & Kraft, 2013). In this study, Bassett and Kraft presented a lightweight feature location approach that improved the classic LDA model by combining some information obtained from a call graph. They believed that the accuracy of a LDA-based feature location approach improves by increasing the weights of the terms derived from method names or method calls. They defined 16 weighting schemes to emphasize the importance of the terms derived from the names of the methods and the names of the called methods. One extension on this study is performed by Eddy and Kraft (Eddy & Kraft, 2014) that increases the number of weighting schemes from 16 to 1024. Furthermore, in comparison to the study of Bassett and Kraft, a broader scope of variables - collected from comments, method names, body comments, parameters, and local variables - was considered and found to have a positive impact on the results of a text analysis based feature location.

On the other hand, consideration of the correlation among source code entities in

[11]This study was mentioned in Section 2.3.1

the study of Poshyvanyk and Marcus (Poshyvanyk & Marcus, 2007) improves the feature location accuracy. Poshyvanyk and Marcus combined Formal Concept Analysis (FCA) (Wille, 2005) with LSI for the purpose of feature location. Using FCA, a hierarchy of the concepts is created which includes collection of objects derived from the source code elements stored in a corpus. In this hierarchy, the objects, which share similar values for a set of properties, are grouped together in a concept. In FCA, each concept contains two main parts including the extension, which is a set of objects associated to the concept, and the intension, which is a set of attributes of the concept. The concepts which are obtained from the FCA create a concept lattice. The approach by Poshyvanyk and Marcus used LSI to provide the objects for FCA that includes a collection of source code elements and the weights of the terms appearing in the corresponding code elements as the attributes. The source code elements that appear in the concept similar to the concepts of the requested query were considered as the relevant source elements. The restriction of Poshyvanyk and Marcus approach is related to the use of FCA. In this approach, for each requested query, FCA is performed after a query and a ranked list is obtained. It means that generation of concept lattice imposes additional cost to the feature location system.

2.3.3 Information Extracted from Non-source Code Repositories

The other set of additional information which is used to improve text analysis based feature location approaches is derived from non-source code repositories such as Issue Tracking System (ITS) or mailing list. The studies of Sisman and Kak (Sisman & Kak, 2012), and Zhou et al. (Zhou et al., 2012) explained in Section 2.3.2, and the studies of Ye et al. (Ye et al., 2014), and Wang et al. (S. Wang & Lo, 2014) described in Section 2.3.1, present some examples of the feature location approaches that used the information extracted from ITS to improve the text analysis based feature location process.

On the other hand, the study of Cleary and Exton (Cleary & Exton, 2007) introduced a feature location approach that incorporated additional non-source code repositories into the classic Language Modeling (LM). This approach defined a semantic space model of relationships between terms appearing in a corpus of source code elements and then used the model when a term from a requested query is compared with the terms appearing in a source code element. If the query terms were found to have no direct correspondence with the terms of the desired source code element, the model is used to find any indirect relation term match. To create the term relations model, the terms were derived from non-source code repositories[12], such as bug reports, emails, and external documentation. This idea is used to extend the requested query by identifying a set of terms that were found in the term relation model to be potentially related to the terms in the query. Generally, this study aims to approve the importance of relevant terms in a query. Also, it shows how properly defined queries improve the text analysis process for feature location. The suggested approach in this study requires the developers to make additional natural language documents outside of the source code available to the technique.

Furthermore, the study of Moreno et al. (Moreno et al., 2014) is an example of feature location approach that combined text data with the additional information extracted from stack traces. This study presented a bug localization approach implemented as a tool referred to as LOBSTER (LOcating Bugs using Stack Traces and tExt Retrieval). In the first step of this approach, the textual similarity of the source code elements with the requested query was measured using an IR model. LOBSTER used Lucene (McCandless, Hatcher, & Gospodnetic, 2010) that combines VSM with a Boolean model to index the source code elements. Then, the obtained results from text analysis were combined with structural information. In this study, the structural information was collected from stack

[12]The use of typical techniques to create term relation model results in incorporation of the terms which may be added to the model that do not have actual relation with the source code elements' terms.

traces found in bug reports[13], and dependency graphs of software system to be used for determining the structural similarity of source code elements with the ones found in stack trace. While the buggy source code elements may not always appear in the stack trace, this study assumed that there is a structural, direct or indirect, relationship between the source code elements in stack traces and the buggy elements. LOBSTER defined a stack trace and a source code element in the software system to be similar when there is a shortest path distance between any source code element in the stack trace with the target code element in the software system through the program dependency graph. Finally, total similarity of a bug report and a source code element was obtained based on their linear combination of textual similarity and structural similarity.

The analysis of the feature location studies reviewed in this chapter indicates that in most of these studies, additional information is used besides text analysis, and not into the text analysis process. On the other hand, extraction of additional information from source code repository or non-source code repositories imposes an overhead to the feature location approach. Furthermore, none of the metadata associated to the text data were taken into account in the text analysis process.

2.4 Other Related Works

The investigation of existing feature location approaches that rely, at least in part, on the use of text analysis methods indicates that the applied methods for analyzing the text data for feature location are mostly taken from the baseline text analysis methods which are originated from a natural language context (Bassett & Kraft, 2013). The natural language context such as the newspaper articles are less structured than the text documents found in software repositories such as source code files (Bassett & Kraft, 2013). Furthermore, unlike the typical context in which these methods are applied, text data in software

[13]Only a few number of bug reports, for instance 8% of Eclipse's bug reports, contain stack trace.

repositories have a corresponding set of metadata to record the who, when and why of the changes occurred in the data (Kagdi, Maletic, & Sharif, 2007; Ratanotayanon et al., 2010). The investigation of the literature shows that efforts to improve the text analysis based feature location approaches have been mainly focused on the use of additional information such as static and dynamic information with text information. This investigation indicates that none of the metadata associated to text data were taken into account in the text analysis process.

As mentioned in Chapter 1, the incorporation of the text data with the related metadata such as time and developer is considered in this thesis as one of the important properties of the repository's text data. Incorporation of the terms with metadata, i.e. time and developer, provides the ability of analyzing the data from different aspects depending on the desired metadata. To find the related studies that deal with the metadata of time and developer associated to the text data, the feature location literature was further investigated. The investigation of the literature indicates that the incorporation of the terms with the metadata is not properly addressed in feature location literature. The only related study is the work of Sisman and Kak (Sisman & Kak, 2012) that dealt with time to predict the defectiveness probability of the source code files. This study was reviewed in Section 2.3.2.

On the other hand, the size of the text data recorded in the repository is larger than that of the typical text documents in the natural language context due to the storage of the history of source code changes (Kagdi, Collard, & Maletic, 2007). Thus, the feature location literature was investigated again to find the related studies that used nouns for feature location and no related study was found. Therefore, the other related research areas were investigated to find related studies that deal with the use of noun terms.

As mentioned previously, the other important parameter in this study is developer-metadata. With the developer-metadata, the data recorded in the repository is able to be analyzed from the aspect of developer who recorded the data in the source code. The investigation of feature location literature indicates that all the existing feature location approaches analyze the data recorded in the repository from the aspect of the location, i.e. file, class, or method, where the data are contained. This means that there is no feature location approach which analyzes the repository's data from the aspect of developer or the aspect of time. Further investigation of the literature also shows that there is still no study that deals with the developer-metadata for feature location. Therefore, the other related research areas were considered to find related studies.

2.4.1 Developer-Metadata Consideration

According to the literature, the analysis of developers' information is more common in bug assignment research area. Bug assignment is the task of assigning a bug or defect, reported to the software project, to one of the project developers who is most likely to be able to address and fix the bug report[14] (Anvik & Murphy, 2007). Many automatic and semi-automatic bug assignment approaches have been proposed to facilitate the bug assignment process (Anvik, 2006; Anvik, Hiew, & Murphy, 2006; Kagdi et al., 2012; Shokripour, Anvik, Kasirun, & Zamani, 2013; Shokripour et al., 2014). The existing bug assignment approaches are classified into two categories (Shokripour et al., 2013), i.e. activity-based and location-based.

The location-based bug assignment approaches applied the inverse of the idea of using developer information in this thesis. These bug assignment approaches used the location information for developer recommendation. This means that they recommended the appropriate developers for bug fixing through predicting the source code locations related to the reported bug (Hossen, Kagdi, & Poshyvanyk, 2014; Kagdi et al., 2012;

[14]In software systems, a bug is an error, fault, or defect in the software program that leads the software to produce an unexpected or incorrect result. In well-developed software systems, the bugs are reported to a bug tracking system to be recorded and keep tracking.

Servant & Jones, 2012; Shokripour et al., 2013, 2014). The location-based bug assignment approaches typically include two phases to first predict the related locations and then recommend the fixer[15] based on collected information from the predicted locations. However, in the research reported in this thesis, the data recorded in the source code is analyzed from the aspect of developer who recorded the data. This means that developer information is used to classify and categorize the data recorded in the source code. Appendix D introduces the structure of the typical location-based bug assignment approaches through some examples of the existing location-based bug assignment approaches.

On the other hand, the developer information is also used to improve the accuracy of traceability link recovery. Traceability link recovery is another related research area to feature location (Dit et al., 2013). Traceability aims to connect different types of software repositories, while feature location is more concerned with identifying the source code file pertinent to a software feature, and not specific parts of a document. Many traceability link recovery approaches have been proposed to automate the link recovering process (Briand, Labiche, & Yue, 2009; Mader & Gotel, 2012; Mader, Gotel, & Philippow, 2008; Omoronyia, Sindre, & Stå, 2011), with IR being one of the most commonly used to facilitate the tracing process (Antoniol, Canfora, Casazza, De Lucia, & Merlo, 2002; Marcus et al., 2004).

The study of Diaz et al. (Diaz et al., 2013) improved the accuracy of IR-based traceability methods by leveraging the developer information specially the ownership[16]of the code. As mentioned in Chapter 1, project developers generally work on specific and related functionalities during software evolution. Based on this issue, Diaz and his colleagues believed that the code elements (e.g., classes) which were authored by a specific

[15]Fixer is the project developer who is able to fix the reported bug.

[16]Author of the code

project developer may be related to specific high-level artifacts (e.g., use cases). This means that if a developer has the ownership of a code element linked to a high-level artifact (e.g., a use case), then other code elements which were authored by the desired ownership would likely be related to the same artifact. Based on this assumption, they introduced a traceability recovery method, called TYRION[17]. The TYRION method has four steps: (i) computing textual similarity between high-level artifacts and source code elements, (ii) identifying the ownership of each source code elements, (iii) defining the ownership context, and (iv) integrating code ownership information with textual information.

2.4.2 Noun Usage

As mentioned in Chapter 1, one of the characteristics of text data recorded in the software repository is recording the history of changes occurred to the data, i.e. source code (Kagdi, Collard, & Maletic, 2007). Due to the storage of the history of changes, the size of the text documents recorded in the repository is larger than that of the typical text documents in the natural language context. The language of the text which is recorded in software documents, e.g. software repositories, is categorized under the technical language (Capobianco et al., 2012). The technical language is referred to the language which is used by different people with common interest or working on a particular area (Jurafsky & Martin, 2014). In this kind of language, noun type of terms provides more information on the semantics of a document and the verb type of terms has a generic semantics and mainly plays a connection role (Zou, Settimi, & Cleland-Huang, 2006, 2008). Thus, it is reasonable to say that the use of only the nouns instead of using all types of terms impact the accuracy of MSR activities. The investigation of feature location literature indicates that there is no feature location approach that used only the nouns to identify source code location pertinent to a software feature. Thus, the use of noun terms

[17]TraceabilitY link Recovery using Information retrieval and code OwNership

in the other related research areas such as traceability link recovery and bug assignment was investigated.

The noun terms has been used by Capobianco et al. (Capobianco, De Lucia, Oliveto, Panichella, & Panichella, 2009; Capobianco et al., 2012) in the area of traceability recovery. The study of Capobianco and his colleagues dealt with the noun terms as a simple dimension reduction method to improve the performance of traceability recovery. They demonstrated that the use of only nouns reduces amount of noisy data stored in various types of software repositories and consequently, reduces the number of false positives retrieved by IR methods. To filter the nouns, first, they preprocessed the data collected from the software repositories and used Part-of-speech (POS) tagger to tag all the terms in order to specify the grammatical nature of the terms such as verb, noun, and adjective. From the tagged terms, only the nouns were used for traceability recovery. All the extracted nouns were used to build the corpus without applying the stop word function and/or the stop word list. Furthermore, they did not use any morphological analysis such as stemming or lemmatizing.

Capobianco and his colleagues extensively evaluated their approach on different IR methods, i.e. the Jenson-Shannon (JS) method (Abadi, Nisenson, & Simionovici, 2008) as well as VSM and LSI, to examine the generalizability of their approach. They conducted a set of experiments on the software repositories collected from five different software projects of various areas, i.e. EasyClinic, eTour, Pine, MODIS, and CM1. The results of the experiments demonstrated a significant improvement on the accuracy of traceability recovery methods as well as the reduction of the size of dataset.

The other research area related to feature location that used only the noun terms is bug assignment. As mentioned above, bug assignment aims at assigning the reported bug to the proper developer for fixing. Shokripour and his colleagues initiated the idea of using the nouns for bug assignment (Shokripour et al., 2013). They examined the impacts of using only the noun terms in both the activity-based and location-based bug assignment approaches. The conducted experimental evaluation showed that the use of nouns significantly reduces the size of the dataset that needs to be analyzed for bug assignment. The results of the experiments indicated that the use of noun provides enough information for bug assignment. The use of nouns makes a small improvement in bug assignment accuracy which is not a significant improvement (the improvement is between 2% to 9%).

2.5 Summary: Inferences from Literature Reviewed

In this chapter, the feature location literature was investigated to find the related papers to the research reported in this thesis. According to the recent survey on feature location (Dit et al., 2013), the existing feature location approaches are classified into three categories based on the way of analyzing the data recorded in software repositories. These categories include dynamic, static, and text analyses. Dynamic analysis is based on the program execution behavior, while static analysis deals with the static properties of software project like class structure, dependency graphs and control/data flow dependencies. The last category of data analysis methods in feature location is text analysis that investigates the text data recorded in the historical project repositories and analyzes them to extract useful information. According to the research of Dit and his colleagues (Dit et al., 2013), text analysis is the most commonly used category of data analysis method in feature location. Furthermore, text analysis process is fundamental in most of feature location approaches and it is used as a part of other feature location approaches that used dynamic analysis and static analysis. It means that the accuracy of text analysis process not only affects the accuracy of feature location approaches that used text analysis individually, but also impacts the ones using dynamic and static analysis beside the text analysis. Thus, as mentioned in Section 1.6, text analysis was selected as the scope of this research to enhance and improve feature location process.

The investigation of the feature location literature indicates that the applied methods for text analysis originate from a natural language context that have some differences with the text data found in software repositories. Unlike the typical context in which these methods are applied, text data in software repositories incorporates with a set of metadata such as time and developer. Furthermore, due to the storage of the history of changes, the size of the text documents recorded in the repositories is larger than that of the typical text documents in the natural language context. Review of the literature indicates that the existing feature location approaches which are based on text analysis do not address these differences within the text analysis process. Thus, *the feature location approaches that rely on the use of text analysis methods do not utilize all possible potential for accurately locating the software features.*

The efforts of the researchers on improving the accuracy of text analysis based feature location approaches mainly focus on the use of additional information besides the text data. Figure 2.2 summarizes different types of additional information used to improve text analysis based feature location approaches. Furthermore, Table 2.6 completes the information presented in Figure 2.2 with examples.

The additional information used to improve text analysis based feature location approaches is classified in two categories based on the software repository which is used to provide the additional information. The first category is the information derived from the source code repository including information extracted from compiled or executed source code, and the information extracted from the properties of the source code entities. The other category is derived from non-source code repositories such as ITS.

As shown in Figure 2.2, the static information which is used to improve the text

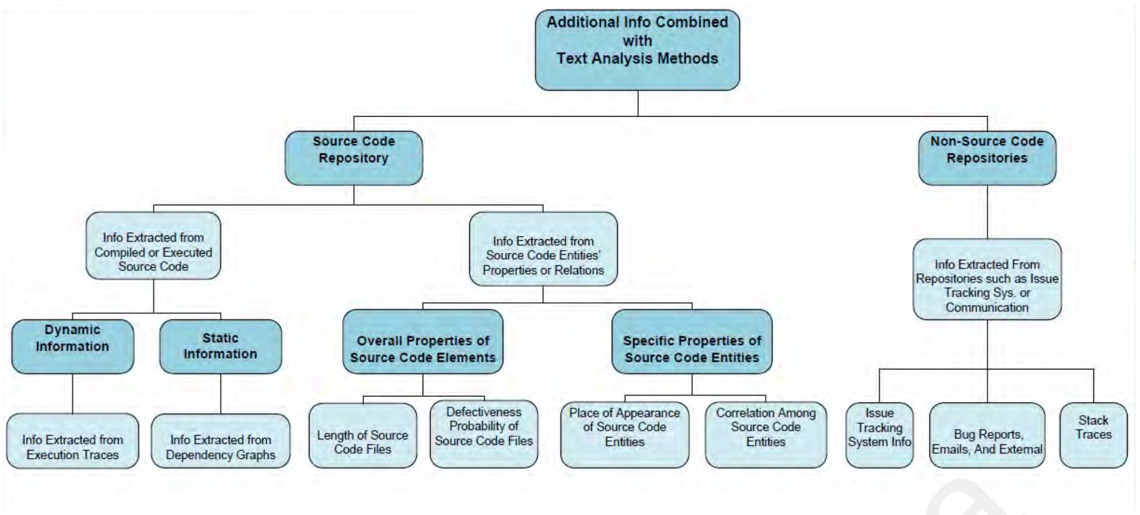

Additional Info Combined with Text Analysis Methods
Source Code Repository
Non-Source Code Repositories
Info Extracted from Compiled or Executed Source Code
Info Extracted from Source Code Entities' Properties or Relations
Info Extracted From Repositories such as Issue Tracking Sys. or Communication
Dynamic Information
Static Information
Overall Properties of Source Code Elements
Specific Properties of Source Code Entities
Info Extracted from Execution Traces
Info Extracted from Dependency Graphs
Length of Source Code Files
Defectiveness Probability of Source Code Files
Place of Appearance of Source Code Entities
Correlation Among Source Code Entities
Issue Tracking System Info
Bug Reports, Emails, And External
Stack Traces

Figure 2.2: Chart of additional information that combined with text data to improve feature location accuracy

Table 2.6: Additional information that combined with text data to improve feature location accuracy

Type of Additional Information	**Source Code Repository**	Information Extracted from Compiled or Executed Source Code	Dynamic Information	Information Extracted From Execution Traces	(Poshyvanyk et al., 2007) (Revelle et al., 2010)
			Static Information	Information Extracted From Dependency Graphs	(Zhao et al., 2006) (Hill et al., 2007) (Scanniello & Marcus, 2011)
		Information Extracted from Source Code Entities' Properties or Relations	Overall Properties of Source Code Elements	Length of Source Code Files	(Zhou et al., 2012)
				Defectiveness Probability of Source Code Files	(Sisman & Kak, 2012)
			Specific Properties of Source Code Entities	Place of Appearance of Source Code Entities	(Hill et al., 2007) (Bassett & Kraft, 2013) (Eddy & Kraft, 2014)
				Correlation Among Source Code Entities	(Poshyvanyk & Marcus, 2007)
	Non-Source Code Repositories	Information Extracted From Repositories such as Issue Tracking Sys. or Communication		Issue Tracking System (ITS) Information	(Sisman & Kak, 2012) (Zhou et al., 2012)
				Bug Reports, Emails, And External Documentation	(Cleary & Exton, 2007)
				Stack Traces	(Moreno et al., 2014)

analysis based feature location approaches is extracted from compiled source code (Hill et al., 2007; Saha et al., 2013; Scanniello & Marcus, 2011; Zhao et al., 2006). The other type of additional information used to improve feature location accuracy is dynamic information which is collected from execution traces of the software project (Poshyvanyk et al., 2007; Revelle et al., 2010). Even though the combination of text data with these types of additional information have made a significant improvement in the accuracy of feature location, obtaining such types of additional information brings a significant overhead, as they require instrumenting and executing the software, and also analyzing large amounts of data. Furthermore, improving the accuracy of text analysis process individually impacts the overall accuracy of feature location process that used text data separately or in combination with the additional information.

The next type of additional information does not need the compilable nor executable source code. This type of additional information focuses on the overall or specific properties of source code entities. According to the literature, a few text analysis based feature location approaches dealt with the properties of the repository's text data in order to improve the feature location accuracy. A number of existing feature location approaches focused on the overall properties of the source code elements like the source code files' length (Zhou et al., 2012) and the probability of defectiveness of the files (Sisman & Kak, 2012). These studies mostly rooted in bug localization research area that aims at locating the defects and bugs in the source code and accordingly dealt with the source code files' properties to rank the files with higher defectiveness probability in the earlier locations in the retrieved list. Moreover, a number of feature location approaches dealt with specific properties of source code entities such as the place of appearance of terms in the source code as an effective property to improve the accuracy of feature location process (Bassett & Kraft, 2013; Eddy & Kraft, 2014; Hill et al., 2007). In addition to the specific properties of the source code entities, consideration of the correlation between the source code entities improves the accuracy of feature location process (Poshyvanyk & Marcus, 2007).

As shown in Figure 2.2, the other set of additional information is collected from the software repositories other than the source code repository. For instance, the information collected from Issue Tracking System (ITS) was used in the studies of Sisman and Kak

(Sisman & Kak, 2012) and Zhou et al. (Zhou et al., 2012) to provide information on the defectiveness of the source code files. Furthermore, the terms derived from non-source code data, such as bug reports, emails, and external documentation were used in the study of Cleary et al. (Cleary & Exton, 2007) to create the term relations model. Moreover, the information extracted from stack traces recorded in ITS for bug reports was used in the study of Moreno et al. (Moreno et al., 2014) to enhance the location identification process.

As mentioned in Chapter 1, one of the important properties of data recorded in the repository is incorporation of the data with metadata such as time and developer. Furthermore, since the history of the changes of the source code document is recorded in the corresponding document, the size of source code documents is larger than that of the natural language documents. The investigation of the literature indicates that these specific properties of repository's text data have not been addressed in the existing text analysis based feature location approaches. This investigation results in addressing the first research objective (Objective 1) that aims to study the existing text analysis methods applied for feature location and identify the current problems in the existing text analysis methods for feature location. Lacks of addressing these properties leads to deficiency of feature location process in utilizing all possible potential of text analysis for accurately locating the software features. On the other hand, due to the availability of these properties, i.e. time and developer metadata, in the source code, there is no need to extract any further information from the source code that imposes an overhead to the feature location approach. Consideration of these properties in text analysis process leads to improving the overall accuracy of feature location. Moreover, compared with the existing enhancing strategies described above, consideration of these properties of repository's text data can be used to complement the existing feature location approaches.

CHAPTER 3: RESEARCH METHODOLOGY

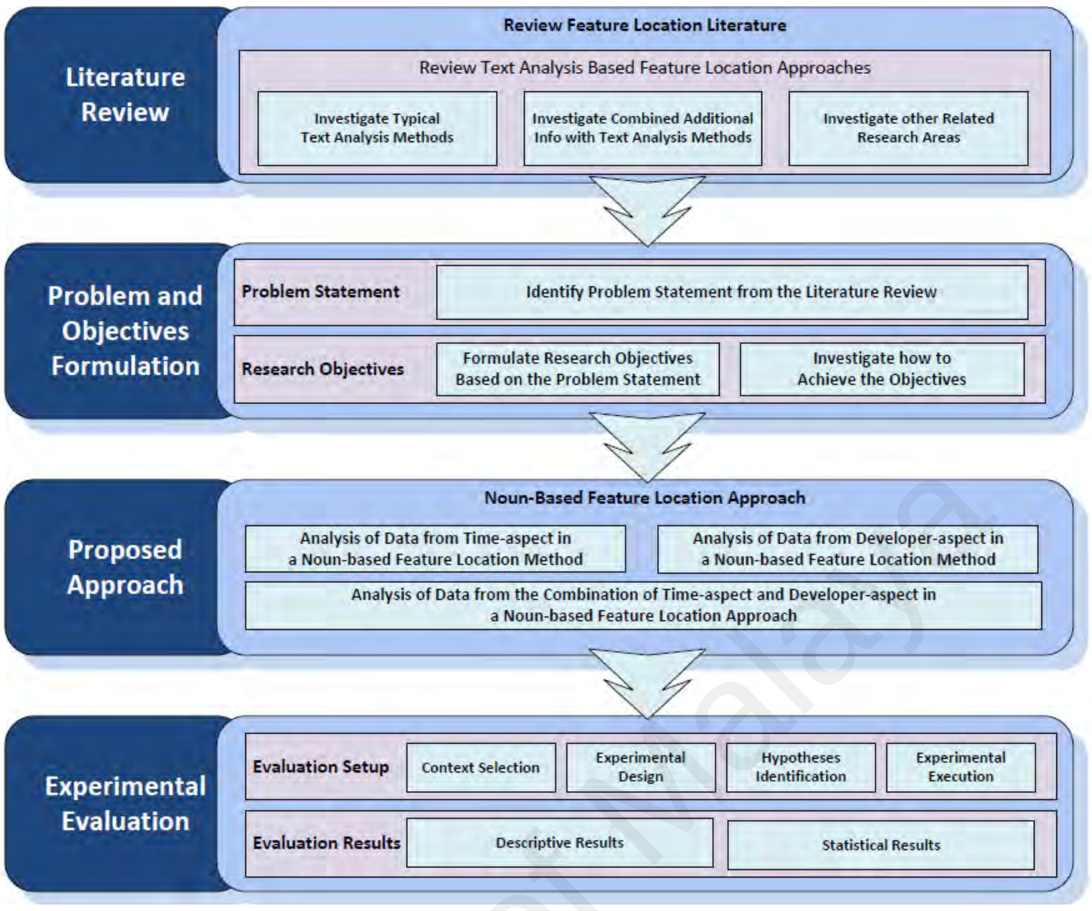
In this chapter, the research methodology applied in this research is explained. As shown in Figure 3.1, the research methodology includes four main steps: (i) review of the existing feature location approaches, (ii) identification of the problem statement and formulation of the research objectives, (iii) proposing a feature location approach, and (iv) the experimental evaluation of the proposed approach. The details of the research methodology steps are presented in the following sections of this chapter.

3.1 Literature Review

To perform this research, first of all, scholarly digital libraries, particularly IEEE, ScienceDirect, and Web of Science were investigated to identify related papers on feature location research studies. Furthermore, the papers published in related conferences such as the International Conference on Software Engineering (ICSE), Mining Software Repositories (MSR), Program Comprehension (ICPC), and Software Maintenance (ICSM) were reviewed. Figure 3.2 shows the overview of the focused topics in reviewing the related papers in the literature review step of this research.

According to the recent survey on feature location (Dit et al., 2013), the existing feature location approaches are classified in three categories including dynamic analysis, static analysis and text analysis, depending on the techniques that they used to analyze the data. The most commonly used analysis technique in feature location is text analysis. As shown in Figure 3.2, in order to utilize the effort spent on reviewing the feature location literature, the researcher mainly focused on the feature location papers that used text analysis as the main part of their proposed approach.

The investigation of text analysis based feature location literature indicates that the applied text analysis methods for feature location are originated from the natural language



Literature Review
Review Feature Location Literature
Review Text Analysis Based Feature Location Approaches
Investigate Typical Text Analysis Methods
Investigate Combined Additional Info with Text Analysis Methods
Investigate other Related Research Areas
Problem and Objectives Formulation
Problem Statement
Identify Problem Statement from the Literature Review
Research Objectives
Formulate Research Objectives Based on the Problem Statement
Investigate how to Achieve the Objectives
Proposed Approach
Noun-Based Feature Location Approach
Analysis of Data from Time-aspect in a Noun-based Feature Location Method
Analysis of Data from Developer-aspect in a Noun-based Feature Location Method
Analysis of Data from the Combination of Time-aspect and Developer-aspect in a Noun-based Feature Location Approach
Experimental Evaluation
Evaluation Setup
Context Selection
Experimental Design
Hypotheses Identification
Experimental Execution
Evaluation Results
Descriptive Results
Statistical Results

Figure 3.1: Overview of research methodology

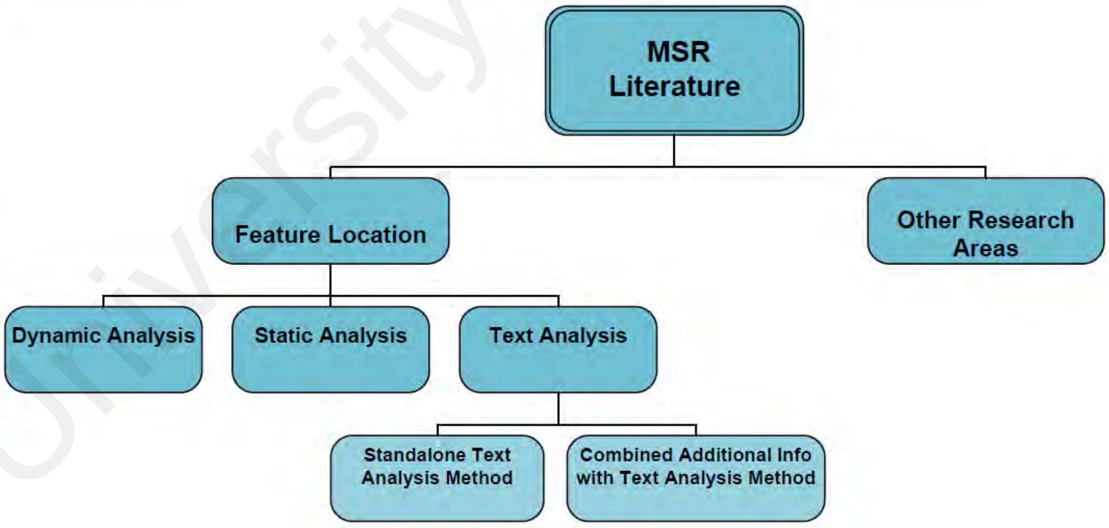


Figure 3.2: Overall topics of literature review

context. Accordingly, the text analysis methods that applied for feature location as well as the additional techniques and information which were used to improve the accuracy of feature location process were investigated. This investigation provides a better understanding of the use of text analysis for feature location and helps the researcher in identifying the research gaps.

3.2 Problem Statement and Objectives Formulation

The investigation of feature location literature and the applied text analysis methods reveals the lack of considering the characteristics of the text data recorded in the software repository compared with the text data in natural language context (Bassett & Kraft, 2013). Lack of considering the specific characteristics of repository's text data, i.e. (i) incorporation of the metadata with the text data, and (ii) larger dataset size, in analyzing the data leads to *low accuracy of text analysis methods applied for feature location.*

After identifying the current gap in feature location research area, in the next step, the researcher defined to what extent the existing text analysis methods can overcome weaknesses. Accordingly, a set of objectives are formulated as follows to address the main goal of this thesis which is improving the accuracy of feature location by considering the specific characteristics of the text data recorded in the repository.

- **Objective 1:** To study the existing text analysis methods applied for feature location and identify the current problems in the existing text analysis methods for feature location.

- **Objective 2:** To propose a feature location approach that considers the differences between the text data recorded in the software repository and the text data in the natural language context to make an improvement in text analysis process for feature location.

  - **Objective 2.1**: To improve the accuracy of feature location by analyzing the text data recorded in the repository from the aspect of time at which the data was recorded in the repository.

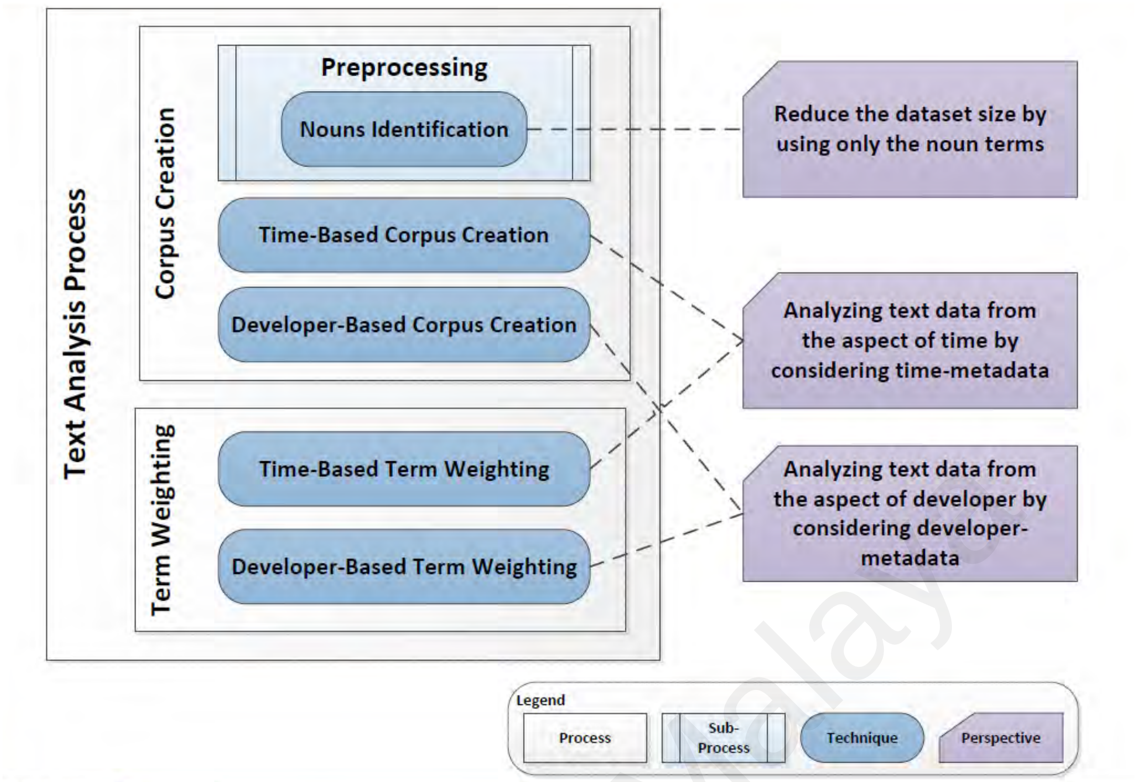
  - **Objective 2.2**: To improve the accuracy of feature location by analyzing the text data recorded in the repository from the aspect of the developer who recorded the data in the repository.
  - **Objective 2.3**: To reduce the size of dataset used for feature location.

- **Objective 3:** To evaluate the impacts of considering the specific characteristics of the text data recorded in the repository within the text analysis process of the proposed feature location approach.

3.3 Proposed Approach

The formulated objectives are addressed in three perspectives, i.e. (i) analysis of the text data from the aspect of time at which the data was recorded in the repository, (ii) analysis of the text data from the aspect of developers who recorded the data in the repository, (iii) reducing dataset size by using only the noun terms that exist in the text data recorded in the repository. To address the three perspectives, the text analysis process was investigated to define to what extent the existing text analysis steps can satisfy the identified perspectives. Accordingly, a set of text analysis steps are identified as suitable steps to consider the perspectives in the text analysis process. Figure 3.3 shows the text analysis steps that were found as the suitable steps to address the perspectives as well as the required techniques to apply the perspectives in the text analysis process.

As shown in Figure 3.3, the preprocessing step is used to refine the text data and reduce the size of the dataset. As Manning and his colleagues showed, preprocessing, which is a sub-step of corpus creation, has a great potential to reduce the size of dataset and also to filter the data to result in less noises (Manning et al., 2008). In this research, the text data are preprocessed to only involve the noun terms of the text data. To select only the nouns, the text stored in the repository is tokenized and categorized. Then, the



Text Analysis Process
Corpus Creation
Preprocessing
Nouns Identification
Time-Based Corpus Creation
Developer-Based Corpus Creation
Term Weighting
Time-Based Term Weighting
Developer-Based Term Weighting
Reduce the dataset size by using only the noun terms
Analyzing text data from the aspect of time by considering time-metadata
Analyzing text data from the aspect of developer by considering developer-metadata
Legend
Process
Sub-Process
Technique
Perspective

Figure 3.3: Modified parts of a typical text analysis process and their corresponding techniques

terms that are recognized as the noun type are selected to participate in the rest of the text analysis process.

To analyze the text data from the aspect of time, two steps of corpus creation and term weighting are found as the suitable steps. To provide the ability of analyzing the data from the aspect of time, the preprocessed data are associated with the corresponding time-metadata to create a time-based corpus. The text data stored in the time-based corpus is used to calculate the similarity of the context of a source code file with a given change request. To determine the similarity of the file with the given change request, the term weighting step was found as a suitable step to consider time of creation or modification of the text data. The term weighting step adjusts the values of the terms appeared in a source code file. The typical term-weighting techniques such as TF and TF-IDF[1], which are used for text analysis, are briefly explained in Appendix C. These techniques weight the

[1]Term Frequency-inverse Document Frequency

terms based on the statistical computations. However, consideration of time of creation or modification of terms enhance the term weighting process. In analyzing the data from the aspect of time, the term‘s weight is determined over time of creation or modification of the term. It means that the value of the term is calculated among the other terms which were created at the same time. Accordingly, in this perspective, the terms are weighted using a time-based term-weighting technique that determines the values of the terms over time based on both the statistical computation and the time of usage.

Similar to analysis of the data from the aspect of time, two text analysis steps of corpus creation and term weighting are considered to provide the ability of analyzing the text data from the aspect of developers. In the corpus creation step, the preprocessed text data are associated with the corresponding developer-metadata as well as the time-metadata[2] to create a developer-based corpus. A document in the developer-based corpus is a source code file that classified in the developer expertise profiles and associated with the time-metadata. To calculate the similarity of the context of a source code file with a given change request, the term is weighted using a developer-based term-weighting technique. This technique determines the value of a term in the developer profiles. It means that the value of a term is calculated based on the developer who used the term in the file and time when the developer used the term.

Consideration of the identified perspectives in the text analysis process results in proposing two feature location methods that were finally combined to embody a feature location approach[3]. Figure 3.4 shows the relationship between the perspectives and the

[2]In this corpus, the data is also associated with the time-metadata to be aware of time of creation or modification of the data by the developer.

[3]The difference between these two terms, “Method” and “Approach”, is very small. In general, a method or an approach refer to a regular and systematic way of accomplishing something. However, a method may contain a set of techniques to perform a desired process and an approach may embody by a set of methods to address the desired process. Accordingly, the combination of the two proposed methods is referred to as an approach.

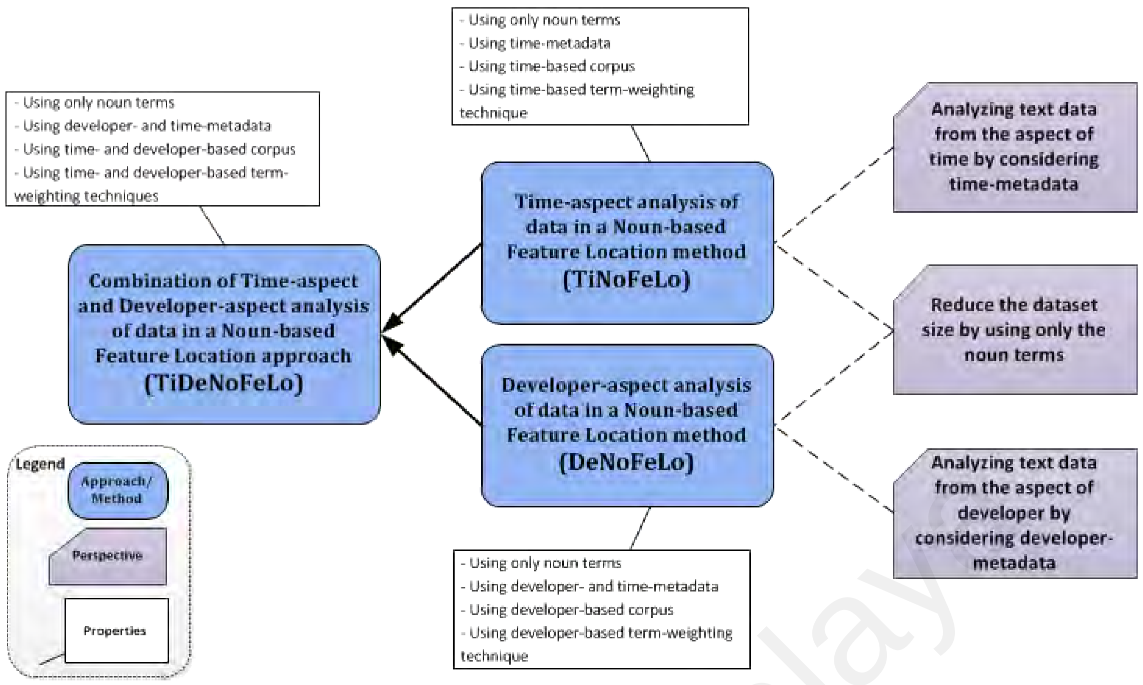

- Using only noun terms
- Using developer- and time-metadata
- Using time- and developer-based corpus
- Using time- and developer-based term-weighting techniques
- Using only noun terms
- Using time-metadata
- Using time-based corpus
- Using time-based term-weighting technique
Analyzing text data from the aspect of time by considering time-metadata
Combination of Time-aspect and Developer-aspect analysis of data in a Noun-based Feature Location approach (TiDeNoFeLo)
Time-aspect analysis of data in a Noun-based Feature Location method (TiNoFeLo)
Reduce the dataset size by using only the noun terms
Developer-aspect analysis of data in a Noun-based Feature Location method (DeNoFeLo)
Analyzing text data from the aspect of developer by considering developer-metadata
Legend
Approach/ Method
Perspective
Properties
- Using only noun terms
- Using developer- and time-metadata
- Using developer-based corpus
- Using developer-based term-weighting technique

Figure 3.4: Proposed Approach in relation with the proposed methods and the identified perspectives

proposed methods and the proposed approach.

The first proposed method identifies the related source code files to the given change request by analyzing the data from the aspect of time when the data was recorded in the source code files. This method uses a time-based corpus that contains the nouns extracted from the source code files as the dataset and weights the noun terms recorded in the dataset using a time-based term-weighting technique. The time-based technique weights the terms over time. This proposed method is referred as Time-aspect analysis of data in a Noun-based Feature Location method (***Ti***NoFeLo).

The second proposed method analyzes the data stored in the repository from the aspect of developer who stored the data in the source code files. This method uses a developer-based corpus that contains the nouns appeared in the source code files as the dataset and weights the noun terms using a developer-based term-weighting technique. The developer-based term-weighting used the latest time of usage of the noun term. This proposed method is referred as Developer-aspect analysis of data in a Noun-based Feature

Location method (***De***NoFeLo).

The combination of Time-aspect and Developer-aspect analysis of data is used to propose a Noun-based Feature Location approach (***TiDe***NoFeLo). This approach used a time and developer-based corpus that contain nouns incorporated with time and developer metadata. In this approach, nouns are weighted using both the time-based and developer-based term-weighting techniques. It means that each noun has two weights. The combination of these two weights results in calculation of total weight of the noun [4].

3.4 Experimental Evaluation

The proposed methods and the proposed approach are evaluated to assess the impact of considering the identified perspectives in text analysis process of feature location. One typical methodology to evaluate an approach in software engineering research area is experimental evaluation. One of the well-known guidelines for conducting experimentation in software engineering is provided by Wohlin et al. (Wohlin et al., 2012). Since this guideline is used by several researchers in mining software repositories research area (De Lucia, Oliveto, & Tortora, 2009; Scanniello & Marcus, 2011), this research follows the same organization for empirical evaluation of software systems recommended by Wohlin et al. (Wohlin et al., 2012). To conduct an experimental evaluation in this research, a set of settings, known as experimental evaluation setup, needs to be specified.

Figure 3.5 shows steps of the experimental evaluation setup including: (i) context

[4]In this thesis, two expression of “time-based” and “time-aware” are used. The “time-based” expression is used in the first proposed term-weighing technique that focuses on the use of time-metadata and determines weight of terms based on the time when the term was used. The “time-aware” expression is used in the second proposed technique, developer-based technique, that weights the terms based on the developer who used the term. The developer-based term-weighting technique focuses on the use of developer-metadata and augmenting the time-metadata. Since the difference between the use of these two expression (time-based and time-aware) is very small, the researcher used these two expressions to differentiate between the use of time in the two proposed weighting techniques. Accordingly, the “time-based” expression is used for the first term-weighting technique to emphasize on the use of the time and the “time-aware” expression is used for the second term-weighting technique to de-emphasize on the use of time since it focuses on the use of developer information. In the proposed approach, both the time-based and developer-based techniques are used to weight the terms. Accordingly, to cover both of the techniques used in this approach, the “time-aware” expression is used in the title of this thesis.

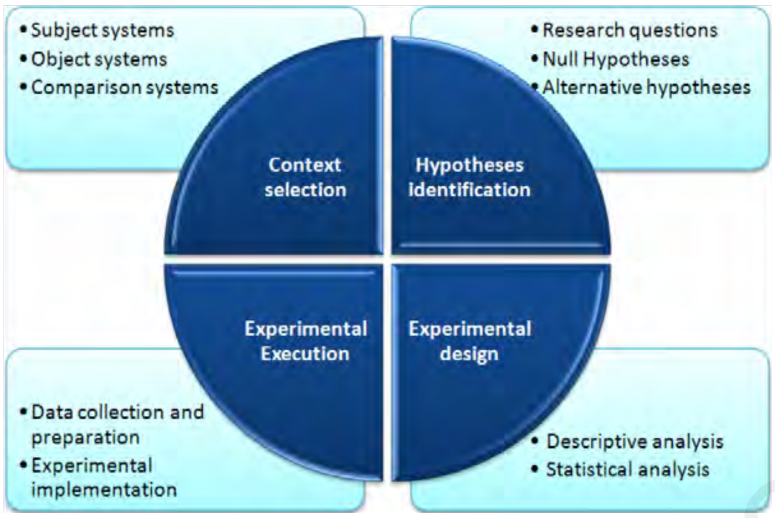


Figure 3.5: Evaluation setup

selection, (ii) experimental design, (iii) hypotheses identification, and (iv) experimental execution. The evaluation context deals with identifying the subject systems, object systems and baseline feature location approaches for evaluating the proposed methods and the proposed approach. Then, the experimental design explains the descriptive and statistical analysis setups. Next, in hypotheses identification step, a set of research questions and the corresponding hypotheses are formulated along with the main research question identified in Chapter 3.1. Finally, how the data was collected from the subject systems and preprocessed as well as the implementation of the experiments is explained in experimental execution step of the experimental evaluation. The details of the evaluation setup are explained in Chapter 5. According to the evaluation setup settings, the required experiments are conducted on the proposed methods and the proposed approach. The obtained results from the experiments are assessed from two aspects, i.e. descriptive results and statistic results.

To summarize, the identified research gap in this research study is addressed by proposing two feature location methods. These two proposed methods embody a feature location approach that considers the specific characteristics of repository‘s text data in text analysis process. The details of the proposed methods and the proposed approach are presented in Chapter 4. Furthermore, the proposed methods and approach are experimentally evaluated based on the evaluation setup which is explained in Chapter 5. The results of the experimentations are reported and analyzed in Chapter 6.

CHAPTER 4: PROPOSED APPROACH

To address the identified problem statement including the lack of utilizing the possible potential of text analysis methods for analyzing the repository's text data for feature location, two feature location methods, i.e. ***Ti***NoFeLo and ***De***NoFeLo, are proposed. ***Ti***NoFeLo stands for *Ti*me-aspect analysis of data in a *No*un-based *Fe*ature *Lo*cation method, whereas ***De***NoFeLo stands for *De*veloper-aspect analysis of data in a *No*un-based *Fe*ature *Lo*cation method. In these two methods, the noun terms stored in the dataset are analyzed from two different aspects of time and developer. In relation with these two proposed methods, a new feature location approach, called ***TiDe***NoFeLo[1], is proposed which is the combination of *Ti*me-aspect and *De*veloper-aspect analysis of data in a *No*un-based *Fe*ature *Lo*cation approach. The proposed approach analyzes the data from both aspects of time and developers to locate the features in the source code. The proposed methods and the proposed approach collect the required data from the Version Control System (VCS) of the software projects and follow a set of functional components to identify the related source code locations to a change request.

Accordingly, in this chapter, first of all, an overall view of the proposed methods and the proposed approach is explained in Section 4.1. Then, the details of the functional components that embody ***Ti***NoFeLo and ***De***NoFeLo methods are presented in Sections 4.2 and 4.3, respectively. Next, Section 4.4 describes the details of the ***TiDe***NoFeLo approach, which is the combination of the two proposed methods. Finally, a summary of this chapter is given in Section 4.5.

[1]***Ti***NoFeLo ***De***NoFeLo ***TiDe***NoFeLo

4.1 Overview

As mentioned in Chapter 1, the main goal of this thesis is to improve the feature location accuracy by considering the specific characteristics of the repository's text data, i.e. incorporation of the metadata of time and developer, and larger dataset size, in text analysis process. To address these characteristics in text analysis process of feature location, three perspectives were identified that deal with, (i) the analysis of data from the aspect of time, (ii) the analysis of data from the aspect of developer, and (iii) the use of only the noun terms to reduce the dataset size. For the rest of this section, first, the notion and rationales of identifying these perspectives are explained in detail. Then, the features and functional components of the proposed methods and approach are described briefly. The complete descriptions of the proposed methods and approach are presented in Sections 4.2, 4.3 and 4.4.

4.1.1 Perspectives

As explained in Chapter 1, in this thesis, three perspectives are identified to improve the text analysis process of feature location, i.e. the analysis of data from the aspect of time, the analysis of data from the aspect of developer, and the use of only noun terms to reduce the dataset size. In this section, after briefly explaining these perspectives, the rationales of identifying these perspectives are presented in detail.

4.1.1.1 Time-aspect analysis of data

Analysis of the data from the aspect of time provides the ability of dealing with the importance of the data overtime when the data was recorded in the repository. In other words, in the time-aspect analysis of data, the value of the data is determined around the context in which it was recorded at the same time. It is anticipated that analysis of the data from the aspect of time leads to more accurately locating a change request in the source code. To facilitate the analysis of data from the aspect of time, the data collected from the software repository is associated with their corresponding time-metadata.

Consideration of time-metadata in analyzing the text data for feature location is based on the assumption that for a new change request, the source code entities with the highest textual similarity that were more recently modified have more relevancy with the change request. This assumption is based on two principles, (i) defect localization, and (ii) software evolution. Defect localization refers to the fact that more recent modifications to a project are more likely the cause of future bugs or defects (Hassan & Holt, 2005; Zimmermann et al., 2005). By considering recent modifications in the source code, it leads to finding relevant locations that are the cause of a new change request (Sisman & Kak, 2012). On the other hand, software evolution deals with the different goals and requirements that a software project follows in different periods of its lifecycle (Gómez et al., 2009). For a given change request, the requested modification to the source code in the same time period of the project's lifecycle will likely have the same goals. This assumption is further elaborated in the following paragraphs.

In every period of a project's lifecycle, the terms that are used across different repositories, such as source code version and issue tracking, are consistent with the requirements of the project during that specific time period (Gómez et al., 2009). Change requests occurring in a different time period of the project's life may have different goals. For instance, the change requests which are reported in the initial period of the project life are usually focused on the fundamental requirements of the project. A common way of addressing this type of change requests is to create new file(s) or make extensive modifications to existing files. Consequently, it is common to have a large number of modified files resulting from this set of change requests, and the changes that are made are correspondingly extensive.

This claim is supported by the systems used in this study. For example, in the first working day of the JDT[2] project, around 490 files were modified in 490 commits to the project's source code repository. Similarly, in the first day of the AspectJ[3] project, 87 files were modified in 2873 commits. The number of modified files and the extent of modifications gradually decreased over time and became stable. The time needed to reach this stability depends on the age of the project. Accordingly, the time stamp of when the context of the source code files was used or modified in the project play a significant role in determining the degree of relevancy of the context with a new change request. This observation encourages to deal with the time-metadata associated with the repository's text data to locate the change request in the software source code.

4.1.1.2 Developer-aspect analysis of data

In developer-aspect analysis of the data, the value of the data is determined based on the developer who created or modified the data in source code files. In this aspect, the text data that were created or changed by a project's developer in a source code file is taken as the developer's expertise profile on the corresponding file (Schuler & Zimmermann, 2008). Accordingly, the developer's expertise profile contains the vocabularies and terms used by the project developer during the activities she performed on the file such as fixing a bug, adding new feature or improving an existing feature that resulted in the changes in the source code file.

The analysis of developers' expertise is more common in bug report assignment research area that aims to determine an appropriate project's developer to fix a new bug reported to the project (Anvik et al., 2006; Anvik, 2006; Kagdi et al., 2012). Recently, some bug assignment researches recommend developers through identifying the most sim-

[2]http://www.eclipse.org/jdt/.

[3]http://www.eclipse.org/aspectj/.

ilar source code files to the newly reported bug which is due to the undeniable correlation between the data in the source code files and the project developers who are working on the source code files (Hossen et al., 2014; Kagdi et al., 2012; Servant & Jones, 2012; Shokripour et al., 2013, 2014). The basis of the idea of developer-aspect analysis of data for feature location is determining the location of a new change request based on the concentration of the developers' activities on a subject in a source code location that indicates the importance of the subject for the location. Centralization of the relevant activities of the developers to the concept of the change request on a specific location of the project increases the potential of this location to be the correct location pertinent to the new change request.

The notion of the developer-aspect analysis of data compared to the typical analysis of data for feature location is illustrated and elaborated through an example. Consider that the term "security" has appeared in four changesets[4] made by different developers in a source code file. In a typical analysis of data, the term "security" is only weighted one time due to the appearance of the term in a single source code file. In contrast, in developer-aspect analysis of data, the term would be weighted differently depending on the number of developers who used the term in their changesets for the file. In this case, having two or more developers working on a related subject (e.g. bug report or feature) in a source code file indicates a higher concentration of the subject in the corresponding file. This knowledge can then be used to relate a new change request to the locations that need to be changed in the source code to satisfy the new change request. In short, in developer-aspect analysis of data, the term that had been used by different developers in a source code file has more impact in determining the relevancy of the change request and the source code file.

[4]A changeset is an atomic set of changes of the source code files committed to the source code repository by a project developer during a maintenance activity (Ratanotayanon et al., 2010)

4.1.1.3 Noun Usage

In many languages, nouns are known as the most important category of terms that represents the meaning of a text. Moreover, previous research on text analysis indicates that nouns carry most of the meaning of a sentence (Bouras & Tsogkas, 2010). Accordingly, selecting the nouns, which exist in a text, results in better semantic representation of the text. Furthermore, the use of only nouns results in reducing the amount of noises in the extracted entities from the data sources (Capobianco et al., 2012; Sarawagi, 2008), thereby enhancing the effectiveness of improvements made by other means. Moreover, using only the noun terms leads to significant reduction of the dataset size (Capobianco et al., 2009; Shokripour et al., 2013).

On the other hand, the noun usage in other research areas such as traceability (Capobianco et al., 2012) and bug assignment (Shokripour et al., 2014) leads to the improvement of traceability and bug assignment process. Nevertheless, using only nouns leads to independence of the approach from dimensionality reduction methods, which is one of the challenges in IR methods (Crain et al., 2012). The use of only nouns provides enough information to make a feature location decision for a given change request. The results of this research (reported in Chapter 6) indicate the sufficiency of the information. According to these enumerated benefits, the proposed approach in this thesis deals only with the noun terms to locate a change request in the source code of the project.

4.1.2 Features and Components

Applying these perspectives in text analysis process resulted in proposing two methods. Then, from the combination of these methods, a new approach is proposed. The features and main functional components of the proposed methods and approach are briefly explained in the next few sub-sections.

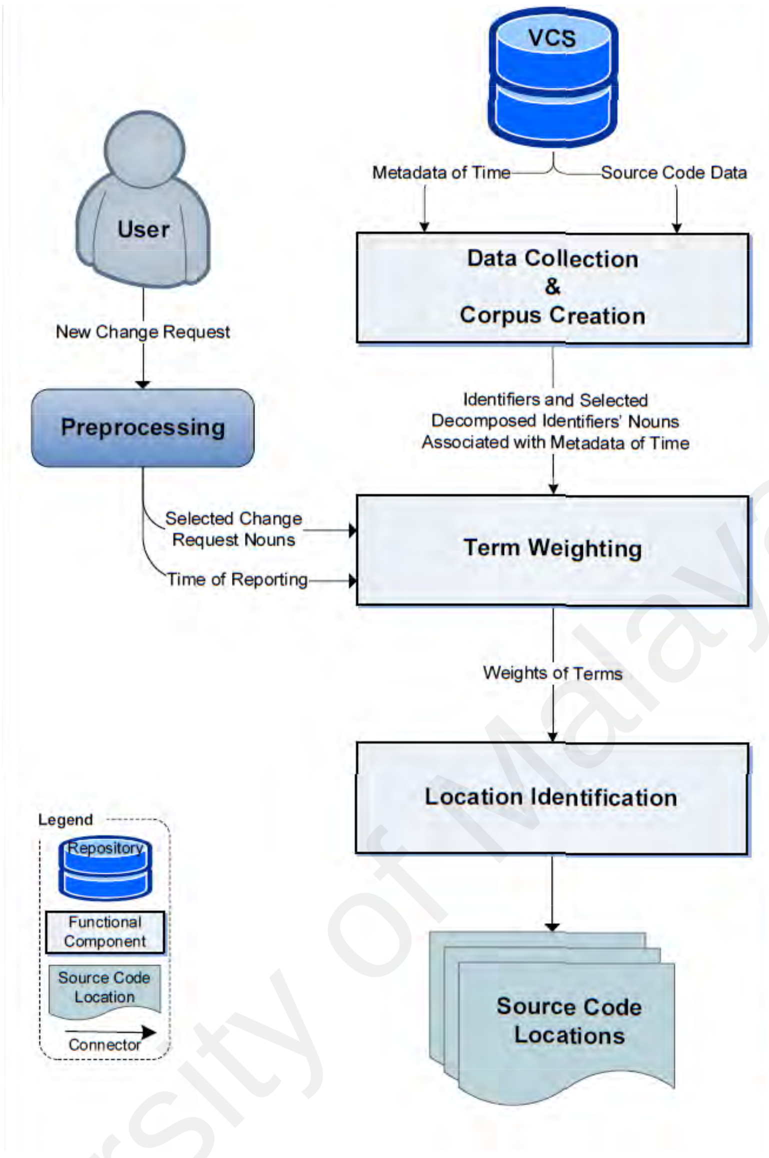


Figure 4.1: Abstract view of the first proposed method, ***Ti***NoFeLo

4.1.2.1 ***Ti*NoFeLo Method Overview**

The first proposed method in this thesis, ***Ti***NoFeLo, analyzes the data from the aspect of time when the data was recorded in the repository. ***Ti***NoFeLo extracts the noun terms that were recorded in the source code files to determine the similarity between the source code files and the desired change request. This implies that the two perspectives of analyzing the data from the aspect of time and the use of only noun terms to reduce the dataset size were applied in this method to improve the text analysis process of feature location.

As it is shown in Figure 4.1, the ***Ti***NoFeLo method locates a change request through three main functional components, i.e. Data Collection and Corpus Creation, Term

Weighting, and Location Identification. In the Data Collection and Corpus Creation component, the required data are collected from the VCS of the software system and preprocessed into a time-based corpus in preparation for the next component - Term Weighting. The output of this component includes the selected noun terms from the VCS that are linked to the corresponding source code files as well as the time-metadata related to the terms creation in the files. The next component, Term Weighting, uses a time-based term-weighting technique to determine the values of the noun terms extracted from the VCS based on the number of appearances in the associated source code files and the times of using those terms in the files. Finally, in the last component, the source code files are indexed in descending order based on the sum of the calculated weights for the common terms that appeared in both the source code files and the desired change request. The higher ranked files in the list have higher similarity with the given change request.

4.1.2.2 ***De*NoFeLo Method Overview**

The second proposed method, ***De***NoFeLo, improves the text analysis process of feature location by applying the two perspectives of analyzing the text data from the aspect of developer and using only the noun terms. As shown in Figure 4.2, similar to the ***Ti***NoFeLo method, ***De***NoFeLo relies on three main functional components to identify related source code locations for a new change request by analyzing the developers' expertise profiles. The functionality of these components is similar to that of the corresponding components in the ***Ti***NoFeLo method with a few differences. The first component, Data Collection and Corpus Creation, collects the required data from the VCS, and then, preprocesses into a developer-based corpus. In the developer-based corpus, the selected noun terms from the VCS are linked to the corresponding developer's expertise profile on the source code files as well as the time-metadata. In the next component, Term Weighting, the values of the terms that appear in both the developer's expertise profile and the new change request,

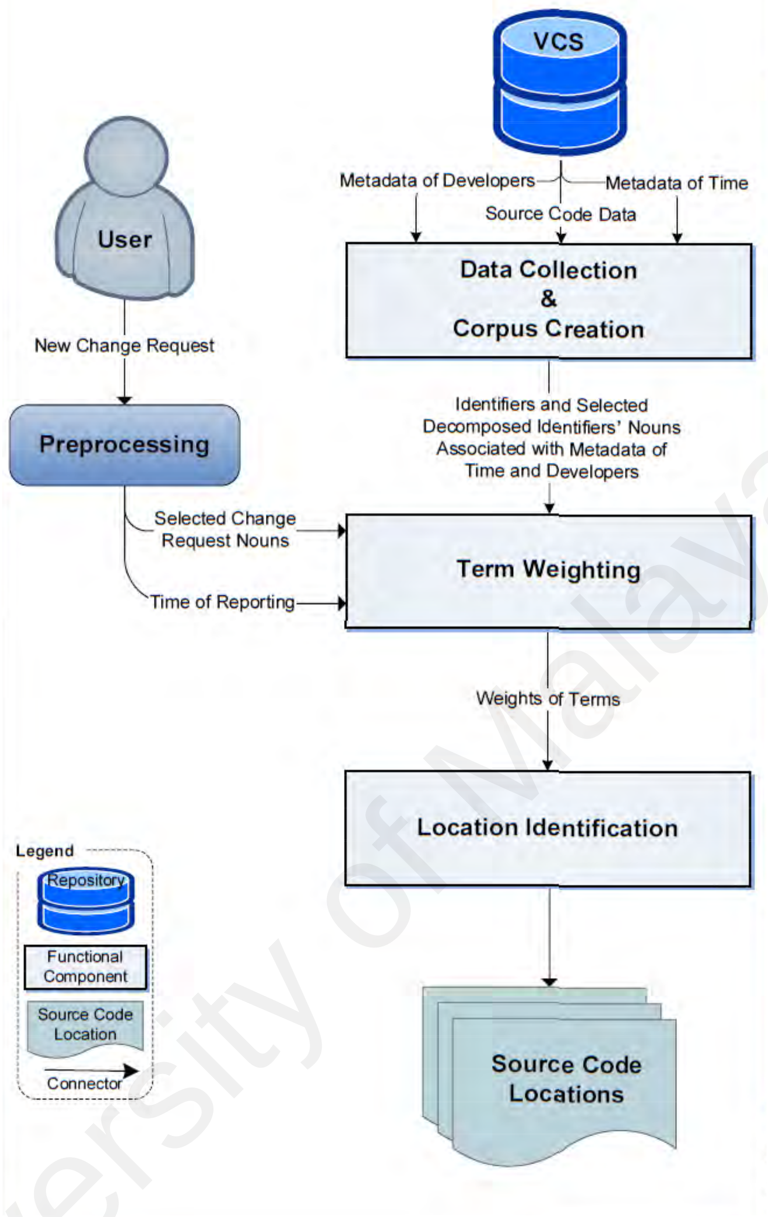


Figure 4.2: Abstract view of the second proposed method, *De*NoFeLo

known as the common terms, are determined based on the frequency of usage by the developer in the corresponding source code files and time of usage. Finally, in the last component, the source code files are indexed and sorted in descending order based on the summation of the calculated weights for the common nouns.

4.1.2.3 ***TiDe*NoFeLo Approach Overview**

Having the proposed methods, ***Ti***NoFeLo and ***De***NoFeLo, a new feature location approach is proposed based on the combination of the two proposed methods (See Figure 4.3). The

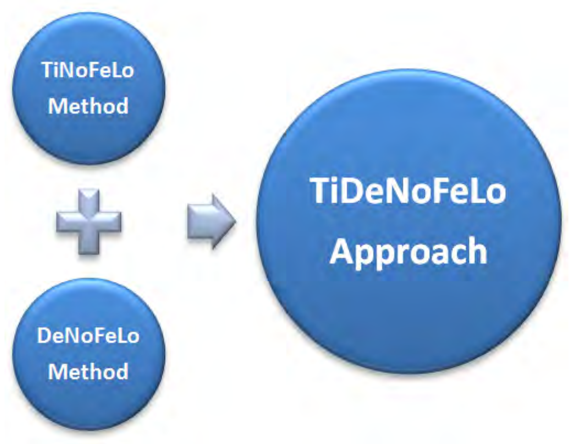
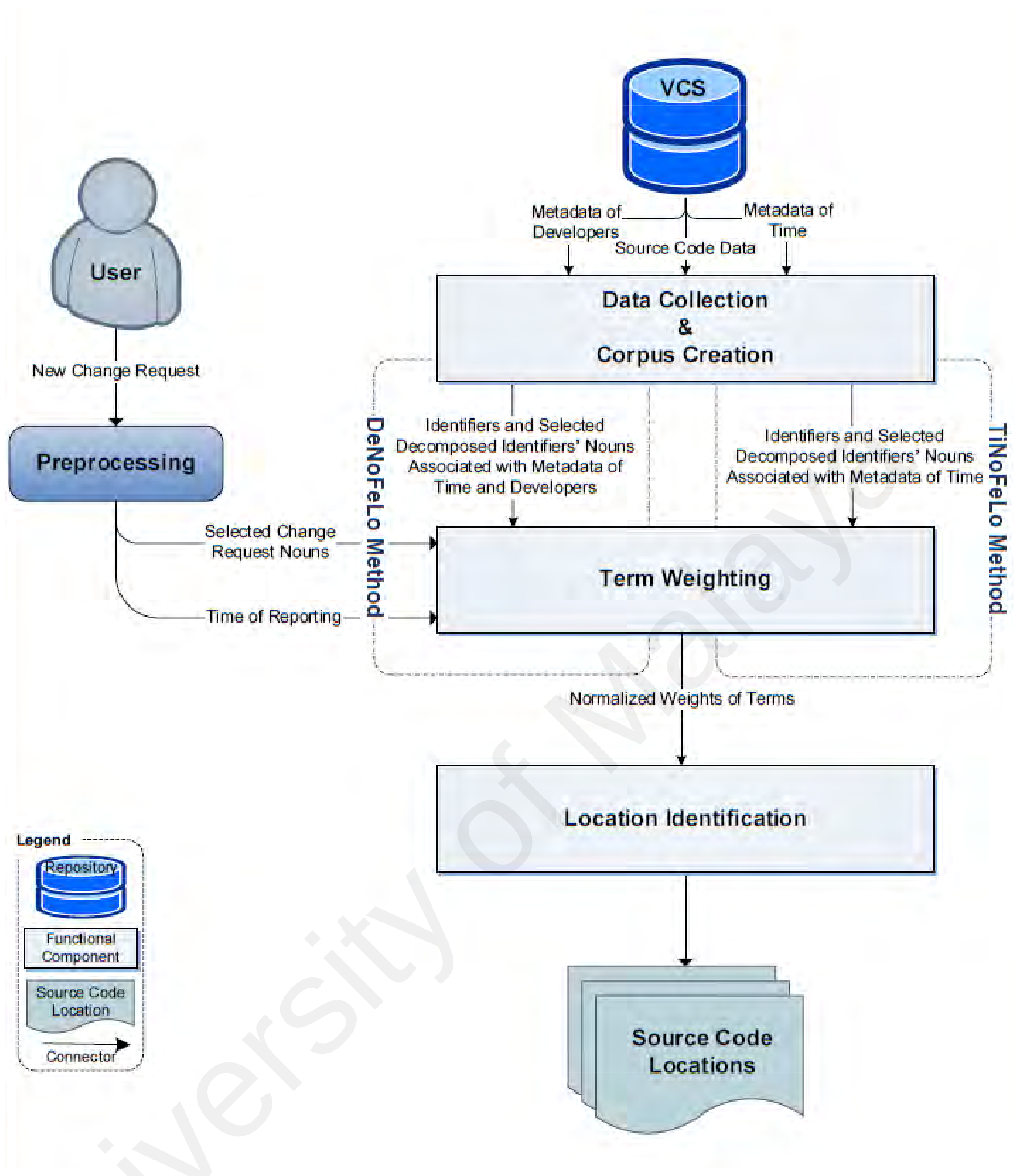


Figure 4.3: *Ti*NoFeLo and *De*NoFeLo methods embody *TiDe*NoFeLo approach

proposed approach, named ***TiDe***NoFeLo, addresses all identified perspectives in one approach. The proposed approach analyzes the data from both the aspects of time and developer, and also uses only the nouns to improve locating the change request in the source code.

As shown in Figure 4.4, the proposed approach, similar to the proposed methods, includes three main components to determine the similarity of the source code files with the given change request. The required data are collected from the VCS in the Data Collection and Corpus Creation component and preprocessed to extract and refine the noun terms. The output of this component is a corpus that includes the source code files as the documents. The documents in this corpus include the refined noun terms that were linked to the corresponding source code files as well as the developer-metadata and time-metadata. Then, in the Term Weighting component, the noun terms that appear in both the source code file and the given change request, known as common nouns, are weighted through two aspects, i.e. the aspect of time and the aspect of developer. This implies that two different weights are calculated for each common noun term. The calculated weights are combined to determine the total weight of the term. The summation of the total weights of the common noun terms is used to calculate their scores in the Location Identification component. Finally, the files are ranked based on the scores of the files in



VCS
Metadata of Developers
Metadata of Time
Source Code Data
Data Collection & Corpus Creation
User
New Change Request
Preprocessing
Identifiers and Selected Decomposed Identifiers' Nouns Associated with Metadata of Time and Developers
Identifiers and Selected Decomposed Identifiers' Nouns Associated with Metadata of Time
DeNoFeLo Method
TiNoFeLo Method
Selected Change Request Nouns
Time of Reporting
Term Weighting
Normalized Weights of Terms
Location Identification
Source Code Locations
Legend
Repository
Functional Component
Source Code Location
Connector

Figure 4.4: Abstract view of the proposed approach, *TiDe*NoFeLo

descending order.

The details of the components and functionality of each one in the proposed methods and approach are explained in the rest of this chapter. The details of the ***Ti***NoFeLo and ***De***NoFeLo methods are presented in Sections 4.2 and 4.3, respectively. Finally, the ***TiDe***NoFeLo approach which is embodied by ***Ti***NoFeLo and ***De***NoFeLo is introduced in details in Section 4.4.

4.2 Detailed View of ***Ti*NoFeLo** Method

In the first phase of this research, a new feature location method is proposed to address the differences between the repository's text data and the text data in natural language context. In the proposed method, ***Ti***NoFeLo, the extracted information from the source code repository, VCS, is used as the dataset. As mentioned previously, only the noun terms recorded in this dataset is used to identify related source code locations for a new change request. In ***Ti***NoFeLo, the nouns are weighted based on the frequency of their appearances in the context and the time of usage. Accordingly, a new term-weighting technique called Time-Aware Term-Weighting technique, TATW, is proposed to consider both the frequency and time parameters.

In order to locate the new change request in the source code, the proposed method, which is organized into three components, is applied on the collected data from the VCS. As shown in Figure 4.5, the main functional components of ***Ti***NoFeLo are as follows:

- Data Collection and Corpus Creation
- Term Weighting
- Location Identification

Through these components, the data collected from the VCS is analyzed to identify the correct source code locations pertinent to the new change request. Details about the functionality of these components are described in the rest of this section.

4.2.1 Data Collection and Corpus Creation Component

The Data Collection and Corpus Creation component integrates the collected data and metadata from the software repository based on the identified source code locations. Then, it provides the integrated data in a format that can be used by the next component. This

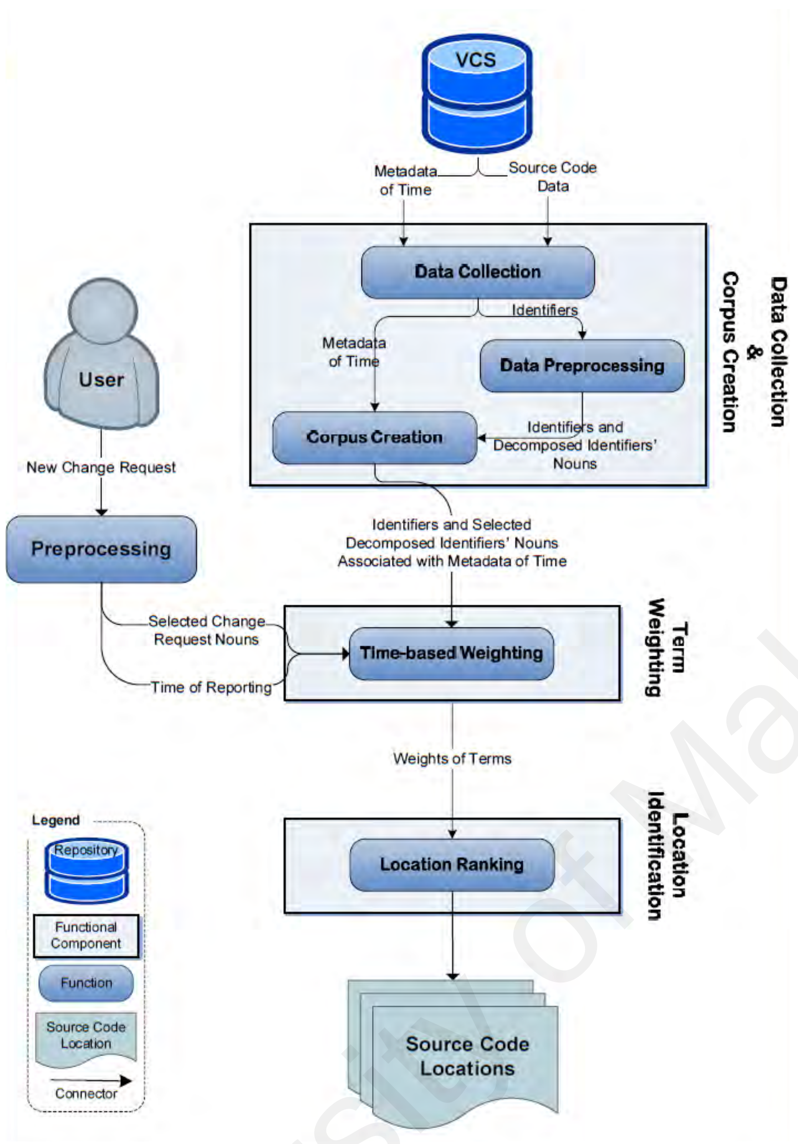


Figure 4.5: Detailed view of the *Ti*NoFeLo method

component uses a set of functions called Data Collection, Data Preprocessing, and Corpus Creation. Note that this component can be reused for other Mining Software Repositories (MSR) tasks. The rest of this section discusses these functions.

4.2.1.1 Data Collection

The Data Collection function collects the source code files referred to in the activity logs from the VCS. The data collected from the VCS log contains a history of the committed modification activities for the source code entities, typically called source code revisions or commits. A commit involves a number of items including a revision number, a commit date, a commit message, and the modified code. From the source code files, the identifiers which are used to name the source code entities such as files, classes, and methods are extracted.

Identifiers are usually constructed as a combination of terms that identify the functionality and behavior of the desired entity (Abebe & Tonella, 2010). In other words, this set of terms has a meaningful relationship with the functionality of the associated source code entity. Based on this relationship, useful information can be extracted from the identifiers. In this study, the names of classes, methods, fields, and method parameters are extracted as identifiers, as they contain the most useful and least noisy data. Time of creation or modification of each of the identifiers is also extracted from the VCS log and then associated with the desired identifier.

4.2.1.2 Data Preprocessing

A rich set of contents in a variety of formats is found in the software repositories of a project, and as a result, different techniques are required to analyze them. From the source code files found using data from the VCS, the identifiers are extracted. Recall that the identifiers include the names of classes, methods, fields, and method parameters. For each extracted identifier, the time of commit (which may also be considered the identifier's creation or modification time), file name and revision number are extracted to be used in the term weighting process. As previously discussed, identifiers are usually created by concatenating a set of terms. Therefore, the identifiers are decomposed using the approach recommended by Butler et al. (Butler, Wermelinger, Yu, & Sharp, 2011). This results in a set of terms from which the identifiers are formed (hereafter referred to as decomposed identifiers). Furthermore, change request reporters sometimes mention the name of a class or method directly in the explanation of the change requests (Bacchelli, Lanza, & Robbes, 2010). Therefore, both the complete and decomposed identifiers are used for identifying source code locations.

Furthermore, the new change request that needs to be located in the source code of the project must be preprocessed to be unified with the data collected from the repository. From the new change request, Summary and Description fields are used to determine the textual similarity with the source code files. The text appearing in the Summary and the Description of the change request is written in natural language and therefore contains a lot of noisy data. To address this issue, Named Entity Recognition (NER) (Sarawagi, 2008) is used to refine the text data. The output of NER consists of all of the terms in these two fields that are categorized as nouns, adjectives, or verbs in the sentences. As previously mentioned, there are many benefits to using only noun terms. Therefore, only the nouns that are found in the Summary and Description of the new change request are used.

To extract the noun terms, first, the roles of the terms in their associated sentences are identified. Then, based on the identified roles, the categories of the terms are determined. From the categorized terms, only the nouns are extracted to be used in the location identification process. In this case, the terms that can appear in various roles in a sentence are categorized based on their role in the specific sentence. For example, if the term 'book' appears in two roles like "book room" and "room booking", the noun extraction step can categorize the first one as a verb and the second one as a noun.

The nouns of the decomposed identifiers are used to find similarities in the data of source code entities and the new change request as well as the complete identifiers. More details on the noun categorization and extraction are explained in Chapter 5. Moreover, nouns that contain symbols, digits or have less than three characters are filtered out as it has been done in other mining repository approaches (Shokripour et al., 2013). All of the selected nouns are lemmatized in order to reduce the different forms of a term.

Lemmatization refers to the preprocessing of terms for use with a particular vocabulary and morphological analysis of the terms. Typically the goal of lemmatization is the removal of inflectional endings only, and to return the base or dictionary form of a term. This form is known as the lemma (Capobianco et al., 2012; Manning et al., 2008).

4.2.1.3 Corpus Creation

As mentioned previously, in this research, both the complete and decomposed identifiers are used to identify the related source code locations for a new change request. All this data are associated with the corresponding metadata extracted from the VCS log including the file name, revision number, and the time of commit to the source code repository. It means that in this component, each file is represented by the complete identifiers and the nouns of decomposed identifiers which are associated to their corresponding metadata. The collection of the files storing the source code of software system forms the time-based corpus. In this corpus, the metadata associated to the terms provides the ability of analyzing the data from the aspect of time when the data was created.

4.2.2 Term Weighting Component

In this component, the collected text data from the VCS that were linked to their associated metadata are weighted using the proposed term-weighting technique. In this technique, the text data are weighted based on the textual similarity to the new change request, taking into consideration the value of similar text over time. Due to the importance of time in this technique, the proposed term-weighting technique is called Time-Aware Term-Weighting technique (TATW).

Two factors are taken into account in the TATW technique. The first is term frequency with respect to the source code identifiers. The second is time when the term was used. Term frequency has been used in other term-weighting techniques, such as the Term

Frequency technique and the TF-IDF technique (Manning et al., 2008). It focuses on the number of appearances of a term in a document or corpus. In the TF-IDF technique, which is the most-used term-weighting technique in text analysis methods, the weight of a term is related to the frequency of that term in an individual document, and its inverse frequency across the corpus. As previously mentioned, the terms that are used in the proposed method are restricted to only nouns. To be more specific, noun frequency is used in the proposed term-weighting technique instead of the more general term frequency.

In addition to considering the noun frequency, the proposed term-weighting technique considers time of usage, or how far in the past a noun term was used. The consideration of when a noun term was used is the key difference between the proposed term-weighting technique and the classic term-weighting techniques. More specifically, in the proposed term-weighting technique, the following four parameters are considered in the relevancy calculation of the noun term from the data sources with regard to the new change request.

- Frequency of the noun term *i* (N_i) in the set of noun terms that were created or modified at the same time. This parameter represents the frequency of which the noun term was used overtime.
- Frequency of the noun term *i* in a specific document which is a source code file (F_k).
- Frequency of a noun term *i* in the corpus containing all the source code files of the project (FPrj). This parameter represents the generality of the noun term in the corpus.
- Time interval between when the new change request was reported ($Date_{NCR}$[5]) and the time of creating or modifying the noun term *i* in the software repository.

[5]The format of date is such as “2005-11-21” and the time interval between these two dates is based on the number of days

The first three parameters are the frequencies of noun term *i* at different levels of documentation. Existing term-weighting techniques primarily focus on using the frequency of appearances of a term in source code files across the project (Manning et al., 2008; Zhou et al., 2012). However, in the TATW technique, the value of the noun term over time is considered for the new change request.

In the proposed term-weighting technique, the time difference between uses of a noun term in the new change request and the source code files is negatively correlated with the weight of the noun. In other words, a noun that is recently used in this resource, i.e. a short time difference has a higher value when compared with another noun that was used farther in the past. The time difference is based on the number of days and can therefore be very large when the term was used a long time ago.

To normalize the time difference, mathematical solutions such as square root or logarithm can be used. Table 4.1 shows the effect of square root and logarithm on some possible time difference (number of days) samples. As shown in this table, the use of logarithm strongly reduces the time difference value when compared with using square root. On the other hand, the differences between the logarithms of larger values for the number of days are not very much. For instance, the difference between the logarithms of 30 days and 180 days is 0.778 and the difference between the logarithms of 30 days and 365 days is 1.085. However, the differences of square roots of these pairs of days are 7.939 and 13.628, respectively. In other words, the use of a logarithm dampens the time difference too much for practical use in a term-weighting technique. Therefore, the use of square root is more suitable and more effective for term weighting in the proposed method. The decision to use square root is also supported by the use with the time differences in other MSR fields (Kagdi et al., 2012; Voinea & Telea, 2006).

In this component, the source code identifiers and the selected noun terms of the

Table 4.1: Effect of square root and logarithm on time differences

Number of Days	Square Root	Logarithm
7	2.646	0.845
30	5.477	1.477
90	9.487	1.954
180	13.416	2.255
365	19.105	2.562
730	27.019	2.863

decomposed identifiers (both referred to as N$_i$) are weighted with respect to those four aforementioned parameters. Accordingly, Equation 4.1 is used to calculate the weight of noun term i (N_i) in the changeset j (C_j) of the source code file k (F_k). A changeset is an atomic set of changes of the source code files committed to the VCS, and contains the modified code that is a new revision. In this equation, Freq$_{N_i,C_j}$ is the frequency of using noun term i in the changeset j. Freq$_{N_i,F_k}$ is the frequency of the noun term i in the source code file k and is used in existing term-weighting techniques. It highlights the occurrences of the noun term among all of the noun terms of a file (Manning et al., 2008).

Freq$_{N_iFPrj}$ indicates the frequency of the noun term i across all the project's source code files in the VCS. Finally, Date$_{NCR}$-Date$_{N_i,C_j}$ is the time difference between when a new change request was reported (Date$_{NCR}$) and when a noun term i was used in the changeset j (Date$_{N_i,C_j}$).

$$WeightN_i, C_j, F_k \quad \frac{Freq_{N_i,C_j} * Freq_{N_i,F_k}}{Freq_{N_iFPrj} * \sqrt{Date_{NCR} - Date_{N_i,C_j}}} \tag{4.1}$$

4.2.3 Location Identification Component

Having calculated the weights of the nouns appearing in both the new change request and the source code files, these weights are summed up to determine the scores of the source code locations. In the existing term-weighting techniques that do not weight terms with respect to time, each specific term has a single weight (Manning et al., 2008) and the summation of the term weights is the score of the file. However, using the TATW technique, the weight of a noun term is related to the creation or modification time of the term in the data source. In other words, the noun may have more than one weight depending on the number of times when it was used.

Equation 4.2 is used to compute the score of a source code file (F$_k$) based on the weights of the nouns collected from the changesets that were recorded in the VCS (Score(F$_k$)), and the weights of the common nouns (i.e. those nouns appearing in both the new change request and the source code file *(*k)). A summation of the number of common nouns between the new change request and the changeset *j* of the file *k* is made for each of the changesets for the file *k*. In other words, the nouns that appear in more than one changeset are weighted individually and these weights are added together to calculate the score of the file *k*.

$$ScoreF_{kVCS} \quad \overset{\#ChangesetsforF_k}{\underset{j1}{}} \quad \overset{\#CommonNouns}{\underset{i1}{}} WeightN_i, C_j, F_k \tag{4.2}$$

The output of this component is a ranked list of source code files in descending order, based on the calculated scores for the files. Therefore, the most relevant locations for the new change request are ranked at the top.

To summarize, this method focuses on two main aspects. The first contribution of the ***Ti***NoFeLo method is consideration of time-metadata in analyzing the text data recorded in the software repository. This contribution provides the ability of analyzing the context of the repositories over time. It means that we can deal with the evolution of the software project which is one of the important differences of the text recorded in the repositories and text in natural language context. The second aspect which is taken into account in ***Ti***NoFeLo is using only the noun terms to make an improvement in text analysis of feature location by dealing with more meaningful and less noisy data. By using only the noun terms, not only sufficient data for feature location is provided, but also the size of dataset is significantly reduced[6]. Further elaboration on the effects of time consideration and noun usage is discussed in Chapter 6.

4.3 Detailed View of ***De***NoFeLo Method

For the second step of this research, another difference of the repository's text data with the natural language text is considered in text analysis process of feature location. The data recorded in the repository are associated with the project developers who are working on the software project. This implies that the developer who created and modified the data in the project repository is considered as the owner of the modified data. Accordingly, in the second phase of this research, a new feature location method is proposed to consider this specific characteristic of the repository's text data in order to further improve the accuracy of feature location. This method is referred to as Developer-aspect analysis of data in a Noun-based Feature Location method, ***De***NoFeLo. To consider the metadata of developer in this method, a new term-weighting technique, called Time-Aware Developers' expertise Term Weighting technique (TADTW), is proposed.

As shown in Figure 4.6, similar to the ***Ti***NoFeLo method, ***De***NoFeLo relies on three main functional components to identify related source code locations for a new change request. In the first component, the required data are collected from the VCS and preprocessed into a corpus in preparation for the next component - Term Weighting. In the Term Weighting component, the values of the common terms, which appear in both the developer's expertise profile and the new change request, are determined based on the frequency of use by the developer and the last time of usage of the term by the developer. Finally, in the last component, the source code files are indexed and sorted in descending

[6]The other types of terms such as verbs and adjectives are excluded from the dataset.

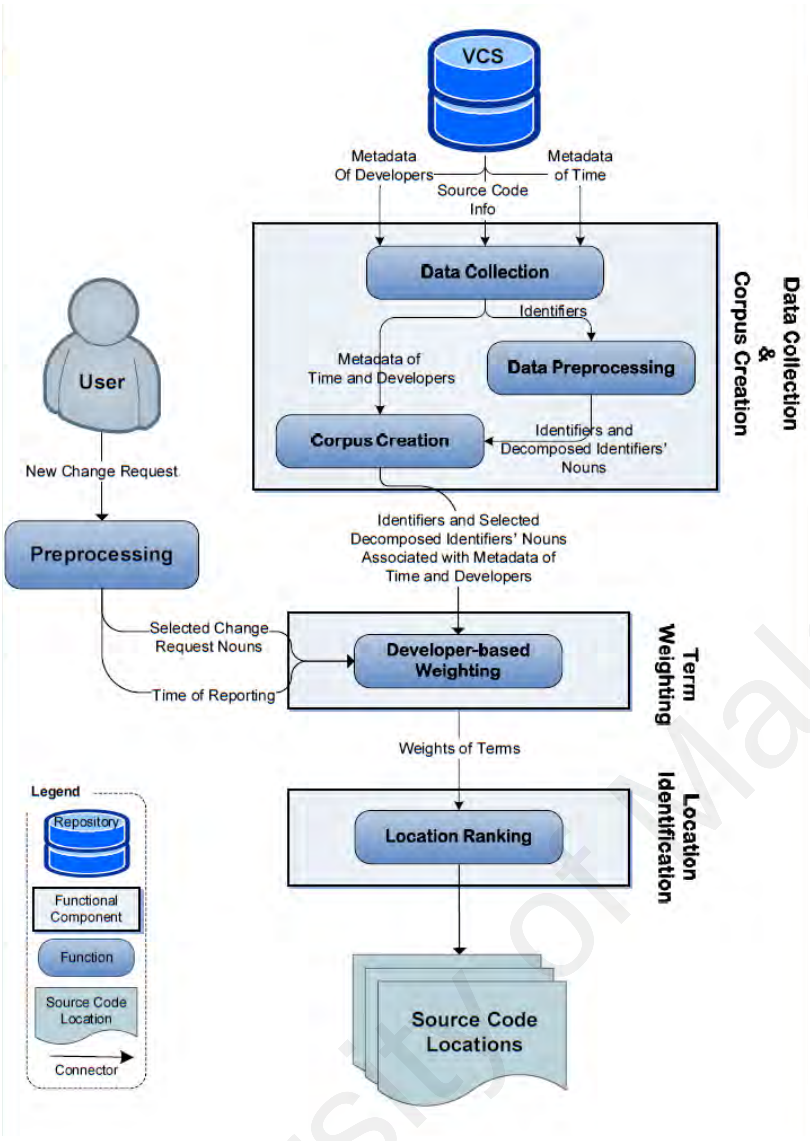


Figure 4.6: Detailed view of the *De*NoFeLo method

order based on the sum of the calculated weights for the common terms. Details about the functionality of these components are described in the rest of this section. Some of the functions in these components are similar to the corresponding ones in the previous method, ***Ti***NoFeLo. Accordingly, a brief description on these functions is presented for this method and readers are referred to the ***Ti***NoFeLo method for further details.

4.3.1 Data Collection and Corpus Creation Component

This component collects the required data from the software repository of the VCS. Then, the collected data are refined and associated with the metadata of developer and time to be prepared in the format of data which are used by the next component. This component involves a set of functions called Data Collection, Data Preprocessing, and Corpus Creation.

4.3.1.1 Data collection

Similar to ***Ti***NoFeLo, the main dataset for ***De***NoFeLo is the source code data which is collected from the VCS. In typical feature location methods, a source code file or method is treated as a unique document of a corpus. However, as mentioned earlier, the recorded data in the source code file or method can be analyzed from different aspects, e.g. analysis of the data from the aspect of time as in the ***Ti***NoFeLo method. In the ***De***NoFeLo method, the data are analyzed from the aspect of developer who works on the source code files to identify the locations related to the new change request. Accordingly, ***De*** NoFeLo's corpus, which is a developer-based corpus, is different from the typical corpus used for feature location.

The required data to create the developer-based corpus is collected from the VCS log and the history of the committed modification activities performed by the developers on the source code entities, typically called source code revisions or commits. A commit has a number of attributes including a revision number, commit date, commit message, identifier of the developer who modified the code, and the part of code that was modified.

4.3.1.2 Data Preprocessing

The preprocessing steps in ***De***NoFeLo are similar to the ones performed in ***Ti***NoFeLo. From the entire history of the project's source code, the identifiers which are used to name the classes, methods, fields, and method parameters are extracted to be involved in the corpus of ***De***NoFeLo. Since an identifier is usually constructed as a combination of terms (Abebe & Tonella, 2010); the identifiers are decomposed to obtain a set of terms from which the identifiers are formed. From the decomposed identifiers, only the noun terms are extracted excluding the noun terms that contain symbols, digits or have less than three characters. The noun terms are then lemmatized in order to reduce the different forms of a term (Capobianco et al., 2012; Manning et al., 2008; Shokripour et al., 2013). The detailed explanation of each of the preprocessing steps has been presented in Section 4.2.1.2.

4.3.1.3 Corpus Creation

Since the ***De***NoFeLo method analyzes the source code data from the aspect of the developers who work on the source code files, the ***De***NoFeLo's corpus is different from the typical feature location corpus and also different from ***Ti***NoFeLo's corpus. In this function, both the complete identifiers and the nouns of decomposed identifiers are used to create the developer-based corpus. In this corpus, all the data are associated with the corresponding metadata extracted from the VCS log including the file name, developer ID, revision number, and time of commit to the source code repository. Associating the term data to this set of metadata provides the ability to analyze the data from the developer-aspect. Each document in the developer-based corpus is a source code file from which profiles of developers' expertise can be derived.

4.3.2 Term Weighting Component

In this component, the data stored in the developer-based corpus are weighted based on the amount of similarity of the source code files with the given change request. Similarity of a source code file is determined by analyzing the developers' expertise profiles for the file. As mentioned above, the terms used by a developer in a specific file form the developer's expertise profile for that file.

To determine the weight of a term in a developer's expertise profile, in addition to the statistical computation (e.g. term frequency) which is typically used in existing term-weighting techniques such as TF-IDF, the developer who modified the term and also time of usage of the term are taken into consideration. Due to the consideration of these new factors in the term weighting process of the ***De***NoFeLo method, a new term-weighting technique which is named Time-Aware Developer expertise's Term Weighting, TADTW, is proposed. In this technique, the weight of a term is determined based on its importance among the other terms which were used by the same developer in a source code file and last time of usage of the term by the developer.

As mentioned earlier, similar to ***Ti***NoFeLo, the terms that are used in this method are restricted to only the nouns. Accordingly, the noun frequency is used in the proposed term-weighting technique, TADTW, to be more specific, instead of the more general term frequency. In TADTW, the frequency of appearances of a noun term in the developer's expertise profile and the last time of usage of the term by the developer is analyzed. These factors are the main differences of the proposed term-weighting technique from the existing ones. For more clarification on the proposed term-weighting technique, the important parameters of weighting a noun term are further investigated. Three parameters are considered in the similarity calculation of the noun term found in the developer's expertise profile with regard to the new change request:

- Frequency of the noun term *i* (N_i) in the set of noun terms that were created or modified by a specific developer D_j on a specific file F_k. This parameter calculates the frequency of usage of the noun term or the number of times that the noun appeared in the developer's expertise profile on the file F_k.

- Frequency of a noun term *i* (N_i) in all developers' expertise profiles. Each noun term may be used by one or more project developer in different source code files. We refer to this set of noun terms as keyword noun terms or briefly keywords. This parameter determines the value of the noun term in the corpus.

- Time interval between when the new change request was reported (Date_{NCR}) and the last time of usage of the noun term i by the developer j.

The frequency of a noun term in all developers' expertise profiles shows the importance of the noun in the project. The noun terms that are used by most of the developers are general terms in the project. Inversely, the ones that are used by a very small percentage of the developers have a higher value for determining the similarity of a source code file and a change request. For an instance of the keywords, in a Health Insurance software project, the term "insurance" is likely to be used by almost every developer of the project. However, the term such as "security" may appear in a few developers' expertise profiles. The terms that appear too often need to have less effect on determining the similarity. Conversely, the nouns that are used by a very small set of the developers have a higher value for determining the similarity. As mentioned above, this set of noun terms that appear in a small set of profiles are referred to as keywords.

With respect to these parameters, Equation 4.3 is used to calculate the value of a term from the aspect of developer. This equation considers the statistical analysis of the noun term used by the corresponding developer and the last time of usage of the noun by the developer. In this equation, the weight of the noun term i (N_i) in the file k (F_k) that appears in the expertise profile of developer D_j is calculated based on the time and frequency of usage of the noun by the developer. The first parameter in determining the weight of the noun term is $\text{Freq}_{N_i,D_j,F_k}$ that adjusts the importance of the noun in the expertise profile of the developer D_j on the file F_k. This parameter calculates the frequency of using the noun term or the number of times that the noun appeared in the developer's expertise profile on file F_k.

$$
\begin{aligned}
WeightN_i, D_j, F_k \ & \left(Freq_{N_i,D_j,F_k} \times log \frac{\#PrjDev}{Freq_{N_i,D_j}} \right) \\
& \times \left(\frac{1}{\sqrt{Date_{NCR} - Date_{N_i,D_j}}} \ \frac{1}{Freq_{N_i,D_j}} \right)
\end{aligned} \tag{4.3}
$$

Only dealing with the raw noun frequency to determine the similarity signifies that all the nouns in the developer's expertise profile are considered equally important. However, some of the noun terms may have more significant role in locating the new change request. The terms that appear too often need to have less effect on determining the similarity. Conversely, the nouns that are used by a very small percentage of the project developers have a higher value for determining the similarity. This set of noun terms that appear in a small set of profiles are referred to as keywords. Accordingly, in addition to the noun frequency, there is a need to specify the keywords by giving more weight to the nouns that were used by a small number of developers in the project. Accordingly, the logarithm of the total number of the project developers (#PrjDev) over the number of developers who used noun term i (N_i) in the project, represented by developer frequency (Freq$_{N_i,D_j}$), is also considered in weighting the noun term.

As previously mentioned, an effective parameter in determining the weight of a term in the source code file is the last time when it was used by the developer. The time difference between uses of a noun term in the new change request and the developer's expertise profile is negatively correlated with the weight of the noun. Thus, a noun that is recently used in this resource, i.e. a short time difference, has a higher value when compared with another noun that was used farther in the past. On the other hand, since the time difference is based on the number of days, it can be very large for the nouns used a long time ago. Accordingly, similar to the ***Ti***NoFeLo method, the inverse of the square root of time differences, between the last time of using the noun by the developer and the time of reporting the change request, is taken into account in weighting the nouns in the TADTW technique.

However, in ***Ti***NoFeLo, all of the specific time(s) that a noun was used in the file is considered; whereas in the ***De***NoFeLo method, only the last time that the developer D_j used the noun i (N_i) in her expertise profile is examined. This is to simplify the term weighting process. Furthermore, in ***Ti***NoFeLo, the data are analyzed from the aspect of time. Thus, all the specific times when the term was used in the source code file need to be weighted separately. However, in the ***De***NoFeLo method, the basic assumption is analysis of the data from the aspect of developer and the focus is not on time of usage of the term. Consideration of the last time that a developer used a noun term indicates the most recent time when the developer worked on the feature or subject related to the change request. In this regard, a higher value is given to the related noun terms that were created or modified most recently. In time consideration of Equation 4.3, Date_{N_i,D_j} indicates the last time when the noun N_i was used by the developer, and Date_{NCR} indicates time that the new change request was reported to the project[7]. In this case, the noun terms that had been used farther in the past would obtain a lower weight.

However, due to the significant role of the keywords in finding the related locations in the source code (Poshyvanyk, Gethers, & Marcus, 2012), a higher value is given to these noun terms, even if they have been used further in the past. Accordingly, the inverse of the number of developers who used noun N_i which is represented by the developer frequency (Freq_{N_i,D_j}), is combined with time consideration part of the equation. Without the inverse of developer frequency, the keywords that were used long time ago will have a lower weight due to the large time difference. In this case, the keyword terms that

[7]Note that the time difference is calculated in days

were used recently have a higher weight when compared to the other terms having the same frequency. In general, this equation gives a higher weight to the unique noun terms (keywords) that were most recently used and gives a lower weight to the terms that were used either further in the past or were used by most of the project developers.

4.3.3 Location Identification Component

Having computed the weights of the common noun terms that appear in both the new change request and the developers' expertise profiles, the expertise score for the developer D$_j$ is next calculated for the change request with respect to the file F$_k$. Equation 4.4 calculates the expertise score of developer *j* by summing up the weights of all common nouns appearing in the expertise profile of the developer *j* on the file *k* and the new change request.

$$ScoreD_j, F_k \quad \overset{\#Common\ Nouns}{\underset{i1}{}} WeightN_i, D_j, F_k \tag{4.4}$$

Since more than one developer may work on the same file, the final score of the file *k* is determined by summing the calculated expertise scores of the developers who work on the file *k*. Equation 4.5 calculates the final score of the file F_k.

$$FinalScoreF_k \quad \overset{\#Developer\ for\ F_k}{\underset{j1}{}} ScoreD_j, F_k \tag{4.5}$$

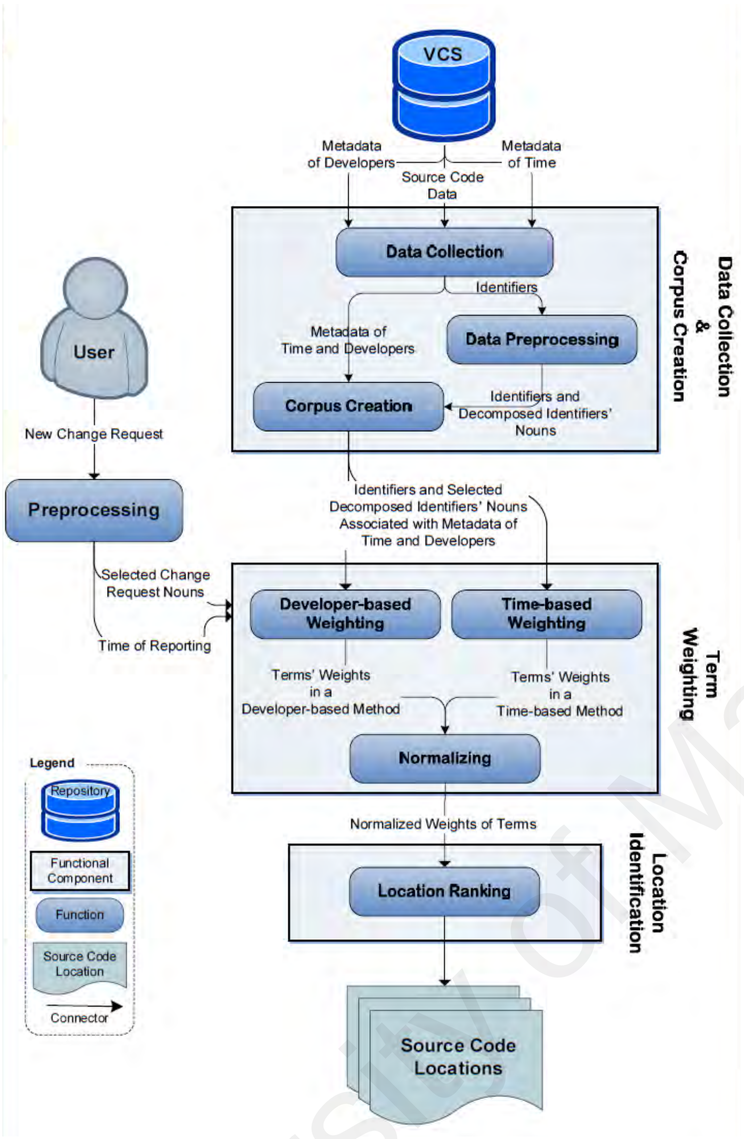
Finally, the source code files are then sorted in descending order based on the final scores. The output of this component is a ranked list of the files in descending order of relevancy. In this list, files ranked at the top of the list have a higher similarity with the given change request than those ranked lower in the list.

4.4 Detailed View of ***TiDe*****NoFeLo Approach**

To apply all identified perspectives together in text analysis process of feature location, a new feature location approach is proposed in the last step of this study. The identified perspectives include (i) the analysis of data from the aspect of time, (ii) the analysis of data from the aspect of developer, and (iii) the use of only noun terms to reduce the dataset size. As mentioned in Section 4.2, the ***Ti***NoFeLo method analyzes the data from the aspect of time and uses only the noun terms. Moreover, the ***De***NoFeLo method analyzes the data from the developer-aspect and also uses only the noun terms (See Section 4.3). The combination of these two proposed methods results in a feature location approach that addresses all the identified perspectives. Since this approach is the combination of ***Ti***NoFeLo and ***De***NoFeLo methods, we call it ***TiDe***NoFeLo which is taken from the combination of the names of ***Ti***NoFeLo and ***De***NoFeLo.

In this approach, the data recorded in source code files is analyzed from both the aspects of time when the data was recorded in the file and the aspect of the developer who recorded the data. Furthermore, only the noun terms are extracted from the datasets to locate the new change request in the source code of the project. This implies that all the identified perspectives are addressed in the ***TiDe***NoFeLo approach. As shown in Figure 4.7, similar to the proposed methods, this approach includes three functional components to determine the similarity of the source code files with the new change request. These components include (1) Data Collection and Corpus Creation, (2) Term Weighting, and (3) Location Identification.

In the first component, the required data are collected from the VCS and then preprocessed to extract and refine the noun terms from the data recorded in the source code files. The refined noun terms were used to create the corpus based on the developer-metadata



VCS
Metadata of Developers
Source Code Data
Metadata of Time
Data Collection
Identifiers
Metadata of Time and Developers
Data Preprocessing
Corpus Creation
Identifiers and Decomposed Identifiers' Nouns
Data Collection & Corpus Creation
User
New Change Request
Preprocessing
Identifiers and Selected Decomposed Identifiers' Nouns Associated with Metadata of Time and Developers
Selected Change Request Nouns
Time of Reporting
Developer-based Weighting
Time-based Weighting
Terms' Weights in a Developer-based Method
Terms' Weights in a Time-based Method
Normalizing
Term Weighting
Normalized Weights of Terms
Location Ranking
Location Identification
Source Code Locations
Legend
Repository
Functional Component
Function
Source Code Location
Connector

Figure 4.7: Detailed view of the *TiDe*NoFeLo approach

and time-metadata. Next, in the Term Weighting component, the noun terms, which appeared in both the source code file and the new change request, are weighted through two aspects, i.e. the time-aspect and the developer-aspect. This implies that there are two weights for each term. The determined weights are combined in the next step to calculate the total weight of each noun term. The summation of the total weights of the noun terms that appeared in both the source code file and the new change request is used to calculate the score of the file in the Location Identification component. Finally, the files are ranked based on the score of the files in descending order. The higher the file in the ranked list, the more similarity it has with the new change request.

Details about the functionality of these components are described in the following sub-sections. Since this approach is the combination of both the two previously proposed methods, some details of these components are similar to the corresponding ones in the previous methods.

4.4.1 Data Collection and Corpus Creation Component

Similar to the two proposed methods, this component collects the required data from the VCS which are used to record the changes to the source code of software projects. Then, the collected data are refined and associated with the metadata of developers and time to be prepared in the format of data which is used by the next component. This component involves three functions including Data Collection, Data Preprocessing, and Corpus Creation.

4.4.1.1 Data Collection

As mentioned previously, ***TiDe***NoFeLo analyzes the data recorded in the source code files from the aspect of time when the data was recorded and the aspect of the developer who recorded the data to identify the source code locations pertinent to a new change request. Accordingly, the corpus that is used in this approach is different from the typical corpus used in previous feature location methods and approaches.

In the Data Collection function, the data is collected from the VCS log and the history of the committed modification activities performed by the developers on the source code entities, typically called source code revisions or commits.

4.4.1.2 Data Preprocessing

After collecting the VCS data, the identifiers that identify the source code entities are extracted from the entire history of the project's source code. The details of extracting the identifiers are explained in Section 5.4.1. From the extracted identifiers, the names of classes, methods, fields, and method parameters are used in this approach similar to ***Ti***NoFeLo and ***De***NoFeLo. Furthermore, the identifiers are decomposed using the approach recommended by Butler et al. (Butler et al., 2011) and the terms that recognized as a noun were extracted from the decomposed identifiers. Moreover, the terms that contain symbols, digits or have less than three characters are filtered from the set of noun terms. Then, the terms are lemmatized in order to reduce the different forms of the terms (Capobianco et al., 2012; Manning et al., 2008; Shokripour et al., 2013).

4.4.1.3 Corpus Creation

In this study, both the complete and the refined nouns of the decomposed identifiers are used to create the corpus. All the data in this corpus needs to be associated with the corresponding metadata extracted from the VCS log including the file name, developer ID, and time of commit to the source code repository. Since, both the time-metadata and developer-metadata are important in this corpus, the corpus of ***TiDe***NoFeLo approach is similar to the one used in the ***De***NoFeLo method. Association of the term data to this set of metadata provides the ability to analyze the data from different aspects of developer and time.

4.4.2 Term Weighting Component

As mentioned earlier, in this approach, the analysis of the data for identifying the related source code files to the desired change request is performed from two aspects of time and developer. To determine the amount of relevancy of a source code file with the desired change request, the values of the terms that are appeared in both the source code file and the change request is calculated in this component. To calculate the weights of the terms, this component contains three functions involving Time-based Term Weighting, Developer-based Term Weighting, and Total Term Weighting. The functions of Time-Based Term

Weighting and Developer-Based Term Weighting have been used to determine the weight of the term from the aspect of time and the aspect of developer, respectively. Then, the calculated weights by these two functions are combined in the last function, namely Total Term Weighting. The details of these components are explained in the rest of this section.

4.4.2.1 Time-based Term Weighting

In this function, the terms recorded in a source code file are weighted from the aspect of time when the terms were used in the file. The value of the term over time is determined based on the TATW technique which was proposed to weight the terms in the ***Ti***NoFeLo method. The TATW technique determines the weight of the term based on the term frequency and time of usage. As mentioned in Section 4.2.2, TATW deals with four parameters to determine the weight of the noun. The first is the frequency of appearances of the noun in the set of noun terms that were created at the same time. This parameter deals with the value of the term in the changeset[8]. The second parameter is the frequency of the term in the corresponding source code file. The third parameter deals with the frequency of appearances of the term in the corpus. This parameter counts the number of files in the corpus that contain the desired noun term. Finally, the fourth parameter is time interval between time when the new change request was reported and time of commit of the desired noun term in the source code file (cf. Section 4.2.2).

4.4.2.2 Developer-based Term Weighting

In the second function of the Term Weighting component, the weight of the term is determined from the aspect of the developer who used the term in the source code file. The term-weighting technique used in this function is the same technique which was explained for the ***De***NoFeLo method. As mentioned in Section 4.3.2, in ***De***NoFeLo, the

[8]A changeset is an atomic set of changes of the source code files committed to the VCS, and contains the part of code that modified in a new revision.

TADTW term-weighting technique is used to adjust the value of the term from the aspect of developer. In this term-weighting technique, the weight of term is calculated based on the frequency of appearances of the term in the developer's expertise profile on a source code file and the last time that the developer used the term in the file. Three parameters are considered to determine the weight of term in TADTW. The first parameter is the frequency of appearances of the term in the set of terms that were used by the developer on the desired source code file. This parameter calculates the number of times that the term appeared in the developer's expertise profile on the source code file. The second parameter is the frequency of the appearances of the term among all the developers' expertise profiles. This parameter determines the value of the term in the corpus. In other words, this parameter determines the values of the keywords in the corpus. Finally, the last parameter is time interval between time when the new change request was reported and last time of usage of the term by the developer in the file. The details of this term-weighting technique have been described in Section 4.3.2.

4.4.2.3 Total Term Weighting

As mentioned above, for each term that appeared in both of the file and the change request, two different weights, i.e. time-based and developer-based weights, are calculated using TATW and TADTW techniques, respectively. Thus, the total weight of the term is the combination of these two calculated weights. Since the calculated weights through TATW and TADTW have different value ranges, the weights therefore need to be normalized for combination. A common technique for normalizing numbers from different ranges is to map them to the same range, such as a range between zero and one (Zhou et al., 2012). This mapping of values is done using Equation 4.6. In this equation, *Weight(max)* and *Weight(min)* are the maximum and minimum weights obtained for the noun terms in each function of the Term Weighting component.

$$NormalWeightN_i \quad \frac{WeightN_i - Weightmin}{Weightmax - Weightmin}$$

$$0 \leq NormalWeightN_i \leq 1 \tag{4.6}$$

After normalizing the weights of the noun terms, the total weight of the noun i ($TotalWeightN_i$) is the summation of the normalized time-based weight and the normalized developer-based weight. The obtained total weight for each term is between zero and two.

4.4.3 Location Identification Component

Having computed the total weights of the common noun terms that appear in both the new change request and the source code file, the score of the source code file with respect to the change request is calculated by Equation 4.7.

$$ScoreF_k \quad {}^{\#Common\ Nouns}_{i1} \, TotalWeightN_i, F_k \tag{4.7}$$

The output of this component is a ranked list of source code files in descending order, based on the calculated score for the file. Therefore, the most relevant locations for the new change request are ranked at the top.

4.5 Summary

The effort spent on addressing the objectives of this thesis results in proposing two new methods, ***Ti***NoFeLo and ***De***NoFeLo. From the combination of these two proposed methods, a new feature location approach, ***TiDe***NoFeLo, is proposed that considers a set of specific characteristics of text data recorded in the repository. In Table 4.2, the main properties of these proposed methods and the proposed approach are highlighted.

Table 4.2: Main properties of the proposed methods and the proposed approach

Method/ Approach	**Perspectives**	**Properties**
***Ti*NoFeLo Method**	- Analyzing text data from the aspect of time - Reduce the dataset size by using only the noun terms	- Analyze data from the time-aspect - Use time-based corpus - Use time-based term-weighting technique - Use only noun terms
***De*NoFeLo Method**	- Analyzing text data from the aspect of developer - Reduce the dataset size by using only the noun terms	- Analyze data from the developer-aspect - Use developer-based corpus - Use developer-based term-weighting technique - Use only noun terms
***TiDe*NoFeLo Approach**	- Analyzing text data from the aspect of time - Analyzing text data from the aspect of developer - Reduce the dataset size by using only the noun terms	- Analyze data from both the aspects of time and developer - Use time- and developer-based corpus - Use time- and developer-based term-weighting technique - Use only noun terms

As shown in Table 4.2, ***Ti***NoFeLo method - the first proposed method in this study – analyzes the data from the aspect of time when the data was recorded in the repository. In this method, only the noun terms are used to identify the related locations to a newly reported change request in order to reduce the dataset size. The ***Ti***NoFeLo method locates the change requests using three main functional components, i.e. Data Collection and Corpus Creation, Term Weighting, and Location Identification. In the Data Collection and Corpus Creation component, the required data are collected from the VCS of the software system and preprocessed into a time-based corpus. In this component, from the recorded data in the VCS, only the noun terms are extracted and linked to the corresponding metadata of time when the terms were created in the corresponding source code file. The next component, Term Weighting, uses a time-based term-weighting technique to determine the values of the noun terms based on the frequency of appearances in the corpus and time of using the term in the corresponding source code file. Finally, in the last component of ***Ti***NoFeLo, the source code files are indexed in descending order based on the sum of the calculated weights for the common terms that appeared in both the source code file and the desired change request. The higher the file in the ranked list, the more similarity it has with the given change request.

The second proposed method, ***De***NoFeLo, improves the text analysis of feature location by analyzing the text data from the aspect of developer and using only the noun terms. Similar to the ***Ti***NoFeLo method, ***De***NoFeLo relies on three main components to identify related source code locations for the new change request. With a few differences, the functionality of these components is similar to that of the corresponding component in the ***Ti***NoFeLo method. The first component, Data Collection and Corpus Creation, collects the required data from the VCS, and then, preprocesses and extracts the noun terms to create a developer-based corpus. In this corpus, the noun terms are linked to the corresponding developer-metadata and time-metadata. In the next component, Term-Weighting component, the value of the term stored in the developer-based corpus is determined based on the frequency of usage of the term by the developer in the corresponding source code file and the last time of usage of the term by the developer. Finally, in the last component, the scores of source code files are calculated by summing up the determined value for the term that appear in both the developers' expertise profile of the source code file and the new change request. Finally, the source code files are indexed in descending order based on the calculated scores.

From the combination of two proposed methods, ***Ti***NoFeLo and ***De***NoFeLo, a new feature location approach is proposed, named ***TiDe***NoFeLo. In this approach all identified characteristics of the text data recorded in the repository are considered to improve the accuracy of location identification process. The proposed approach analyzes the data from both the aspects of time and developer, and also uses only the nouns. The proposed approach, similar to both the proposed methods, includes three main components to determine the similarity of the source code files with the new change request. The required data are collected from the VCS in the Data Collection and Corpus Creation component and preprocessed to extract and refine the noun terms. Then, the noun terms are linked to the corresponding metadata of developers and time. Next, in the Term Weighting component, the noun terms that appear in both the source code file and given change request are weighted. Since ***TiDe***NoFeLo is the combination of ***Ti***NoFeLo and ***De***NoFeLo, the terms are weighted through two aspects, i.e. the aspect of time and the aspect of developer, which means that two different weights are calculated for each noun term. The calculated weights are combined to obtain the total weight of the term. The summation of the total weights of the common noun terms is used to calculate the score of the file in the Location Identification component. Finally, the files are ranked based on the scores of the files in descending order.

The proposed methods, ***Ti***NoFeLo and ***De***NoFeLo, and the proposed approach, ***TiDe***NoFeLo, are experimentally evaluated to assess the effects of analyzing the data from the aspect of time and aspect of developer. Furthermore, the impact of using only the noun terms is assessed in the experimental evaluation. Accordingly, in Chapter 5, the setting that needs to be considered in the setup of an experimental evaluation for evaluating the proposed methods and the proposed approach are explained in detail. Then, in Chapter 6, the results of the experimental evaluation of the ***Ti***NoFeLo and ***De***NoFeLo methods and the ***TiDe***NoFeLo approach are presented and analyzed.

CHAPTER 5: EXPERIMENTAL EVALUATION SETUP

In software engineering, one of the typical methodologies to evaluate an approach is experimental evaluation. To conduct an experiment, a set of specific settings needs to be identified and setup in relation to the approach. The most well-known guideline for experimental evaluation in software engineering is recommended by Wohlin et al. (Wohlin et al., 2012). Accordingly, this research follows the same organization that is recommended in Wohlin's guideline to conduct the empirical evaluation. Figure 5.1 shows the steps of experimental evaluation setup including context selection, experimental design, research questions and hypotheses formulation, experimental execution, and threats to validity. These steps are explained in detail in the rest of this chapter.

The first part of this chapter presents the evaluation context including subject systems (Section 5.1.1), object systems (Section 5.1.2) and comparison systems (Section 5.1.3).

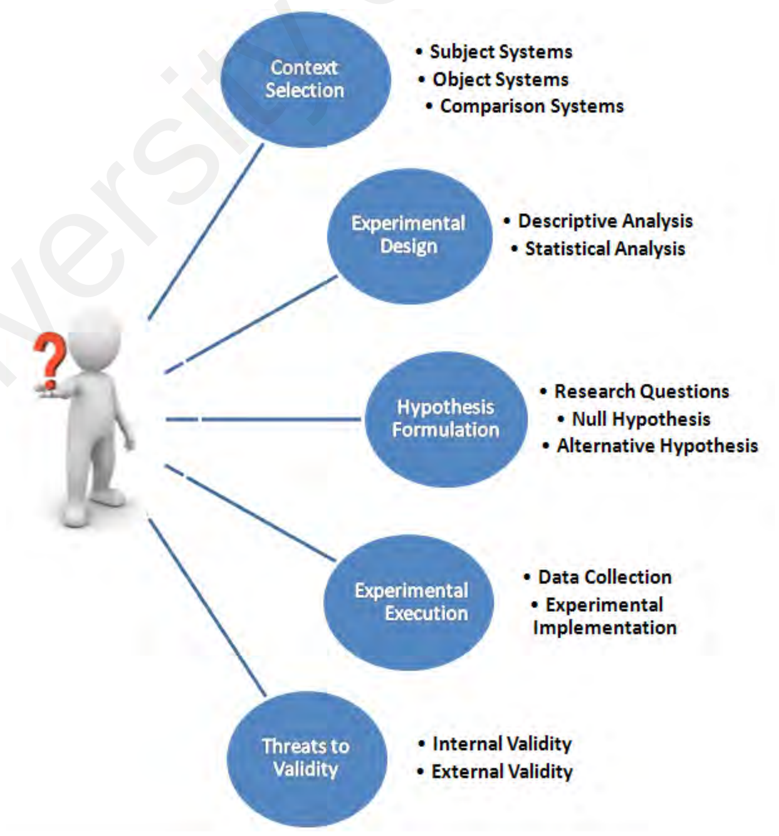


Figure 5.1: Evaluation setup steps

Then, the experimental design including the descriptive and statistical analysis setups is explained in Section 5.2. Next, according to the main research question that were identified in Section 1.4, a set of research questions and corresponding hypotheses are formulated specifically for each of the proposed methods and approach in Section 5.3. After that, how the data was collected from the subject systems and preprocessed as well as the implementation of the experiments are explained in Section 5.4. Finally, the internal and external threats to validity of the proposed methods and approach are presented in Section 5.5.

5.1 Context Selection

This section first presents the methodology by which the four open-source projects were selected as the subject systems. Next, the process of choosing the test sets from the subject systems is explained. Finally, the last part of this section explains how the existing feature location methods/approaches are chosen as the baseline feature location approaches against the proposed methods and the proposed approach.

5.1.1 Subject Systems

To evaluate the proposed methods and the proposed approach, a set of real-world projects is needed. Thanks to the growing popularity of open-source software projects, researchers are now able to use the information available in the repositories of these projects. Thus, due to the availability of data from open-source projects, such projects were investigated to identify the subject systems for the evaluation of the proposed methods and approach. Although, there are many open-source projects that could be used, the subject systems were selected based on the following criteria:

- Projects that have been used by other feature location or software repository mining researchers. Such projects have already been considered valid subject systems by other researchers.

- Projects that have different sizes of dataset in their repository. It is assumed that the number of files, commits, and fixed change requests in a project's software repositories gives an indication of the scale of the project.
- Projects that have different evolution speeds. As mentioned previously, time-metadata is one of the important factors in the proposed methods and approach. It is assumed that the size of data over a project's lifetime is an indication of the project's evolution speed, and therefore, the amount of data over the period of the project's lifetime will affect the value of the source code locations.
- Projects that have different number of developers. As mentioned earlier, developer-metadata is another important factor in the proposed methods and approach. By selecting projects with different number of developers, the impact of number of developers on the accuracy of the method and approach that rely on the developer-metadata will be evaluated.

Based on these criteria, four subject systems were chosen as the subject systems:

- **JDT**[1]: The JDT project provides the tool plug-ins that implement a Java integrated development environment (IDE).
- **AspectJ**[2]: The AspectJ project is a simple and practical extension of the Java programming language that adds aspect-oriented programming (AOP) capabilities to Java.

[1]http://www.eclipse.org/jdt/

[2]http://www.eclipse.org/aspectj/

- **Netbeans**[3]: The Netbeans project is an IDE for Java development. It also supports other languages, in particular PHP, C/C++, and HTML.
- **Rhino**[4]: The Rhino project is an implementation of Java-Script developed entirely in Java and is typically embedded into Java applications to provide scripting capability to end-users.

Some of the important metrics of these subject systems, such as the numbers of files, developers and change requests, are shown in Table 5.1. According to the assumption that the size of data represents the scale of the project, JDT, AspectJ, Netbeans, and Rhino are treated as large (JDT), medium (Netbeans) and small-scale (AspectJ and Rhino) projects. On the other hand, the different average sizes of changes per day for the subject systems indicate differences in their rates of project evolution. Therefore, the effect of time in weighting the terms for projects with different speeds of evolution would be evaluated. According to the properties of the selected subject systems, it is expected that, using these subject systems, the applicability of the proposed methods and the proposed approach on different scales of projects, different number of developers and different evolution speed will be shown.

To evaluate the proposed methods and approach, the information recorded in the VCS[5] of the selected subject systems is used as dataset. Also, a set of change requests which were recorded in the ITS [6] repository of each of the subject systems were selected as the test set. The details of the test set selection are presented in the next section.

[3]https://netbeans.org/

[4]https://developer.mozilla.org/en/docs/Rhino

[5]Version Control System or source code repository

[6]Issue Tracking System or bug tracking repository

Table 5.1: Properties of the subject systems

Property	JDT	AspectJ	Netbeans	Rhino
First Commit	2001-05-03	2002-12-17	1999-02-04	1999-04-19
Last Commit	2011-12-15	2012-01-04	2010-06-25	2012-05-04
# of Java Files	8308	5998	1375	422
# of Commits	162321	37232	67216	9079
# of Developers	58	8	175	18
Avg. of Changes per Day	48.8	11.26	16.16	1.9
# of Reported Change Requests	47265	2554	185578	834
Avg. of Change Requests per Day	12	0.77	44.62	0.17
# of Fixed Change Requests	21466	1598	69651	617
Avg. of Fixed Change Requests per Day	5.5	0.48	16.74	0.13

5.1.2 Object Systems

As mentioned above, a set of change requests which were reported to each subject system are selected to evaluate the proposed methods and the proposed approach using the subject systems. Typically, all reported change requests are recorded in the Issue Tracking System (ITS) of the projects. ITS is a software repository that records and tracks change requests reported to the project. To select the test sets of change requests for experimental evaluation of the proposed methods and approach, all the change requests reported to ITS of the selected subject systems were investigated. From the project's ITS, only the change requests marked as FIXED were collected. Since, this set of change requests is already resolved, a set of source code locations are modified to satisfy the requests. Thus, this set of source code locations are considered as the oracle set[7]. Accordingly, the fixed change requests of subject systems are investigated to select suitable test sets.

The investigation of feature location literature showed that there is no standard for the optimal test set size for evaluating a feature location approach. In most of the previous feature location research studies, a few change requests from a specific release, meaning a specific period of project lifecycle, were selected as a test set to evaluate an approach

[7]Oracle set is the answer set for the test set. This set is used as the control to find how accurately the approach can find the source code locations for the change request which exists in the test set.

(Poshyvanyk et al., 2007; Bacchelli et al., 2010). As discussed in Chapter 1, the change requests reported in different periods of the project lifecycle may have different goals and requirements. Therefore, to avoid evaluating the approach in a specific period of the project lifecycle where confounding conditions may occur and also to assess the applicability of the proposed methods and approach at different periods of project lifecycle, the change requests were randomly chosen from all of the fixed change requests of subject systems as the test set. Furthermore, randomly selecting the change requests for the test sets will demonstrate the overall performance of the approach on the subject systems. The number of randomly selected change requests in this study is 200 change requests. This value is chosen as it is more than that used by Poshyvanyk et al. (Poshyvanyk et al., 2007) (three from Eclipse and five from Mozilla) but less than that used by Zhou et al. (Zhou et al., 2012) (3000 from Eclipse).

To select the test set from the fixed change requests, first, a list of IDs of the fixed change requests was created using a search on the status of the reports in the ITS repository. Then, the corresponding change requests were downloaded and stored in XML format. The benefit of downloading XML format of the change request is having specific tags for each entity of change request that simplify the process of analyzing the content of the request. In the next step, there is a need to link the fixed change request to the set of source code locations that were modified to satisfy the requests. There is no default link between the change requests and the modified source code locations, due to the disconnection of the software repositories during the project lifetime. Thus, the actual source code locations which are modified to support the fixed change request are not known. Accordingly, to find the modified locations that make the oracle set for the test set, the fixed change requests need to be assigned and linked to the corresponding source code locations.

To link the fixed change requests to the corresponding source code locations, a number of ways are used (Anvik et al., 2006; Corley, Kraft, Etzkorn, & Lukins, 2011; Shokripour et al., 2013). The first way is from the patches[8] attached to the change requests. In this way, the patches are examined to extract the source code location information. If the change request does not have an attached patch, the next way is investigating the comments of change request that refer to the specific commit ID[9] in the VCS[10] of the software project. Typically, when the developer resolves a change request, she/he puts the ID number of commit in the comment of the change request in the ITS. If none of the previous ways leads us to the location of the change request, the commit messages[11] in the log of the project's VCS are used to determine the link. In this case, the commit messages are analyzed to find the change request's IDs that may appear in the messages. This way refers to the assumption that the developer may leave the ID of the change request in the commit message of the VCS when submitting changes to the projects. Although there are some keywords, such as 'fix' and 'bug', that developers typically use in conjunction with the change request IDs, there is no accepted convention.

To detect the bug IDs in commit messages and the commit IDs in the change request comments, which are both written in natural language, a set of text mining tools are employed to analyze the context. To detect these IDs in the commit messages or in the change request comments, a ruled-based Named Entity Recognition (NER) method was used (Sarawagi, 2008). To apply NER, the NE transducer component of ANNIE[12] plug-in

[8]A patch is a text file that contains the lines that have been removed and added to a source code file. In short, it is a text file that includes the modifications made to a specific file in the project. Patch is frequently used for updating of source code to a newer version.

[9]The action of saving the changes sent to a software project is known as committing the changes. Any changes in the source of project is stored in a unique commit that has a specific number as its ID.

[10]Version Control System

[11]For any change in the source of project, the developer who made the change should leave a message to explain the reasons of changing.

[12]http://www.aktors.org/technologies/annie/

of GATE[13] (Cunningham, Maynard, Bontcheva, & Tablan, 2002) was employed that uses JAPE[14] grammars to define the rules of detecting the specific entities. The rules are defined based on the results of the various steps of text analyzing such as part of speech (POS) tagger and morphological analysis, or a combination of these steps.

To find the identified rules in the context, the commit message, the comment of the change request, and the summary and description of the change request are analyzed by the text mining tools. Accordingly, first of all, the whole text was split into sentences by the sentence splitter component of ANNIE. Next, the tokenizer component was used to separate the terms and symbols into tokens. Then, POS tagger component was employed to determine the roles of the terms in the sentences. In this step, the change request ID numbers and the commit ID numbers are extracted from the context of commit message and change request comments, respectively, by using the identified rules in NE transducer component.

In the next step, the extracted ID numbers are compared with the collected bug report IDs to improve the confidence of the results obtained from NER. For ID which is matched with an ID in the list, the date of the commit is also compared with the creation and resolution date of the change request. For the case that the date of the commit is after the creation date of the change request or before the resolution date of the request, the user ID of the committer and user ID of the fixer of the change request are compared. It is common that the developers have different user names in the VCS and the ITS. To address this issue, we used a manually created mapping of user names from two systems when determining a match. In the case that all these comparisons indicate the correct match, the locations which are changed in the desired commit were linked to the change

[13]http://gate.ac.uk/

[14]Jolly And Pleasant Experience

request. From the fixed change requests that are linked to the correct locations, 200 change requests were randomly selected as the test set and their associated source code locations were considered as the oracle set. Appendix E lists the IDs of the selected change requests as the test set for each of the subject systems.

5.1.3 Comparison Systems

The results of the experiments that will be conducted for evaluating the proposed methods and the proposed approach need to be compared with existing feature location approaches. Accordingly, the feature location literature was reviewed to find suitable approaches or methods to assess different aspects considered in the proposed methods and approach. According to the recent feature location survey in (Dit et al., 2013), Information Retrieval (IR) is the most commonly used feature location method for analyzing the text data recorded in historical software repositories. Many models have been used in prior IR-based feature location research studies. The popular IR models, i.e. Smoothed Unigram Model (SUM) (Rao & Kak, 2011), Latent Dirichlet Allocation (LDA) (Lukins et al., 2010), Latent Semantic Indexing (LSI) (Poshyvanyk et al., 2007), and Vector Space Model (VSM) (Zhou et al., 2012), are described in Appendix B.

The performance and accuracy of these IR models for locating software features were evaluated by Rao and Kak (Rao & Kak, 2011), Zhou et al. (Zhou et al., 2012), and Wang et al. (S. Wang et al., 2011). According to these research studies, SUM and VSM obtained the highest accuracy among IR-based feature location methods. Correspondingly, to evaluate the proposed methods and the proposed approach, SUM and VSM are selected for comparison. These two models are briefly explained again as follows.

Vector Space Model (VSM): VSM analyzes text documents in a corpus and measures their similarity with a query in three steps. First, VSM makes a vector of the terms appeared in each of the documents. Next, the weights of the terms are determined for each document. Different term-weighting techniques have been used, with TF-IDF being the most common (Salton, Wong, & Yang, 1975; Manning et al., 2008). Lastly, the cosine similarity of document vectors against a vector for the query is calculated and the documents are indexed based on their similarity with the query. To implement VSM, an experiment was created in TraceLab[15] (Dit, Moritz, & Poshyvanyk, 2012). TraceLab is a recently published experimentation framework which is used to evaluate feature location approaches. In this thesis, an experimental approach which was implemented by Dit et al. (Dit et al., 2012) for utilizing VSM with a TF-IDF term-weighting technique is used. Hereafter, VSM refers to the VSM approach that utilizes the TF-IDF term-weighting technique.

Smoothed Unigram Model (SUM): Unigram Model (UM) is a simple case of a language model where the probability of each term is determined independent of other terms. UM uses the term count frequencies in each document for calculating the probabilities. The problem of UM assigning a zero probability to a class that is missing one term of a query (Witten & Bell, 1991) is resolved in Smoothed Unigram Model (SUM) by measuring the probability of a term in all documents instead of a single document (Bai, Nie, & Paradis, 2004). In this study, an experiment of SUM with the Dirichlet smoothing method (Zhai & Lafferty, 2004) implemented in Lingpipe[16] API is used to conduct the evaluation. Hereafter, SUM refers to the SUM approach with the Dirichlet smoothing method.

Applying the metadata, i.e. time, in the text analysis process of feature location results in proposing new term-weighting techniques. To evaluate the impact of considering the metadata in term-weighting process, the proposed methods are evaluated in the

[15]http://coest.org/coest-projects/projects/tracelab.

[16]http://alias-i:com/lingpipe/index:html

case of using the proposed term-weighting techniques in compared to the case of using the baseline term-weighting technique. The popular term-weighting techniques are explained in Appendix C. The TF-IDF technique is the most commonly used term-weighting technique in IR-based feature location methods. This technique is explained in details in Appendix B. TF-IDF deals only with the frequency of appearances of the term in the document and corpus. Thus, comparison of the proposed methods in the case of using the proposed term-weighting techniques in comparison to the case of using the TF-IDF technique reveals the impact of metadata consideration in location identification process.

Furthermore, to assess the other possible impacts of consideration of time-metadata and developer-metadata in the proposed methods and approach, a set of evaluations are conducted. In the rest of this section, the specific comparisons that are performed to evaluate each of the proposed methods and approach are explained.

Ti*NoFeLo:** In light of using time-metadata for feature location in this method, the ***TiNoFeLo method is evaluated both with and without the use of time-metadata. Since time-metadata is used in term-weighting process, ***Ti***NoFeLo is evaluated in the case of using the TATW technique in compared to the case of using the TF-IDF technique. To conduct this evaluation, first, a version of ***Ti***NoFeLo using the TATW technique is applied to the subject systems (***Ti***NoFeLo). Then, TATW is replaced by the TF-IDF technique, and the same experiment is performed (***Ti***NoFeLo$_{TF-IDF}$).

On the other hand, another contribution of this study is the use of only the noun terms in locating a change request. As mentioned earlier, all the proposed methods and the proposed approach are restricted to use only the noun terms. Since, the ***Ti***NoFeLo method is the first case that uses only the noun terms for feature location, the impact of noun usage is evaluated for this method and the results of this evaluation is used for the next step of this study. To evaluate the use of only noun terms, first, all of the terms were extracted from the text resources in the software repository and used in the ***Ti***NoFeLo method, which is referred as ***Ti***NoFeLo$_{All-Term}$. Then, only the noun terms are extracted from text data and the same experiment was run using those terms.

To summarize all the comparisons that are conducted to evaluate the ***Ti***NoFeLo method, the results of ***Ti***NoFeLo are compared with the results of the approaches from SUM, VSM, ***Ti***NoFeLo$_{All-Term}$, and ***Ti***NoFeLo$_{TF-IDF}$ in order to evaluate the different aspects of the proposed method.

De**NoFeLo:** To evaluate the impacts of developer and time consideration in the ***De***NoFeLo method, the results of ***De***NoFeLo are compared with the first proposed method, ***Ti***NoFeLo. Since the way of dealing with the time in ***De***NoFeLo is different from that of ***Ti***NoFeLo, the impact of time consideration is evaluated again for the ***De***NoFeLo method. To assess the impact of considering time in a developer-based feature location method, the results of ***De***NoFeLo are compared with a version of the method that does not take into account the time-metadata of developers' activities. This version of the method is referred to as ***De***NoFeLo$_{No-Time}$.

Furthermore, as it is mentioned in Chapter 4, in regards to the importance of the keyword terms in analyzing the terms even they were used further in the past, the inverse of developer frequency is considered in addition to the time difference. Accordingly, the results of ***De***NoFeLo are compared with a version of the method that does not have this parameter and is referred to as ***De***NoFeLo$_{No-Keywords}$. The term-weighting technique which is used in ***De***NoFeLo$_{No-Keywords}$ is similar to the TF-IDF technique. Thus, this comparison evaluates ***De***NoFeLo in the case of using keywords with the baseline term-weighting technique, TF-IDF. However, instead of using the name of ***De***NoFeLo$_{TF-IDF}$, the name of ***De***NoFeLo$_{No-Keywords}$ is used to emphasize on the main goal of this comparison that focuses on analyzing the impact of keyword consideration in the proposed method.

To summarize the comparison system for the ***De***NoFeLo method, the results of this method are compared with the results of the approaches from SUM, VSM, ***Ti***NoFeLo, ***De***NoFeLo$_{No-Time}$, and ***De***NoFeLo$_{No-Keywords}$ to evaluate the different aspects in the proposed method.

TiDe**NoFeLo:** Since the impacts of consideration of noun, time and developer are investigated in the evaluation of ***Ti***NoFeLo and ***De***NoFeLo methods; the impacts of considering these factors are not further evaluated for the ***TiDe***NoFeLo approach. Thus, here only the impact of combination of two proposed methods in the proposed approach, ***TiDe***NoFeLo, is evaluated. Accordingly, the results of the proposed approach are compared with the results of the ***Ti***NoFeLo method and the ***De***NoFeLo method. Furthermore, the results of ***TiDe***NoFeLo are assessed against the feature location baseline approaches. Accordingly, the proposed approach is assessed over the time-based feature location method, ***Ti***NoFeLo, the developer-based method, ***De***NoFeLo, and the location-based approaches, SUM and VSM.

5.2 Experimental Design

After conducting the experiments of evaluating the proposed methods and approach on the selected test sets from each of the subject systems, the obtained results are analyzed from two aspects, i.e. descriptive analysis and statistical analysis. In descriptive analysis, the results of the evaluation are assessed using a set of metrics presented below, and the statistical analysis further analyzes the results using the statistical tests and the effect size.

5.2.1 Descriptive Analysis

The proposed methods and the proposed approach are evaluated on the subject systems using the selected test sets. The selected tests sets are considered as the independent variables in the experiments. For experimental evaluation, a set of metrics were identified that are considered as the dependent variables in this experiment. The identified metrics used in this research are as follows:

- **Top N Rank or Likelihood** (Rao & Kak, 2011; Zhou et al., 2012): For a new change request, if top N ranked results contain at least one source code location where the change request should be fixed, it is counted as a correct answer. This metric is used to evaluate the accuracy of a feature location approach. In this case, the higher the value of the metric, the better the accuracy of the feature location approach.

- **Effectiveness** (Liu et al., 2007; Poshyvanyk et al., 2007): In feature location, effectiveness is defined as the position of the first relevant source code location in the ranked list. Those approaches that rank relevant locations near the top of the list are deemed more effective because they reduce the number of false positives a developer has to consider. For this metric, the lower the value, the less effort is required by the developer, leading to a more effective feature location approach.

- **Mean Reciprocal Rank (MRR)** (Baeza-Yates & Ribeiro-Neto, 1999): The reciprocal rank is the inverse of the rank position of the first relevant location. In fact, it is the inverse of the effectiveness metric. MRR is the average of the reciprocal ranks of a set of queries and is calculated using Equation 5.1. In this equation, Q is the number of queries in the test set, i.e. the number of change requests used for testing. The higher MRR value, the better the performance of the feature location approach.

$$MRR \quad \frac{1}{|Q|} \, {}^{|Q|}_{i1} \, \frac{1}{Rank_i} \tag{5.1}$$

- **Mean Average Precision (MAP)** (Baeza-Yates & Ribeiro-Neto, 1999): MAP is the mean of the average precisions for a set of queries. Precision is the fraction of predicted source code locations that are relevant to the given query. It is calculated using Equation 5.2. *P(k)* is the precision at the given rank *k*. The higher MAP value, the better the feature location performance.

$$Pk \quad \frac{NumberofPositiveInstancesInTopkPositions}{k} \tag{5.2}$$

5.2.2 Statistical Analysis

The descriptive analysis of whether one feature location approach outperforms another one is not enough. A statistical analysis is also needed to determine whether the difference between the obtained results is significant or not. Due to the sufficiency of the number of outputs for the accuracy (10 outputs) and effectiveness (more than 60 outputs) metrics, the statistical analysis is conducted for these two metrics. For two other identified metrics, MRR and MAP, the evaluation generates only one output that may not be statistically analyzed. To analyze the accuracy, the results from recommending Top1 to Top10 locations are used. For the effectiveness metric, the results of the first relevant location for all change requests in the test sets for a subject system is used. The output of the accuracy and effectiveness metrics are respectively, the scale and the ordinal variable types.

To identify a suitable statistical test, dependency of the data, whether results are paired or unpaired, and the normality of the results must be determined (Wohlin et al., 2012). As this research is dealing with paired data, statistical tests that work on the paired data are needed. To evaluate the normality of distribution of the results, the Shapiro-Wilk test based on $\alpha = 0.05$ is used. A p-value of the Shapiro-Wilk test that is more than α indicates that the data follows a normal distribution. Furthermore, the values of Skewedness and

Kurtosis are investigated. If the values of Skewness and Kurtosis are between -2 and 2, the data is considered normally distributed.

For the results that were found as the normally distributed, the paired T-test is used for statistical comparison of two approaches. In the case of comparing multiple results, the two-way ANOVA was conducted and a Least Significant Difference (LSD) test was used as the post hoc test. To statistically analyze the case that an approach is evaluated with or without one or two factors, a repeated measure ANOVA test -with a Greenhouse-Geisser correction- is conducted. For results that are not normally distributed, the Wilcoxon test was conducted for paired data comparison and the Friedman test was used for comparing multiple results. The significance level of the tests is $\alpha = 0.05$, meaning that if the p-value is lower than 0.05, the difference is statistically significant. In the case that the p-value is equal to α, it can be said that the null hypothesis is rejected with 95% confidence, obtained by subtracting 1 from α $(1 - \alpha)$.

In addition to assess the statistical significance (the test's p-value), the effect size of the difference between the approaches is reported to complement the inferential statistics. Unlike the statistical significance, effect size is not influenced by the statistical test type and the sample size. In this research, for the normally distributed data, the effect size of the Hedges' g is used, and for the non-normal distributed data, the Cliff's delta is calculated[17]. Hedges'g is a more accurate version of Cohen's d that adds a correction factor for small sample sizes. The Hedges' magnitude is performed using the thresholds provided by Cohen (Cohen, 1992). The thresholds are $|g|<0.2$ for "negligible", $|g|<0.5$ for "small", $|g|<0.8$ for "medium", and "large" for $|g| \geq 0.8$. The magnitude of the Cliff's delta is assessed using the thresholds provided by Romano (Romano, Kromrey, Coraggio, & Skowronek, 2006) that consider |delta|<0.147 as "negligible", |delta|<0.33 as "small",

[17]http://softeng.polito.it/software/effsize/

|delta|<0.474 as “medium”, and |delta|≥ 0.474 as “large”.

5.3 Research Questions and Hypotheses Formulation

As mentioned in Chapter 1, the main research question of this thesis investigates whether consideration of specific characteristics of the text data within the text analysis process make an improvement in feature location accuracy. To address this research question and to be more specific, in evaluation of each of the proposed methods and the proposed approach, a set of research questions along with the contribution of the proposed methods and the proposed approach are identified in this section. Corresponding to these specific research questions, a set of null and alternative hypotheses are formulated that need to be statistically analyzed to demonstrate possible significance of the differences when dealing with time-metadata, and developer-metadata and also using only the nouns.

In the remaining parts of this section, first in Section 5.3.1, specific research questions for the ***Ti***NoFeLo method and the corresponding null and alternative hypotheses are formulated. After that, the specific research questions and hypotheses along with the research questions are formulated for the ***De***NoFeLo method and the ***TiDe***NoFeLo approach, in Sections 5.3.2 and 5.3.3, respectively.

5.3.1 Research Questions and Hypotheses of ***Ti*****NoFeLo Method**

As mentioned in Section 5.1.3, a set of evaluations and comparisons needs to be performed to assess the robustness of the ***Ti***NoFeLo method. The main goal of these evaluations is finding the answers of the following questions:

- **Ti.RQ**$_1$: Does ***Ti***NoFeLo outperform SUM and VSM, as the baseline feature location approaches?
- **Ti.RQ**$_2$: What is the impact of the use of time-metadata in the term-weighting

process for feature location?

- **Ti.RQ**$_3$: What is the impact of using only noun terms instead of using all types of terms in feature location process?

These research questions led to identifying a set of null hypotheses which are presented in Table 5.2.

Table 5.2: Null Hypotheses of ***Ti***NoFeLo method

$\mathbf{H}_{0,TiNoFeLovs.SUM}$	There is no significant difference between the results of ***Ti***NoFeLo and SUM.
$\mathbf{H}_{0,TiNoFeLovs.VSM}$	There is no significant difference between the results of ***Ti***NoFeLo and VSM.
$\mathbf{H}_{0,TATWvs.TF-IDF}$	There is no significant difference between the results of ***Ti***NoFeLo using TATW technique and using TF-IDF technique.
$\mathbf{H}_{0,All-Termsvs.Noun-Terms}$	There is no significant difference between the results of ***Ti***NoFeLo when using all types of the terms, and when using only noun terms.

If a null hypothesis can be rejected with high confidence (95%), the corresponding alternative hypothesis is supported. The alternative hypotheses are formulated in Table 5.3.

Table 5.3: Alternative Hypotheses of ***Ti***NoFeLo method

$\mathbf{H}_{a,TiNoFeLovs.SUM}$	There is significant difference between the results of ***Ti***NoFeLo and SUM.
$\mathbf{H}_{a,TiNoFeLovs.VSM}$	There is significant difference between the results of ***Ti***NoFeLo and VSM.
$\mathbf{H}_{a,TATWvs.TF-IDF}$	There is significant difference between the results of ***Ti***NoFeLo when using TATW technique and using TF-IDF technique.
$\mathbf{H}_{a,All-Termsvs.Noun-Terms}$	There is significant difference between the results of ***Ti***NoFeLo when using only noun terms as its input and the results of the method when using all types of terms.

Statistical testing of these hypotheses demonstrates possible significance of the improvements, for the ***Ti***NoFeLo method. With regards to the identified metrics in Section 5.2.1, the null and alternative hypotheses for either accuracy or effectiveness metrics are derived analogously from the list of the hypotheses. As mentioned previously, the output of two other metrics, including MRR and MAP, is only one result that might not be statistically analyzed.

5.3.2 Research Questions and Hypotheses of *De*NoFeLo Method

The goal of evaluating the ***De***NoFeLo method is to address the following questions:

- **De.RQ**$_1$: Does DeNoFeLo outperform SUM and VSM, as the baseline feature location approaches?
- **De.RQ**$_2$: Does ***De***NoFeLo, which is a developer-based method, outperform ***Ti***NoFeLo as a time-based feature location method?
- **De.RQ**$_3$: What is the impact of consideration of time in ***De***NoFeLo as a developer-based method?
- **De.RQ**$_4$: What is the impact of consideration of keywords in ***De***NoFeLo as a developer-based method?

Extending from these research questions, a set of null hypotheses were formulated in Table 5.4. Statistical testing of these hypotheses was used to determine the significance of the improvements, if any.

Table 5.4: Null Hypotheses of ***De***NoFeLo method

$\mathbf{H}_{0,DeNoFeLovs.SUM}$	There is no significant difference between the proposed developer-based feature location method, ***De***NoFeLo, and SUM, as a feature location baseline approach.
$\mathbf{H}_{0,DeNoFeLovs.VSM}$	There is no significant difference between ***De***NoFeLo and VSM, as a feature location baseline approach.
$\mathbf{H}_{0,DeNoFeLovs.TiNoFeLo}$	There is no significant difference between ***De***NoFeLo as a developer-based feature location method and ***Ti***NoFeLo as a time-based feature location method.
$\mathbf{H}_{0,DeNoFeLovs.DeNoFeLoNo-Time}$	There is no significant difference between the proposed developer-based feature location method with time-metadata consideration, ***De***NoFeLo, and without time-metadata consideration, ***De***NoFeLo$_{No-Time}$.
$\mathbf{H}_{0,DeNoFeLovs.DeNoFeLoNo-Keywords}$	There is no significant difference between the proposed developer-based feature location method with keywords consideration, ***De***NoFeLo and without keywords consideration, ***De***NoFeLo$_{No-Keywords}$.

If a null hypothesis can be rejected with high confidence (95%), then the corresponding alternative hypothesis is supported. With respect to the identified null hypotheses, the alternative hypotheses for the ***De***NoFeLo method are formulated in Table 5.5.

Statistical testing of these hypotheses demonstrates possible significance of the improvements, for the proposed developer-based method.

Table 5.5: Alternative Hypotheses of ***De***NoFeLo method

$\mathbf{H}_{a,DeNoFeLovs.SUM}$	There is significant difference between the results of the proposed developer-based feature location method, ***De***NoFeLo, and SUM as the location-based feature location baseline approach.
$\mathbf{H}_{a,DeNoFeLovs.VSM}$	There is significant difference between the results of the proposed developer-based feature location method, ***De***NoFeLo, and VSM as the location-based feature location baseline approach.
$\mathbf{H}_{a,DeNoFeLovs.TiNoFeLo}$	There is significant difference between the results of the ***De***NoFeLo method and the ***Ti***NoFeLo method as a time-based method.
$\mathbf{H}_{a,DeNoFeLovs.DeNoFeLoNo-Time}$	There is significant difference between the results of the ***De***NoFeLo method when considering time-metadata compared with the case of no time-metadata consideration, ***De***NoFeLo$_{No-Time}$.
$\mathbf{H}_{a,DeNoFeLovs.DeNoFeLoNo-Keywords}$	There is significant difference between the results of the ***De***NoFeLo method when considering keywords compared with the case of no keywords consideration, ***De***NoFeLo$_{No-Keywords}$.

5.3.3 Research Questions and Hypotheses of ***TiDe*NoFeLo** Method

Since the ***TiDe***NoFeLo approach is embodied by ***Ti***NoFeLo and ***De***NoFeLo methods, the robustness of the proposed approach needs to be evaluated in relation to its components, ***Ti***NoFeLo and ***De***NoFeLo, in addition to comparing with the feature location baseline approaches. Accordingly, the following research questions are identified:

- **TiDe.RQ**$_1$: Does ***TiDe***NoFeLo outperform SUM and VSM as the location-based feature location approaches?
- **TiDe.RQ**$_2$: Does ***TiDe***NoFeLo outperform ***Ti***NoFeLo as a time-based feature location method?
- **TiDe.RQ**$_3$: Does ***TiDe***NoFeLo outperform ***De***NoFeLo as a developer-based feature location method?

Along with these questions, the following set of hypotheses was formulated in Table 5.6.

With respect to the identified null hypotheses, the alternative hypotheses for the ***TiDe***NoFeLo approach are formulated in Table 5.7.

Table 5.6: Null Hypotheses of ***TiDe***NoFeLo method

$\mathbf{H}_{0,TiDeNoFeLovs.SUM}$	There is no significant difference between the ***TiDe***NoFeLo approach and SUM as a location-based feature location approach.
$\mathbf{H}_{0,TiDeNoFeLovs.VSM}$	There is no significant difference between the ***TiDe***NoFeLo approach and VSM as a location-based feature location approach.
$\mathbf{H}_{0,TiDeNoFeLovs.TiNoFeLo}$	There is no significant difference between the ***TiDe***NoFeLo approach and ***Ti***NoFeLo as a time-based feature location method.
$\mathbf{H}_{0,TiDeNoFeLovs.DeNoFeLo}$	There is no significant difference between the ***TiDe***NoFeLo approach and ***De***NoFeLo as a developer-based feature location method.

Table 5.7: Alternative Hypotheses of ***TiDe***NoFeLo method

$\mathbf{H}_{a,TiDeNoFeLovs.SUM}$	There is significant difference between the results of the ***TiDe***NoFeLo approach and the SUM approach.
$\mathbf{H}_{a,TiDeNoFeLovs.VSM}$	There is significant difference between the results of the ***TiDe***NoFeLo approach and the VSM approach.
$\mathbf{H}_{a,TiDeNoFeLovs.TiNoFeLo}$	There is significant difference between the results of the ***TiDe***NoFeLo approach and the ***Ti***NoFeLo method.
$\mathbf{H}_{a,TiDeNoFeLovs.DeNoFeLo}$	There is significant difference between the results of the ***TiDe***NoFeLo approach and the ***De***NoFeLo method.

Rejection of a null hypothesis leads to acceptance of the corresponding alternative hypothesis. As mentioned previously, the null and alternative hypotheses for either accuracy or effectiveness metrics are derived from the list of the hypotheses.

5.4 Experimental Execution

In this section, the process of collecting the required data from the subject systems as well as the way to implement the proposed methods and the proposed approach are explained.

5.4.1 Data Collection and Preparation

As mentioned in Chapter 4, the main data resource which is used in this study to provide the required data for feature location is the source code of the projects. Moreover, the source code and its related information are recorded in the VCS (Version Control Repository) which is one of the most important repositories of the software projects. To collect and extract the required data from the VCS, some tools were employed.

First, the commit log was downloaded from the VCS of the subject systems. Different projects used different types of VCS repository tools, such as CVS[18], Git[19], and Mercurial[20]. The subject systems employed in this thesis used CVS (JDT and AspectJ), Git (Rhino), and Mercurial (Netbeans) as their version control systems. We used CVS-ANALY[21] to transfer the data from a project's VCS to a local repository. CVSANALY organizes this data based on the source code revisions which include the revision number, the commit date, and the commit message. Based on the source code revision numbers, the identifiers were extracted from the source code of the project.

In order to extract the identifiers, first of all, the source code files that were created or modified in each revision were regenerated using the appropriate VCS commands. Since, all the subject systems were written in Java programming language, a suitable tool was employed to extract the identifiers from the generated source code files. Therefore, JELDoclet[22] tool was used to convert the source code files to the JavaDoc XML format. This conversion makes it possible to extract the identifiers from the source code. From the generated XML files, the text entities of the <jelclass>, <fields>, <method>, and <params> tags are extracted to collect the name of classes, fields, methods, and method parameters, respectively.

The identifiers are typically a concatenation of a set of terms. To decompose an identifier, the approach recommended by Butler et al. (Butler et al., 2011) was used to produce a set of terms used to create the identifier. To extract the noun terms from the decomposed identifiers, ANNIE[23] plug-in of GATE[24] (Cunningham et al., 2002) was

[18]http://sourceforge.net/apps/trac/sourceforge/wiki/CVS

[19]http://sourceforge.net/apps/trac/sourceforge/wiki/Git

[20]http://sourceforge.net/p/forge/documentation/Mercurial/

[21]http://metricsgrimoire.github.com/CVSAnalY/

[22]http://jeldoclet.sourceforge.net/

[23]http://www.aktors.org/technologies/annie/

[24]http://gate.ac.uk/

employed. To determine the roles of the decomposed identifiers, part-of-speech (POS) tagger of ANNIE plug-in was used. The POS tagger was trained by a large corpus taken from the Wall Street Journal as a training set to make the lexicon and rule set which are used to tag the input terms. From the output of the POS tagger, the terms that are labeled as noun (such as NN and NNP) were extracted. Recall that both the complete identifier and the noun terms of the decomposed identifiers are used as identifier dataset in this study. For each source code file in the project, the corresponding identifiers and the noun terms of the decomposed identifiers were used to create the noun index. From the extracted identifiers in each revision of the source code file, any identifiers that were not already in the noun index of the associated source code file or developer were added. All of the entities in the noun index were associated with the corresponding source code file, time-metadata, and developer-metadata to provide the ability of analyzing the data from the aspects of location, time, and developer.

5.4.2 Experimental Implementation

The proposed methods, ***Ti***NoFeLo and ***De***NoFeLo, and the proposed approach, ***TiDe***NoFeLo, were implemented as TraceLab experiments. As mentioned previously, TraceLab[25] is a recently published experimentation framework which is used to evaluate the feature location approaches. TraceLab is used for creating, conducting, and sharing experiments on feature location. This framework was used to implement an experimental approach for each of the proposed methods and the proposed approach. Since, the implemented experiment in TraceLab for the proposed methods and approach is similar with a few differences, an overall description for the implemented TraceLab experiment for the proposed methods and approach is presented in the rest of this section.

Figure 5.2 shows an overview of the TraceLab experiment used to implement the

[25]http://coest.org/coest-projects/projects/tracelab

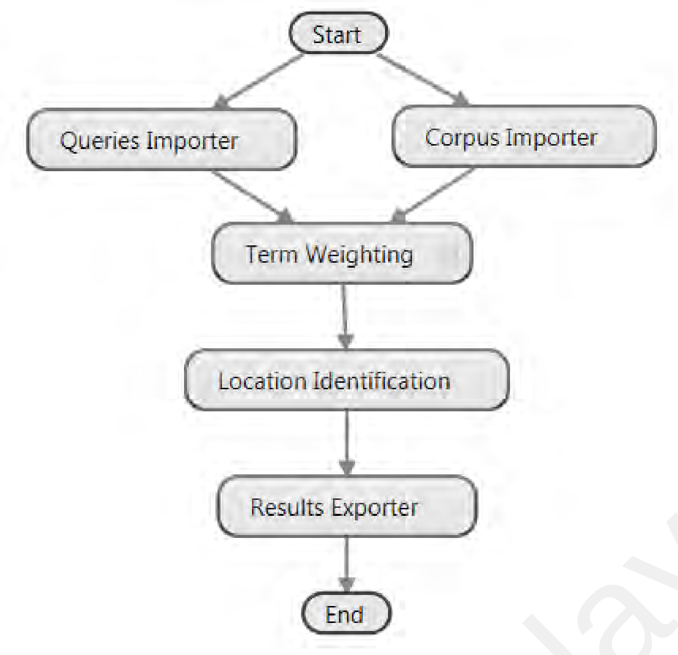


Figure 5.2: TraceLab experiment for the proposed methods and the proposed approach

proposed methods and approach. This experiment is available online[26] for readers wanting to replicate this work. The data collected from the source code files of each subject system was preprocessed and refined using the methods that were explained in Section 5.4.1. The output of this step is a set of identifiers and the noun terms of the decomposed identifiers with the corresponding time-metadata and developer-metadata extracted from the VCS. The terms along with their associated metadata, form the input of the TraceLab experiment. This data was imported using Corpus Importer and Queries Importer components, which were created by Moritz and his colleagues (Dit et al., 2012). The imported data was used to calculate the weights of the terms found in the source code files by Term Weighting component. These weights were used to determine the scores of the source code files in Location Identification component. The output of this experiment is a ranked list of files in descending order of the scores based on their relevancy for each change request in the test set.

The results of these experiments were evaluated using the identified metrics, and

[26]https://docs.google.com/file/d/0B0sa-hXpOgi JbjYyRzVTei1wZ2M/edit?usp=sharing

compared with the results of SUM, implemented in Lingpip[27] API, and VSM, implemented in TraceLab.

5.5 Threats to Validity

Presented in this section are some of the threats to the validity of this study, specifically threats to the Construct validity (factors that may affect method and approach accuracy), internal validity (factors that could affect the evaluation results) and threats to the external validity (factors that affect generalizing the evaluation results).

5.5.1 Construct Validity

The threats to the construct validity concerns the means that are used in the proposed methods and proposed approach which affect the accuracy assessment as a depiction of reality.

In the developer-based term-weighting technique, weight of a term may be equal to zero when the number of developers who used the term in the project is equal to the frequency of use of the term by a developer. This is due to the use of logarithm for the $\#PrjDevFreq_{N_i,D_j}$ expression. In this case, there is a need for a smoothing method to resolve the zero problem of the developer-based term-weighting technique. There are several smoothing methods (Zhai & Lafferty, 2004) that could be used to resolve the zero problem. One of the future works of this study is experimentally evaluating different smoothing methods to find the appropriate one that improves the developer-based feature location process as well as resolving the zero problem.

5.5.2 Internal Validity

The threats to the internal validity of this work concern the factors that could affect the evaluation results. First, in this study, it was assumed that the identifiers were appropriately

[27]http://alias-i:com/lingpipe/index:html

named, and that the developers adhered to proper programming practices when naming variables, methods, and classes. In this context, if project developers were to use non-meaningful names, the effectiveness of the proposed approach would be affected. To reduce the effect caused by poor naming practices by developers, only projects whose developers were judged to have generally followed good naming conventions were selected to be subject systems. Also, the subject systems all fall into the category of software development tools. Therefore, it is assumed that there is a high probability that the developers would follow good development practices.

Second, the presented approach is partly based on the content of the new change request for which source code locations were sought. If the content of the selected change request is of low quality, meaning that Summary and Description are not well-defined, then the proposed approach may not be able to identify the correct source code location. Moreover, if a change request does not provide sufficient information, or provides misleading information, the effectiveness of the approach would be adversely affected. However, through manual inspection of a random selection of change requests, it was found that the number of change requests with low quality or insufficient information was negligible.

Third, recall that POS tagger component of ANNIE plug-in was used for term categorization. ANNIE[28] is a plug-in of GATE[29] (Cunningham et al., 2002) which is an open-source software supported by the University of Sheffield since 1995. GATE is used by many researchers and updated to be able to solve almost any text processing problem. POS of ANNIE is a strange tool for categorizing the terms in sentences. However, this component was trained using data from the Wall Street Journal, a domain which is not

[28]http://www.aktors.org/technologies/annie/

[29]http://gate.ac.uk/

related to software engineering. This may have resulted in some categorization mistakes. Due to the important role of the noun terms in the proposed approach, the precision of POS tagger component in term categorization, especially noun determination, could potentially influence the results of the proposed methods and approach.

The last threat to internal validity is related to the text mining methods used to analyze the text resources. The text in the information resources does not always conform to proper grammar. Also, there exists some noisy text in the information resources, such as stack traces in change requests. This noise can cause the text mining methods to incorrectly determine the grammatical category of some terms. However, through manual inspection of a random selection of change requests, it was found that the number of incorrectly determined categories was negligible.

5.5.3 External Validity

External validity is concerned with whether or not the results of the evaluation can be generalized to other datasets besides the datasets used in the study. First, all the datasets used in this work were taken from open-source projects. The nature of the data from open-source projects may be different from that of the closed-source projects. However, the effectiveness and performance of the approach was assessed on four open-source projects that collectively are believed to be good representatives of both projects of different scales (large, medium and small) and projects with different evolution speeds. Despite this, it cannot be claimed that these results would be similar for all other open-source or commercial software projects.

Second, the software projects that were selected met all the factors determined for selecting the most suitable subject systems. All the subject systems were written in Java programming language, and they were all software applications that support software development. This means that all the subject systems fall into a single general software project domain. It is possible that the obtained results from examining these subject systems might be different from those found using projects from a different domain of software projects, such as systems for health care, transportation, or e-commerce. The evaluation of systems from the other domains might present new issues that are not present in software applications that support software development. However, it is believed that this possible difference is minimized by the use of GATE tool that treats different programming languages in an unbiased manner.

Lastly, the size of the evaluation test sets and the number of subject systems remain a difficult issue, as there is no accepted standard to follow. The common belief which is “more is better” may not necessarily yield a rigorous evaluation. In some cases, other noisy information in a project’s issue tracking repository could enter the data of a test set. If this issue is not addressed, it may lead to biased results that are positively or negatively skewed. In this work, 200 fixed change requests were randomly selected from each subject system. However, this data-set size is not as high as that used by Zhou et al. (Zhou et al., 2012) nor it is as low as that used by Poshyvanyk et al. (Poshyvanyk et al., 2007). It is believed that this test set size provides a good compromise between those two sizes of test set which are used by other works.

5.6 Summary

In this chapter, the details of the setup for evaluating the proposed methods and the proposed approach were explained. The main part of evaluation setup includes context selection, experimental design, research questions and hypotheses formulation, experimental execution, and threads to validity.

The context selection comprising the subject systems, object systems and comparison systems were described in Section 5.1. To determine the subject systems, the open-source projects were investigated and JDT, AspectJ, Netbeans, and Rhino projects were selected. From all change requests reported to each of the subject systems, 200 change requests were randomly selected as the object systems. To evaluate the proposed methods and approach, the results of the experiments are assessed and compared to the existing methods and approaches. Specifically, the results of ***Ti***NoFeLo method are compared with SUM, VSM, ***Ti***NoFeLo$_{All-Term}$, and ***Ti***NoFeLo$_{TF-IDF}$. Then, the results of ***De***NoFeLo method are assessed and compared to SUM, VSM, ***Ti***NoFeLo, ***De***NoFeLo$_{No-Time}$, and ***De***NoFeLo$_{No-Keywords}$. Finally, the results of ***TiDe***NoFeLo approach are compared with the results of the baseline approaches and the proposed methods, i.e. ***Ti***NoFeLo and ***De***NoFeLo.

Next, Section 5.2 presented the experimental design that analyzes the results of the experiments from the descriptive and statistical aspects. Descriptive analysis valuates the results based on the set of metrics, i.e. TopN Rank (accuracy or Likelihood), Effectiveness, Mean Reciprocal Rank (MRR), and Mean Average Precision (MAP). Then, the descriptive results are statistically analyzed from the aspects of normality of the results, significance of the differences between the methods and approaches, and the effect size. Accordingly, first, the normality of the results is evaluated using Shapiro-Wilk test and Skewedness and Kurtosis values. Based on the normality evaluation results, suitable comparative tests are used to assess the significance of the differences between two or more methods/approaches. In this study, T-test is used to compare two methods or approaches with normal distributed results. Repeated measure ANOVA test and two-way ANOVA test are used for comparison of multiple methods or approaches with normal distributed results. For non-normal distributed results, Wilcoxon test is used for comparison of two methods or approaches, and Friedman test is used for comparison of multiple methods or approaches. Finally for effect size assessment, Hedges' g and Cliff's delta are used respectively for normal and non-normal distributed results.

Then, in relation with the identified comparison systems, a set of specific research questions and hypotheses were formulated for each of the proposed methods and approach in Section 5.3. Next, how the data was collected from the subject systems and preprocessed as well as the implementation of the experiments are explained under the experimental execution in Section 5.4. Finally, in Section 5.5, the threats to validity of the proposed methods and the proposed approach including the internal and external validities are explained. Based on all the identified setup settings, a set of experiments are performed to evaluate the proposed methods and approach. In the next chapter, the obtained results from the experiments are reported and analyzed based on the descriptive and statistical analysis.

CHAPTER 6: EVALUATION RESULTS AND ANALYSIS

In this chapter, the impacts of considering the metadata of time and developer for locating the software features, as well as the effects of using only the nouns are assessed. Accordingly, the proposed methods, ***Ti***NoFeLo, ***De***NoFeLo, and the proposed approach, ***TiDe***NoFeLo, are experimentally evaluated. As mentioned in Chapter 5, experimental evaluation is one of typical methodologies to evaluate a proposed approach in software engineering research area. Following the recommended guideline by Wohlin and his colleagues, a set of experiments was conducted (Wohlin et al., 2012). The obtained results from the experiments are presented and analyzed in this chapter.

The experimental results of ***Ti***NoFeLo, ***De***NoFeLo, and ***TiDe***NoFeLo, are respectively presented in Sections 6.1, 6.2, and 6.3. In each of these sections, first, the evaluation results including the descriptive and statistical results are reported in separate sections. Then, the analysis of both the descriptive and statistical results is discussed. Finally, summary of the achievements and findings in the experimental evaluation are described in Section 6.4.

6.1 Evaluation of *Ti*NoFeLo Method

This section reports and analyzes the experimental results that measured the effects of time consideration and noun usage in the proposed method, ***Ti***NoFeLo. The settings for the experimentally evaluation of the ***Ti***NoFeLo method were explained in detail in Chapter 5. The results of the experiments are presented in two main parts including the descriptive results, Section 6.1.1, and statistical results, Section 6.1.2. In the descriptive analysis of the results, the obtained results are analyzed from the aspect of the identified metrics (See Section 5.2.1). After that, the obtained results are statistically analyzed in the next section. The detailed discussion on the descriptive and statistical results is presented in the last part of this section with respect to the identified research questions for the ***Ti***NoFeLo method.

6.1.1 Descriptive Results

The descriptive results of the experiments performed to evaluate ***Ti***NoFeLo method are reported in this section. The results are reported and analyzed based on the research questions identified for ***Ti***NoFeLo in Section 5.3.1. To remind, these research questions are represented in Table 6.1.

Table 6.1: Research questions of ***Ti***NoFeLo method

Ti.RQ1	Does ***Ti***NoFeLo outperform SUM and VSM, as the baseline feature location approaches?
Ti.RQ2	What is the impact of the use of time-metadata in the term-weighting process for feature location?
Ti.RQ3	What is the impact of using only noun terms instead of using all types of terms in feature location process?

According to the research questions, first, the results of ***Ti***NoFeLo are evaluated by comparing with the results of the feature location baseline approaches, i.e. SUM and VSM, in Section 6.1.1.1. Next, in Section 6.1.1.2, the proposed method is evaluated in terms of using time-metadata in the proposed term-weighting technique. In this case, the results of ***Ti***NoFeLo when using TATW technique are assessed versus the TF-IDF technique (i.e. ***Ti***NoFeLo versus ***Ti***NoFeLo$_{TF-IDF}$). Finally, the impact of using only the noun terms for feature location is analyzed in Section 6.1.1.3. Accordingly, the results of ***Ti***NoFeLo is assessed when using only the noun terms against using all types of terms (i.e. ***Ti***NoFeLo versus ***Ti***NoFeLo$_{All-Term}$).

6.1.1.1 ***Ti*NoFeLo versus Baseline Approaches**

This section presents the descriptive results of the ***Ti***NoFeLo method assessment compared to the feature location baseline approaches that use SUM and VSM. Table 6.2 shows the results of applying ***Ti***NoFeLo, SUM and VSM on the subject systems. The first two columns of this table list the subject systems and the metrics. To enhance the comparison process, the last two columns of this table show the differences between the results of ***Ti***NoFeLo, and SUM and VSM, respectively. The results obtained from the experiments measure three main aspects of accuracy (Top1, Top5, and Top10), performance (MAP), and effectiveness (Mean Effectiveness, Median Effectiveness, and MRR). The Top1, Top5, and Top10 metrics summarize the accuracy results of the approaches when recommending the top 1 to 10 entries from the ranked list of recommended locations.

Table 6.2: Results of ***Ti***NoFeLo, SUM and VSM for accuracy, performance and effectiveness metrics

Project	**Metric**		**TiNoFeLo**	**SUM**	**VSM**	**Difference TiNoFeLo /SUM**	**Difference TiNoFeLo /VSM**
Eclipse JDT	**Accuracy**	**Top1 (%)**	10.5	3.5	5.5	7	5
		Top5 (%)	44	11	17.5	33	26.5
		Top10 (%)	54.5	16.5	24	38	30.5
	Performance	**MAP (%)**	9	1.8	5.8	7.2	3.2
	Effectiveness	**MRR (%)**	26.4	8.4	12.9	18	13.5
		Mean	71	290.2	193.2	-219.2	-122.2
		Median	7	75.5	40	-68.5	-33
AspectJ	**Accuracy**	**Top1 (%)**	14.5	7	15	7.5	-0.5
		Top5 (%)	57	28	36.5	29	20.5
		Top10 (%)	77	41	45.5	36	31.5
	Performance	**MAP (%)**	12.2	6.3	9.2	5.9	3
	Effectiveness	**MRR (%)**	34.3	17.8	25.4	16.5	8.9
		Mean	7.6	38.9	43.1	-31.3	-35.5
		Median	4.5	14.5	13.5	-10	-9
Netbeans	**Accuracy**	**Top1 (%)**	23.5	10	16	13.5	7.5
		Top5 (%)	55.5	34.5	37	21	18.5
		Top10 (%)	72	44	50	28	22
	Performance	**MAP (%)**	38.6	10.2	25.3	28.4	13.3
	Effectiveness	**MRR (%)**	43.4	24.8	29.1	18.6	14.3
		Mean	9.4	30.2	32.3	-20.8	-22.9
		Median	3	11	8	-8	-5
Rhino	**Accuracy**	**Top1 (%)**	29	18.5	14	10.5	15
		Top5 (%)	62	37.5	43.5	24.5	18.5
		Top10 (%)	74	54.5	59	19.5	15
	Performance	**MAP (%)**	51.5	16.3	32.6	35.2	18.9
	Effectiveness	**MRR (%)**	47	30.4	29.7	16.6	17.3
		Mean	6.8	18	16.8	-11.2	-10
		Median	3	9	6	-6	-3

In terms of accuracy assessment, ***Ti***NoFeLo outperforms SUM on the Top10 ranked locations by as much as 38%, 36%, 28%, and 20% on the Eclipse JDT, AspectJ, Netbeans, and Rhino projects, respectively. In comparing to VSM, the ***Ti***NoFeLo method obtained up to 31%, 32%, 22%, and 15% better accuracy for the Eclipse JDT, AspectJ, Netbeans,

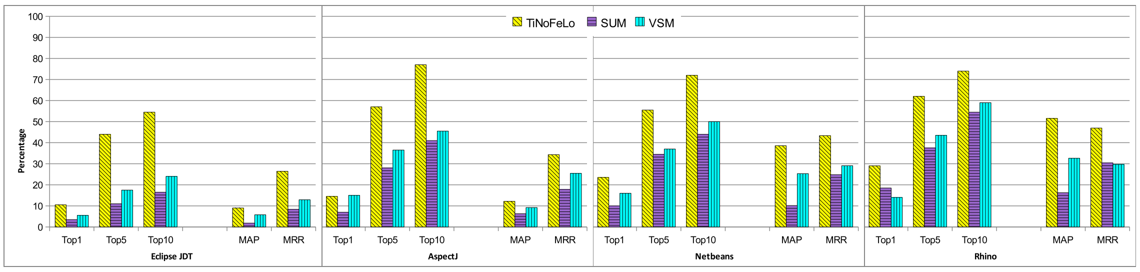


Figure 6.1: Results of ***Ti***NoFeLo, SUM and VSM for TopN, MAP and MRR metrics

and Rhino projects, respectively.

Figure 6.1 depicts graphs of the accuracy, MAP and MRR results for the ***Ti***NoFeLo and the baseline approaches. As shown in this figure, the accuracy of the ***Ti***NoFeLo method is remarkably better than the SUM and VSM approaches for all the subject systems.

The performance of ***Ti***NoFeLo is also evaluated against the SUM and VSM by investigating the results of MAP metric. The MAP results indicate the outperformance of ***Ti***NoFeLo over SUM and VSM by up to 35% and 19%, respectively. As mentioned above, the MAP results are also displayed in Figure 6.1. As shown in this figure, the performance results of ***Ti***NoFeLo are better than the baseline approaches on the subject systems especially on the Netbeans and Rhino projects.

In terms of effectiveness assessment, the results of MRR, Mean Effectiveness, and Median Effectiveness are measured. The effectiveness results indicate a significant improvement in all the cases. The results for the MRR metric indicate an effectiveness improvement over both the SUM and VSM. The improvement over SUM for Eclipse JDT, AspectJ, Netbeans and Rhino are respectively as much as 18%, 17%, 19%, and 17%, respectively; and the improvement over VSM for these projects are respectively up to 14%, 9%, 14%, and 17%. The graph of these results shown in Figure 6.1 confirms the considerable improvement made by ***Ti***NoFeLo in the effectiveness of feature location on all the subject systems.

On the other hand, the results of the mean and median of effectiveness also confirm the effectiveness improvement. As mentioned in Section 5.2.1, low values for these metrics indicates that less developer effort is required to find a correct source code location in the retrieved locations list by the method and consequently the more effective method for feature location. According to this rule, the effectiveness results show that less developer effort is needed when using ***Ti***NoFeLo compared to SUM and VSM. In the worst case, if SUM is used instead of ***Ti***NoFeLo, the developer would need to check on average 219 more locations, and if VSM is used, the developer may need to check 122 more locations (Eclipse JDT).

The effectiveness results are displayed in Figure 6.2. This figure shows box plot diagrams for the effectiveness metric for the subject systems. The top and bottom boxes in these graphs represent the upper and lower quartiles, respectively, and the line between the boxes represents the median. The whiskers above and below the boxes denote the maximum and minimum effectiveness values, respectively. Due to the non-normal distribution of the effectiveness values and very high number of outliers that contain extreme values, the box plot of the effectiveness values cannot be shown properly. Instead, the "outer fences"[1] were temporary removed from the results to make the differences between the methods/approaches visible in the graphs. Due to the high number of outliers, some of outlier numbers still remain in the effectiveness results that are shown by the cycle and star symbols in the box plot diagrams.

In the box plot graphs of effectiveness, low values which represent the positions of relevant files suggest potentially less effort is needed by a developer to locate relevant files, because the ranks are among the first results returned by the proposed method. To remind, the most effective approach for feature location is the approach with the lowest

[1]The formula for calculating the outer fences is Q3+3×(Q3-Q1).

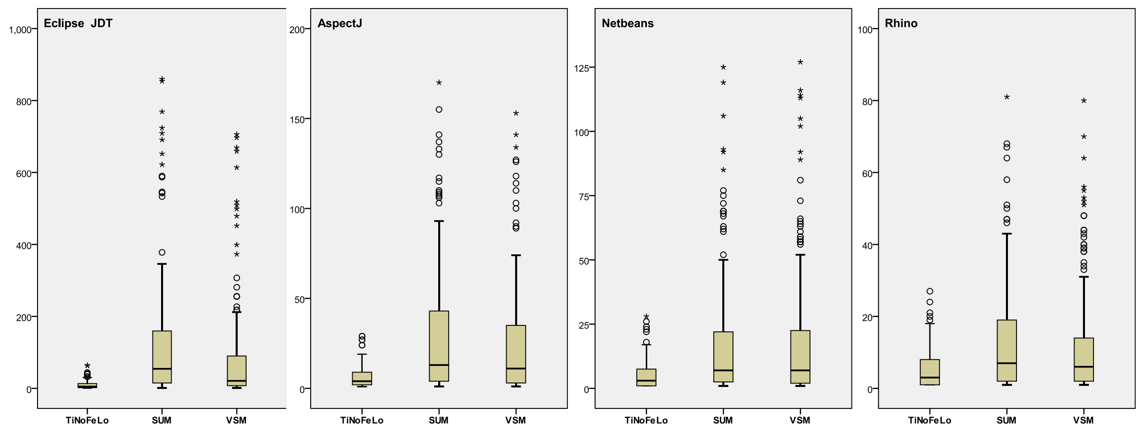


Figure 6.2: Results of *Ti*NoFeLo, SUM and VSM for effectiveness metric

effectiveness value. Note that the graphs for Eclipse JDT, AspectJ, Netbeans, and Rhino have different scales as a result of a different number of files in their respective repositories. This figure shows that for ***Ti***NoFeLo, the box plots are smaller and lower than SUM and VSM. This suggests that developers would potentially require less effort to locate relevant files for a change request when using ***Ti***NoFeLo. The figure also shows that, in general, the effectiveness of ***Ti***NoFeLo is significantly better than the SUM and VSM for all the subject systems.

6.1.1.2 Impact of Time Consideration

To assess the impact of time consideration in term weighting, the ***Ti***NoFeLo method is evaluated both with and without the use of time-metadata for weighting the noun terms. In this case, the use of the TATW technique is compared to the use of the TF-IDF technique. First, a version of ***Ti***NoFeLo using the TATW technique is applied to the subject systems, referred to as the ***Ti***NoFeLo. Then, TATW is replaced by TF-IDF and the same experiment runs which is referred to as the ***Ti***NoFeLo$_{TF-IDF}$. Table 6.3 shows the results of evaluation comparing the use of TATW, and TF-IDF in ***Ti***NoFeLo. This table reports the results of the experiments measured from the aspects of accuracy, performance and effectiveness. To enhance the comparison process for the readers, the last column in this table presents the differences between the results of ***Ti***NoFeLo and ***Ti***NoFeLo$_{TF-IDF}$ for identified metrics.

Table 6.3: Results of ***Ti***NoFeLo using TATW and TF-IDF techniques, for accuracy, performance and effectiveness metrics

Project	**Metric**		**TiNoFeLo**	**TiNoFeLo (TF-IDF)**	**Difference**
Eclipse JDT	**Accuracy**	**Top1 (%)**	10.5	9	1.5
		Top5 (%)	44	30	14
		Top10 (%)	54.5	48.5	6
	Performance	**MAP (%)**	9	0.7	8.3
	Effectiveness	**MRR (%)**	26.4	21.4	5
		Mean	71	88.1	-17.1
		Median	7	10	-3
AspectJ	**Accuracy**	**Top1 (%)**	14.5	5.5	9
		Top5 (%)	57	42.5	14.5
		Top10 (%)	77	63	14
	Performance	**MAP (%)**	12.2	6.2	6
	Effectiveness	**MRR (%)**	34.3	22.6	11.7
		Mean	7.6	19.8	-12.2
		Median	4.5	7	-2.5
Netbeans	**Accuracy**	**Top1 (%)**	23.5	16	7.5
		Top5 (%)	55.5	46	9.5
		Top10 (%)	72	68	4
	Performance	**MAP (%)**	38.6	29.6	9
	Effectiveness	**MRR (%)**	43.4	34.5	8.9
		Mean	9.4	13.6	-4.2
		Median	3	5	-2
Rhino	**Accuracy**	**Top1 (%)**	29	27	2
		Top5 (%)	62	59.5	2.5
		Top10 (%)	74	80	-6
	Performance	**MAP (%)**	51.5	56.2	-4.7
	Effectiveness	**MRR (%)**	47	45.4	1.6
		Mean	6.8	7.4	-0.6
		Median	3	3	0

The accuracy assessment on the Eclipse JDT, AspectJ and Netbeans projects shows that the use of TATW made an improvement in the accuracy, effectiveness and performance. The accuracy results on these subject systems show the improvement on all the TopN ranked locations. The improvements on the Top10 ranked locations retrieved from the Eclipse JDT, AspectJ and Netbeans projects are as much as 6%, 14%, and 4%, respectively. For the Rhino project, however, the results of ***Ti***NoFeLo using TATW indicate the improvement for Top1 and Top5; the result of the experiment for the Top10 shows the

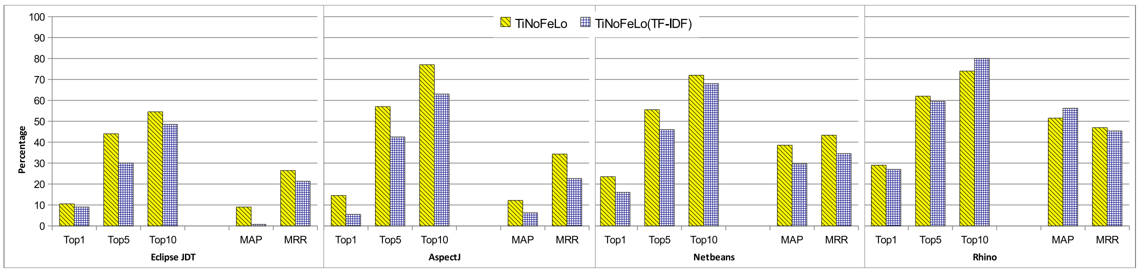


Figure 6.3: Results of ***Ti***NoFeLo and ***Ti***NoFeLo$_{TF-IDF}$ for TopN, MAP and MRR metrics

accuracy reduction as much as 6%. This is likely the result of the low evolution speed of this project, which is further discussed in Section 6.1.3. Figure 6.3 illustrates the graph of the results of accuracy for Top1, Top5, and Top10 as well as the results of performance and effectiveness measured by MAP and MRR. As shown in this figure, the use of the TATW technique resulted in the accuracy improvements for all of the subject systems except Rhino.

For the MAP metric, the use of TATW shows an improvement over the TF-IDF technique between 6% to 9% for JDT, AspectJ and Netbeans. For the Rhino project, the performance of TF-IDF was found to be better than that of TATW. In terms of effectiveness assessment, the MRR results show that the use of TATW technique improves the effectiveness over the use of TF-IDF on the subject systems without any exception. The improvement is between 2% to 12% that indicates, TATW is more effective than the TF-IDF technique. The MRR results of the Rhino project show a slight improvement. The graph of these results shown in Figure 6.3 confirms the improvement made by TATW in the effectiveness of feature location on all the subject systems.

The effectiveness of ***Ti***NoFeLo using TATW is further investigated compared to the use of TF-IDF by analyzing the results of Mean Effectiveness and Median Effectiveness. The comparison of the values of mean and median effectiveness reported in Table 6.3 shows that the results of these metrics follow the same line as MRR results. The effectiveness comparison shows that TATW is more effective than TF-IDF on all subject systems. This

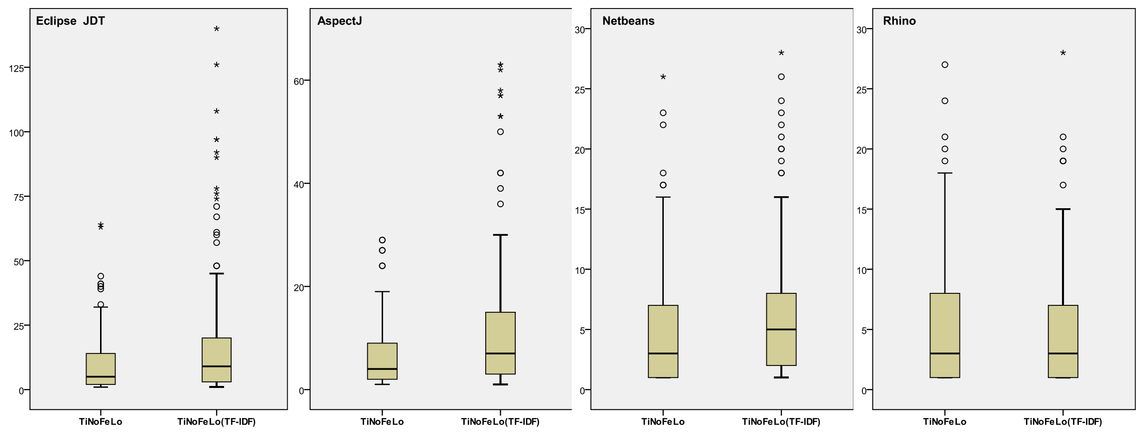


Figure 6.4: Results of ***Ti***NoFeLo and ***Ti***NoFeLo$_{TF-IDF}$ for effectiveness metric

indicates that the developer, in the worst case (Eclipse JDT), needs to check 17 locations more if TF-IDF is used than if TATW is used.

Figure 6.4 shows box plot graphs for the effectiveness results of ***Ti***NoFeLo using TATW and TF-IDF on the subject systems. To remind, low values in the box plots suggest potentially less effort is needed by a developer to locate features in the relevant source code files, because the ranks are among the first results returned by the feature location method. The most effective method/approach for feature location is the one with the lowest effectiveness value. As explained earlier, the graphs have different scales due to the different number of files in the subject systems. Due to the non-normal distribution of the effectiveness values and very high number of outliers that contain extreme values, the box plot of the effectiveness values cannot be shown properly. Thus, the "outer fences" were removed from the results and then the box plot diagrams were drawn to make the differences between the methods visible in the graphs.

As shown in Figure 6.4, the effectiveness results follow the same pattern as that for the accuracy. For the Rhino project, the differences of the results of effectiveness metrics reported in Table 6.3 is around zero to two. It means that the use of TATW in ***Ti***NoFeLo on Rhino project is likely as effective as the use of TF-IDF. This exception is likely an effect of the gradual changes to the Rhino project over time, as compared to more changes over time for the other projects. This difference would naturally reduce the effect that time-metadata has in weighting the terms.

Based on these results, it can be concluded that the use of time-metadata in term weighting makes improvement in the accuracy, performance and effectiveness of feature location applied on the projects with a medium to high speed of evolution, such as Eclipse JDT. It means that the use of the TATW technique produces better results for projects with a higher evolution speed. This conclusion is further discussed in the future.

6.1.1.3 Impact of Noun Usage

To evaluate the impact of using only noun terms, first, all of the terms were extracted from the text resources in the software repository and used in the ***Ti***NoFeLo method with the TATW technique (***Ti***NoFeLo$_{All-Terms}$). Then, only the noun terms were extracted from the text resources and the same experiment was run using those terms (***Ti***NoFeLo).

Table 6.4 presents the accuracy, performance, and effectiveness results of ***Ti***NoFeLo when using either all of the terms or only the noun terms. Last column in this table shows the differences between the results of proposed method using all terms and using only the nouns to enhance the comparison process. As shown in this table, the use of only the nouns notably improves the accuracy of feature location process. The improvements on the Top10 ranked locations for Eclipse JDT, AspectJ, Netbeans, and Rhino are by around 13%, 21%, 24%, and 26%, respectively. Figure 6.5 shows the accuracy graphs of the evaluation for all of the subject systems. These graphs demonstrate a significant improvement in the accuracy for the case of using only noun terms.

In terms of performance assessment, the results of MAP metric indicate that the use of noun-only makes an improvement on all the subject systems. The performance improvement is between 7% and 49%. On the other hand, the effectiveness of the proposed method is assessed by investigating the results of MRR, Mean Effectiveness and Median

Table 6.4: Results of ***Ti***NoFeLo using only the nouns and all types of terms, for accuracy, performance and effectiveness metrics

Project	Metric		TiNoFeLo	TiNoFeLo (All-Term)	Difference
EclipseJDT	**Accuracy**	**Top1 (%)**	10.5	5.3	5.2
		Top5 (%)	44	32.4	11.6
		Top10 (%)	54.5	41.5	13
	Performance	**MAP (%)**	9	0.2	8.8
	Effectiveness	**MRR (%)**	26.4	18	8.4
		Mean	71	296.9	-225.9
		Median	7	13	-6
AspectJ	**Accuracy**	**Top1 (%)**	14.5	17.6	-3.1
		Top5 (%)	57	41.5	15.5
		Top10 (%)	77	56.4	20.6
	Performance	**MAP (%)**	12.2	5.3	6.9
	Effectiveness	**MRR (%)**	34.3	28.5	5.8
		Mean	7.6	81.6	-74
		Median	4.5	8	-3.5
Netbeans	**Accuracy**	**Top1 (%)**	23.5	16.5	7
		Top5 (%)	55.5	37.8	17.7
		Top10 (%)	72	47.9	24.1
	Performance	**MAP (%)**	38.6	27.8	10.8
	Effectiveness	**MRR (%)**	43.4	28.3	15.1
		Mean	9.4	53	-43.6
		Median	3	10	-7
Rhino	**Accuracy**	**Top1 (%)**	29	12.8	16.2
		Top5 (%)	62	34	28
		Top10 (%)	74	47.9	26.1
	Performance	**MAP (%)**	51.5	2.4	49.1
	Effectiveness	**MRR (%)**	47	24.2	22.8
		Mean	6.8	29.1	-22.3
		Median	3	11	-8

Effectiveness. The investigation of the MRR results reveals an improvement as much as 23%, and indicates that the use of only noun terms is more effective than using all of the terms. Figure 6.5, in addition to demonstrating the accuracy results, displays the MAP and MRR graphs. As shown in this figure, the use of nouns makes a notable improvement especially on the Rhino project.

The assessment of the mean and median of the effectiveness results suggest that using only noun terms leads to less developer effort for finding the correct source code location for a change request. In the case of using all of the terms, a developer may need to check

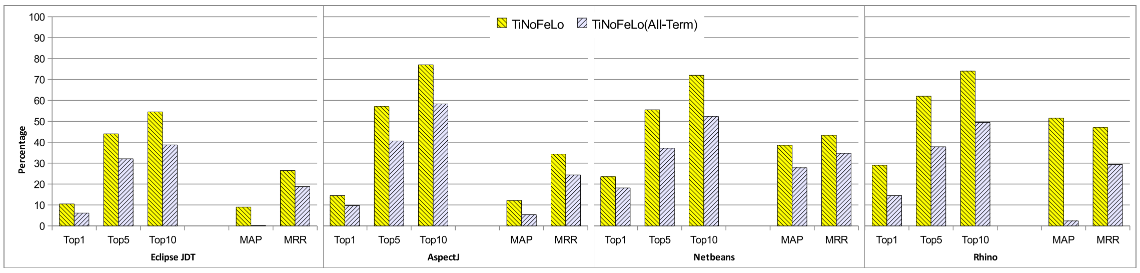


Figure 6.5: Results of ***Ti***NoFeLo and ***Ti***NoFeLo$_{All-Terms}$ for TopN, MAP and MRR metrics

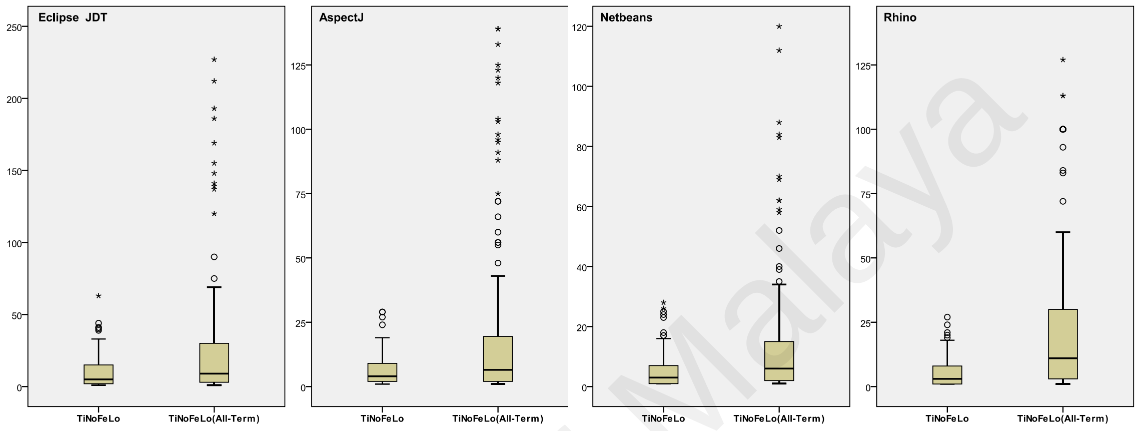


Figure 6.6: Results of ***Ti***NoFeLo and ***Ti***NoFeLo$_{All-Terms}$ for effectiveness metric

226 more locations in the worst case (Eclipse JDT). Figure 6.6 presents effectiveness box plot diagrams for using noun terms with ***Ti***NoFeLo. As mentioned earlier, due to the non-normal distribution of the effectiveness values and very high number of outliers that contain extreme values, the "outer fences" were removed from the results and then the box plot diagrams were drawn to make the differences between the methods visible in the graphs. As shown in the effectiveness graphs, the use of only the noun terms notably improves the effectiveness of the feature location process.

Another benefit to using only the noun terms is on the size of the dataset that needs to be analyzed for identifying the source code location for new change requests. Table 6.5 shows the number of terms extracted from the data-sources for each project's dataset, for both extracting all types of the terms and extracting only the noun terms. As would be expected, the Difference raw of this table shows that the size of dataset decreased significantly when extracting only the nouns. For JDT, AspectJ, Netbeans, and Rhino, the number of terms in the dataset was reduced by 60%, 26%, 39%, and 41%, respectively. With the reduction in the amount of data that needs to be analyzed, the execution time of the feature location process naturally decreases.

Table 6.5: Dataset sizes when using all types of terms and only the nouns

	JDT	**AspectJ**	**Netbeans**	**Rhino**
All Terms	920760	260170	57390	41052
Noun Terms	364523	192109	35293	24334
Differences	556237 (60.41%)	68061 (26.16%)	22097 (38.5%)	16718 (40.72%)

6.1.2 Statistical Results

In this section, the statistical analysis of the obtained results is reported and interpreted. First, the normality of the results is investigated to determine the statistical test to be used for analyzing the statistical significance of the observed improvements from the previous section. To assess the normality of the results, the values of two criteria, Skewness and Kurtosis, and one normality test, Shapiro-Wilk, are investigated. For the normal distributed data, the Skewness and Kurtosis values are between -2 and +2 and the Shapiro-Wilk p-value is higher than α, which is 0.05. For the Skewness and Kurtosis values which are more than +2 or less than -2, the data are recognized as the non-normal distributed data. Also for the Shapiro-Wilk p-value lower than α, ($\alpha < 0.05$), data are not normally distributed.

Table 6.6 presents the results of the normality evaluation of ***Ti***NoFeLo's results using either the TATW technique or TF-IDF technique, and ***Ti***NoFeLo using only the noun terms or all types of terms, as well as the normality evaluation of SUM, and VSM. For the accuracy metric, due to the values of Skewness, Kurtosis, and Shapiro-Wilk p-value, all the accuracy results are normal distributed, and therefore a parametric test can be used for statistical analysis. Therefore, the paired T-test was used for statistical comparison of

Table 6.6: Results of normality test for ***Ti***NoFeLo's experiments

Metric	Project	Method/ Approach	Skewness	Kurtosis	Shapiro -Wilk
Accuracy	**Eclipse JDT**	TiNoFeLo	-0.971	-0.101	0.133
		TiNoFeLo$_{TF-IDF}$	-0.458	-0.703	0.859
		TiNoFeLo$_{All-Terms}$	-0.601	-0.315	0.51
		SUM	-0.414	-0.666	0.725
		VSM	-0.755	-0.338	0.462
	AspectJ	TiNoFeLo	-0.971	0.327	0.371
		TiNoFeLo$_{TF-IDF}$	-0.75	-0.28	0.562
		TiNoFeLo$_{All-Terms}$	-0.762	0.114	0.361
		SUM	-0.551	-0.538	0.742
		VSM	-0.943	-0.061	0.276
	Netbeans	TiNoFeLo	-1.01	0.652	0.432
		TiNoFeLo$_{TF-IDF}$	-0.509	-0.806	0.486
		TiNoFeLo$_{All-Terms}$	-0.617	-0.701	0.529
		SUM	-1.009	0.349	0.221
		VSM	-0.583	-0.712	0.509
	Rhino	TiNoFeLo	-1.186	1.071	0.17
		TiNoFeLo$_{TF-IDF}$	-0.689	-0.089	0.604
		TiNoFeLo$_{All-Terms}$	-0.89	0.416	0.478
		SUM	-0.281	-0.861	0.895
		VSM	-0.792	-0.24	0.389
Effectiveness	**Eclipse JDT**	TiNoFeLo	5.666	34.941	0
		TiNoFeLo$_{TF-IDF}$	11.541	146.133	0
		TiNoFeLo$_{All-Terms}$	8.604	90.537	0
		SUM	2.616	7.157	0
		VSM	6.916	66.46	0
	AspectJ	TiNoFeLo	5.051	32.607	0
		TiNoFeLo$_{TF-IDF}$	6.188	53.347	0
		TiNoFeLo$_{All-Terms}$	5.621	42.977	0
		SUM	6.08	51.548	0
		VSM	4.368	21.919	0
	Netbeans	TiNoFeLo	3.922	16.126	0
		TiNoFeLo$_{TF-IDF}$	3.754	15.049	0
		TiNoFeLo$_{All-Terms}$	3.838	13.588	0
		SUM	3.997	22.607	0
		VSM	3.325	15.648	0
	Rhino	TiNoFeLo	5.618	37.673	0
		TiNoFeLo$_{TF-IDF}$	3.783	15.783	0
		TiNoFeLo$_{All-Terms}$	4.701	16.728	0
		SUM	2.576	7.593	0
		VSM	2.137	4.491	0

the two methods/approaches/techniques. In the case of comparing multiple results, the two-way ANOVA was conducted and a Least Significant Difference (LSD) test was used as the post hoc test.

On the other hand, according to the normality test for the effectiveness measurement, the obtained results are non-normally distributed. For example, for the VSM effectiveness results on the Rhino project, the Skewness and Kurtosis are not in the range of -2 and +2. Also, the Shapiro-Wilk p-value indicates the non-normality of the results. In general, the effectiveness results are non-normal and need non-parametric tests for statistical analysis. Therefore, the Wilcoxon test was used for comparing the results of two methods/approaches/techniques and the Friedman test was used for comparing multiple cases.

Table 6.7: Results of statistical test for ***Ti***NoFeLo comparing with baseline approaches, SUM and VSM

Metric	**Project**	**Two Sample**			**Effect Size**	
		Hypothesis	**Test Type**	**P-value**	**Type**	**Value**
Accuracy	**Eclipse JDT**	$H_{0,TiNoFeLovs.SUM}$	T-test	0	Hedgens' g	2.526
		$H_{0,TiNoFeLovs.VSM}$		0		1.934
	AspectJ	$H_{0,TiNoFeLovs.SUM}$		0		1.655
		$H_{0,TiNoFeLovs.VSM}$		0		0.811
	Netbeans	$H_{0,TiNoFeLovs.SUM}$		0		1.572
		$H_{0,TiNoFeLovs.VSM}$		0		1.27
	Rhino	$H_{0,TiNoFeLovs.SUM}$		0		1.554
		$H_{0,TiNoFeLovs.VSM}$		0		1.109
Effectiveness	**Eclipse JDT**	$H_{0,TiNoFeLovs.SUM}$	Wilcoxon	0	Cliff's delta	0.596
		$H_{0,TiNoFeLovs.VSM}$		0		0.43
	AspectJ	$H_{0,TiNoFeLovs.SUM}$		0		0.477
		$H_{0,TiNoFeLovs.VSM}$		0		0.353
	Netbeans	$H_{0,TiNoFeLovs.SUM}$		0		0.375
		$H_{0,TiNoFeLovs.VSM}$		0		0.313
	Rhino	$H_{0,TiNoFeLovs.SUM}$		0		0.335
		$H_{0,TiNoFeLovs.VSM}$		0		0.304

Table 6.7 shows the statistical analysis on the comparison of ***Ti***NoFeLo and the baseline approaches, SUM and VSM. The statistical test for the accuracy results shows the rejection of all formulated null hypothesis. This means an acceptance of the corresponding

alternative hypothesis. In the other words, there is a significant difference between the accuracy of ***Ti***NoFeLo comparing to SUM and VSM. Furthermore, to complement inferential statistics, the effect size values (Hedges' g and Cliff's delta) are reported in Table 6.7. The values of the effect size for the accuracy measurement suggest a medium to large practical significance on all of the subject systems. Similar results were found for the effectiveness results which the effectiveness of ***Ti***NoFeLo over SUM and VSM is remarkably significant for all subject systems. The effect size values for the effectiveness metric indicated a medium to large practical significance difference between ***Ti***NoFeLo and SUM. Comparing to VSM, the effect size values suggest a small to medium practical significance. In general, the statistical analysis of the obtained results shows that both the improvements in effectiveness and accuracy of ***Ti***NoFeLo over SUM and VSM are significant and the effect sizes mostly suggest a large practical significance.

Table 6.8: Results of statistical test for ***Ti***NoFeLo comparing with ***Ti***NoFeLo$_{TF-IDF}$ and ***Ti***NoFeLo$_{All-Terms}$

Metric	**Project**	Multiple Sample		Two Sample			Effect Size	
		Test Type	P-value	Hypothesis	Test Type	P-value	Type	Value
Accuracy	**Eclipse JDT**	Two-way ANOVA	0	$H_{0,TiNoFeLovs.TiNoFeLoTF-IDF}$ $H_{0,TiNoFeLovs.TiNoFeLoAll-Terms}$	LSD	0 0	Hedgens' g	0.611 0.721
	AspectJ		0	$H_{0,TiNoFeLovs.TiNoFeLoTF-IDF}$ $H_{0,TiNoFeLovs.TiNoFeLoAll-Terms}$		0 0		0.847 1.236
	Netbeans		0	$H_{0,TiNoFeLovs.TiNoFeLoTF-IDF}$ $H_{0,TiNoFeLovs.TiNoFeLoAll-Terms}$		0 0		0.379 1.33
	Rhino		0	$H_{0,TiNoFeLovs.TiNoFeLoTF-IDF}$ $H_{0,TiNoFeLovs.TiNoFeLoAll-Terms}$		0.723* 0		0.027 1.815
Effectiveness	**Eclipse JDT**	Friedman	0	$H_{0,TiNoFeLovs.TiNoFeLoTF-IDF}$ $H_{0,TiNoFeLovs.TiNoFeLoAll-Terms}$	Wilcoxon	0.003 0	Cliff's delta	0.145 0.247
	AspectJ		0	$H_{0,TiNoFeLovs.TiNoFeLoTF-IDF}$ $H_{0,TiNoFeLovs.TiNoFeLoAll-Terms}$		0 0		0.261 0.26
	Netbeans		0	$H_{0,TiNoFeLovs.TiNoFeLoTF-IDF}$ $H_{0,TiNoFeLovs.TiNoFeLoAll-Terms}$		0.002 0		0.136 0.335
	Rhino		0	$H_{0,TiNoFeLovs.TiNoFeLoTF-IDF}$ $H_{0,TiNoFeLovs.TiNoFeLoAll-Terms}$		0.393* 0		0.013 0.436

TiNoFeLo with TATW was also evaluated in terms of term weighting against TF-IDF, and when using only noun terms instead of all terms. First, these cases are compared in general with a multiple group analysis. Table 6.8 reports the results of the statistical analysis of ***Ti***NoFeLo comparing to ***Ti***NoFeLo$_{TF-IDF}$ and ***Ti***NoFeLo$_{All-Term}$. Note that all of the statistical results for the multiple group analysis are less than α (p-value$<$ 0.05). This shows the overall superiority of ***Ti***NoFeLo with respect to the accuracy and effectiveness for feature location. In the other words, ***Ti***NoFeLo significantly improves the process of feature location by using time-aware term-weighting and using only the noun terms.

Also, a post hoc analysis was conducted to further judge the significance of the difference for the most important term-weighting and term usage combinations. The selected pairs of results for comparison are: ***Ti***NoFeLo with ***Ti***NoFeLo$_{TF-IDF}$, and ***Ti***NoFeLo using only noun terms with ***Ti***NoFeLo using all the terms. As shown in Table 6.8, the accuracy and effectiveness results of ***Ti***NoFeLo and ***Ti***NoFeLo$_{TF-IDF}$ are significant for the Eclipse JDT, AspectJ and Netbeans projects. Furthermore, to complement inferential statistics, the effect size values (Hedges' g and Cliff's delta) are reported in Table 6.8. The effect size values suggest medium, large and small significance levels for the accuracy results on the Eclipse JDT, AspectJ and Netbeans projects, respectively. For the effectiveness, the effect size values indicate the negligible to small significance levels for those subject systems.

The statistical analysis for the Rhino project (marked with an asterix) indicates no significant difference between the results of ***Ti***NoFeLo and ***Ti***NoFeLo$_{TF-IDF}$. The reasons of this exception are discussed in Section 6.1.3. Furthermore, the effect size values for both the accuracy and effectiveness of ***Ti***NoFeLo against ***Ti***NoFeLo$_{TF-IDF}$ suggest a negligible difference for Rhino. It means that time consideration on Rhino project does not have a significant impact since the difference is not considerable.

When using noun terms only, the results of ***Ti***NoFeLo is significantly better than using all of the terms with respect to both the accuracy and the effectiveness measurements.

This means that the improvements are not the result of chance and the differences are statistically significant. The effect size results for the accuracy metric suggest a large practical significance for most of the subject systems. For the effectiveness metric, the effect size values suggest a small to medium practical significance. The analysis of the obtained descriptive and statistical results is discussed in the next section.

6.1.3 Discussion

In the first step of evaluating the proposed method, ***Ti***NoFeLo is assessed over the state-of-the-art IR-based approaches, i.e. SUM and VSM, which are treated as baseline feature location approaches. The goal of this assessment is evaluating the proposed time-based method against the location-based feature location approaches. The analysis of the descriptive results for the ***Ti***NoFeLo method compared to the feature location baseline approaches reveals the superiority of ***Ti***NoFeLo in all aspects of accuracy, performance, and effectiveness. Moreover, the statistical analysis of the results emphasizes the significance of the improvement made by the ***Ti***NoFeLo method in feature location process over the baseline approaches. It means, the rejection of the null hypotheses of $\mathbf{H}_{0,TiNoFeLovs.SUM}$ and $\mathbf{H}_{0,TiNoFeLovs.VSM}$ and consequently, acceptance of the corresponding alternative hypotheses. Furthermore, the values of the effect size mostly suggest a large practical significance for ***Ti***NoFeLo when compared with SUM and VSM. These results lead to answering $\mathbf{Ti.RQ}_1$ and concluding that ***Ti***NoFeLo outperforms the feature location baseline approaches. In other words, considering the time of use of the noun terms improves the feature location process over the baseline approaches.

Further comparison of ***Ti***NoFeLo with the baseline approaches in terms of time complexity shows that ***Ti***NoFeLo and VSM are both in order of $|F| \times |T|$ (F is the number of source code files in the corpus and T is the average of the number of terms in the files). The comparison of time complexity of ***Ti***NoFeLo and SUM shows that SUM with

Dirichlet smoothing method is in the order of $|F| \times |q|$ (F is the number of source code files in the corpus and q is the average of the number of terms in the given query) (Zhai & Lafferty, 2004). This means that SUM can be considered as efficient as ***Ti***NoFeLo in terms of time complexity.

Moreover, the impact of time consideration is especially investigated by evaluating ***Ti***NoFeLo when using TATW technique versus the TF-IDF technique. The descriptive results for considering time in the weighting the terms for the JDT, AspectJ, and Netbeans projects show an improvement for all of the metrics. The statistical testing of these results confirms the significance of the differences. It means the rejection of the null hypothesis of $\mathbf{H}_{0,TATWvs.TF-IDF}$and acceptance of the corresponding alternative hypothesis. Moreover, the effect size magnitudes for the accuracy metric suggest a small to large significance level improvement for these projects. For the effectiveness metric, the effect size suggests a negligible to small level of improvement. The effect size value has a direct relationship with the outliers in the data and the outliers lead to a misleading effect size calculation. In these cases, the effectiveness results contain a large set of outliers, and these cannot be removed due to their usefulness and importance for evaluating the effectiveness metric. Therefore, although TATW significantly improves the feature location process, the effect size cannot reflect the real level of improvement for the effectiveness results.

As discussed previously, the consideration of time in term weighting has a different effect on feature location for the Rhino project. The results of the evaluation on this project show that ***Ti***NoFeLo is not significantly better than ***Ti***NoFeLo$_{TF-IDF}$ for projects with a low evolution speed. This result shows the effect of the volume of modified data over time on the time-based feature location process. As indicated in Table 5.1 presented in Chapter 5, the average number of changes or modifications to source code per day for JDT is 48.8, for AspectJ is 11.26 and for Rhino is 1.9. This modification speed is one measure of the evolution speed of the project. Another evolution speed measure is the number of fixed change requests. The average number of fixed change requests per day for JDT, AspectJ, and Rhino were found to be 5.5, 0.48, and 0.13, respectively. In other words, the average number of source code changes for JDT is about 25 times more than for Rhino, and the average number of fixed change requests for JDT is 55 times more than for Rhino. This indicates that the evolution speed of Rhino is significantly lower than that of the other projects. Obviously, the volume of modified data and the number of fixed change requests over time, will affect a technique based on time-metadata.

In other words, when the volume of the data over time is small, it is possible that TATW would not provide considerable improvement over other term-weighting techniques. In the project with low evolution speed, the first parameter of the proposed term-weighting technique, which is consideration of frequency of the term in a set of terms that were created or modified at the same time, may lead to a negligible value. This parameter has a direct effect on the results of the TATW and cause to misleading of the weighting function. In this case, the common term-weighting techniques that only consider the term frequency in the file or project, i.e. TF-IDF, may obtain better results. Therefore, the use of TATW is not recommended for projects with a low evolution speed, such as Rhino.

In general, it can be concluded that the TATW technique outperforms the TF-IDF technique, specifically for those projects with medium and high evolution speeds. The results lead to answering **Ti.RQ**$_2$, the impact of considering time in term-weighting. The consideration of time-metadata in term-weighting can significantly improve the accuracy, performance and effectiveness of a feature location method for projects with a medium to high evolution speed, such as Eclipse JDT. This result is encouraging, as these are the types of projects that most need assistance with feature location. The novelty of ***Ti***NoFeLo is the consideration of the evolution of the text resources over time. This will naturally have a greater effect for a project with a medium to high evolution speed, as the weights of the terms are affected by the volume of modifications to the text resources over time.

In terms of using only the noun terms, the descriptive results show a significant improvement on all aspects of accuracy, performance, and effectiveness. The results of the statistical test on the obtained results indicate the significant improvement when using only the nouns. Furthermore, the effect size values suggest a medium to large practical significance for the accuracy metric and a small to medium improvement for the effectiveness metric. As mentioned above, the low value of the effect size for the effectiveness is the result of the high number of outliers that cause a misleading effect size. In general, the results indicate a positive impact of noun-only use for improving the accuracy, performance and effectiveness of a feature location process on projects of different scales.

Furthermore, the use of only noun terms significantly reduces the dataset sizes thereby improving the execution time for the feature location process. The data collected from the software repositories is very noisy. Using only the nouns, instead of all types of terms, not only provides enough information with which to identifying feature locations, but also reduces the amount of noisy data (Sarawagi, 2008). As with any feature location method, the aim is to identify the source code files for resolving a change request. As the name of a source code file is typically a combination of nouns, these nouns would be the most useful and meaningful terms in determining the relevant source code locations for a change request.

On the other hand, in many languages, nouns are known as the most important category of terms that represents the meaning of a text. Moreover, previous research on text analysis indicates that nouns carry most of the meaning of a sentence (Bouras

& Tsogkas, 2010). Accordingly, selection of the nouns, which exist in a text, results in better semantic representation of the text. Another effect of using only noun terms is the elimination of the need for dimensional reduction and threshold determination methods, which is one of the challenges in the use of IR models (Crain et al., 2012). Also, using only the noun terms can summarize the source code data and enhance MSR tasks (Haiduc, Aponte, & Marcus, 2010). In summary, the use of only noun terms has a high positive impact in the feature location process and strongly supports the fourth alternative hypothesis ($\mathbf{H}_{a,All-Terms vs.Noun-Terms}$). Collectively, these reasons answer **Ti.RQ**$_3$, the impact of using only noun terms in feature location process, that noun usage not only reduces the size of dataset but also significantly improves the accuracy, performance and effectiveness of a feature location process.

6.2 Evaluation of the ***De*****NoFeLo Method**

As explained in Chapter 4, ***De***NoFeLo is a feature location method that analyzes the text data from the aspect of developers, who created and modified the data recorded in software repository, to locate a change request in the source code. The ***De***NoFeLo method is supported by a term-weighting technique, TADTW. To experimentally evaluate the robustness of ***De***NoFeLo for feature location, a set of experiments were conducted. In Chapter 5, the required settings for experimentally evaluation of ***De***NoFeLo were explained in detail. The descriptive and statistical results obtained from the conducted experiments are reported in Sections 6.2.1 and 6.2.2, respectively. The descriptive and statistical results are analyzed in Section 6.2.3 with respect to the identified research questions for the ***De***NoFeLo method.

6.2.1 Descriptive Results

In this section, the results of the experiments are evaluated from the aspect of the identified metrics in Section 5.2.1. The results of the experiments are presented based on the specific research questions defined for ***De***NoFeLo in Section 5.3.2. To remind, the ***De***NoFeLo's research questions are represented in Table 6.9.

Table 6.9: Research questions of ***De***NoFeLo method

De.RQ1	Does ***De***NoFeLo outperform SUM and VSM, as the baseline feature location approaches?
De.RQ2	Does ***De***NoFeLo, which is a developer-based method, outperform ***Ti***NoFeLo as a time-based feature location method?
De.RQ3	What is the impact of consideration of time in ***De***NoFeLo as a developer-based method?
De.RQ4	What is the impact of consideration of keywords in ***De***NoFeLo as a developer-based method?

With respects to the research questions, first, Section 6.2.1.1 reports the results of ***De***NoFeLo by comparing with the results of the feature location baseline approaches, i.e. SUM and VSM. Next, in Section 6.2.1.2, the proposed developer-based method is evaluated against the time-based method that proposed in this thesis. In this case, the results of ***De***NoFeLo are assessed over ***Ti***NoFeLo. Then, as regards with the third and fourth research questions, the impacts of time and keywords consideration in a developer-based feature location method are examined in Section 6.2.1.3.

6.2.1.1 *De*NoFeLo versus Baseline Approaches

In the first step of method evaluation, the results of ***De***NoFeLo are compared to the results obtained from the state-of-the-art IR-based approaches, i.e. SUM and VSM, treated as the baseline feature location approaches. The goal of this comparison is assessing the proposed developer-based method against the location-based feature location approaches. Table 6.10 reports the results of ***De***NoFeLo, SUM, and VSM for the accuracy (Top1, Top5, and Top10), performance (MAP) and effectiveness (MRR, Mean, and Median) metrics. The last two columns of this table show the differences of the results of ***De***NoFeLo with the SUM and VSM, respectively, to enhance the comparison process.

Table 6.10: Results of ***De***NoFeLo, SUM and VSM for accuracy, performance and effectiveness metrics

Project	Metric		DeNoFeLo	SUM	VSM	Difference DeNoFeLo /SUM	Difference DeNoFeLo /VSM
Eclipse JDT	**Accuracy**	**Top1 (%)**	21.5	3.5	5.5	18	16
		Top5 (%)	52	11	17.5	41	34.5
		Top10 (%)	71	16.5	24	54.5	47
	Performance	**MAP (%)**	9.9	1.8	5.8	8	4.1
	Effectiveness	**MRR (%)**	37.7	8.4	12.9	29.4	24.9
		Mean	66	290.2	193.2	-224.2	-127.2
		Median	4.5	75.5	40	-71	-35.5
AspectJ	**Accuracy**	**Top1 (%)**	13.5	7	15	6.5	-1.5
		Top5 (%)	47	28	36.5	19	10.5
		Top10 (%)	64	41	45.5	23	18.5
	Performance	**MAP (%)**	10.4	6.3	9.2	4.1	1.2
	Effectiveness	**MRR (%)**	29.9	17.8	25.4	12.1	4.5
		Mean	22.5	38.9	43.1	-16.4	-20.6
		Median	6	14.5	13.5	-8.5	-7.5
Netbeans	**Accuracy**	**Top1 (%)**	27.5	10	16	17.5	11.5
		Top5 (%)	57.5	34.5	37	23	20.5
		Top10 (%)	78.5	44	50	34.5	28.5
	Performance	**MAP (%)**	41.8	10.2	25.3	31.6	16.5
	Effectiveness	**MRR (%)**	46.6	24.8	29.1	21.8	17.5
		Mean	8.5	30.2	32.3	-21.7	-23.8
		Median	3	11	8	-8	-5
Rhino	**Accuracy**	**Top1 (%)**	30.5	18.5	14	12	16.5
		Top5 (%)	65	37.5	43.5	27.5	21.5
		Top10 (%)	86	54.5	59	31.5	27
	Performance	**MAP (%)**	54.8	16.3	32.6	38.5	22.1
	Effectiveness	**MRR (%)**	49	30.4	29.7	18.6	19.3
		Mean	6	18	16.8	-11.9	-10.8
		Median	3	9	6	-6	-3

As shown in Table 6.10, ***De***NoFeLo is notably more accurate than SUM and VSM for almost all the TopN ranked locations on the JDT, AspectJ, Netbeans, and Rhino projects. The ***De***NoFeLo outperforms the accuracy of SUM as much as 55%, 23%, 35%, and 32%, respectively on the JDT, AspectJ, Netbeans, and Rhino projects. In comparing to the VSM, the accuracy improvement is up to 47%, 19%, 29%, and 27%, respectively, for those subject systems. The accuracy results are also illustrated in Figure 6.7. The superiority of ***De***NoFeLo over SUM and VSM is demonstrated in this figure especially for Eclipse JDT.

In terms of performance evaluation, the MAP results indicate a higher performance for ***De***NoFeLo over SUM and VSM for all the subject systems. For example, on the

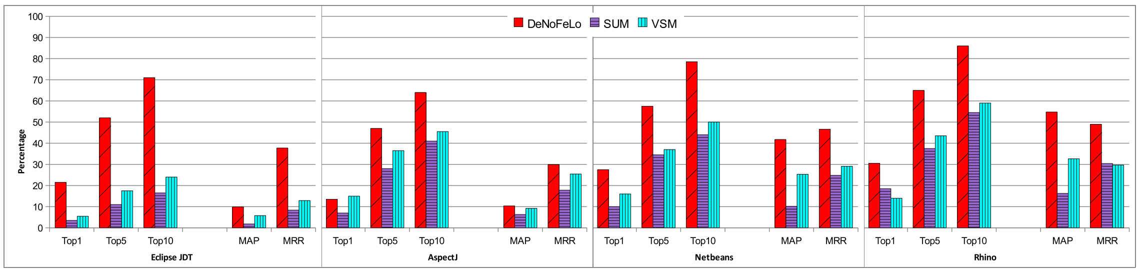


Figure 6.7: Results of ***De***NoFeLo, SUM and VSM for TopN, MAP and MRR metrics

Eclipse JDT project, ***De***NoFeLo's MAP is around five times that of the MAP of SUM and almost two times the MAP of VSM. Also, on the Netbeans project, the ***De***NoFeLo's MAP is almost four times the MAP of SUM and two times the MAP of VSM. The performance improvement over SUM is between 8% and 39% and over VSM is around 1% to 22%. The MAP results are also illustrated in Figure 6.7.

The analysis of the results of effectiveness metrics shows that MRR, Mean and Median Effectiveness follow the same line as accuracy and performance metrics. For the MRR metric, ***De***NoFeLo remarkably outperforms SUM and VSM. The improvement over SUM is by as much as 29%, 12%, 22%, and 19%, on the JDT, AspectJ, Netbeans, and Rhino projects, respectively. Comparing to VSM, the improvement is by up to 25%, 5%, 18%, and 19%, on these projects. Figure 6.7 also displays the bar chart diagram of the MRR results for ***De***NoFeLo, SUM, and VSM.

Further analysis of the effectiveness by investigating the mean effectiveness values shows that identifying the location using SUM or VSM requires checking around 12 to 224 and 11 to 127 more locations, respectively, for the subject systems, when compared to ***De***NoFeLo. Due to the non-normal distribution of the effectiveness results, in addition to examining the average number of locations that will need to be checked, the median of the effectiveness results also needs to be investigated. The median of the number of locations to check for SUM is 6 to 71 and, for VSM, it is 3 to 36 more locations over the median of ***De***NoFeLo. Both the mean and median of the effectiveness indicate that ***De***NoFeLo is

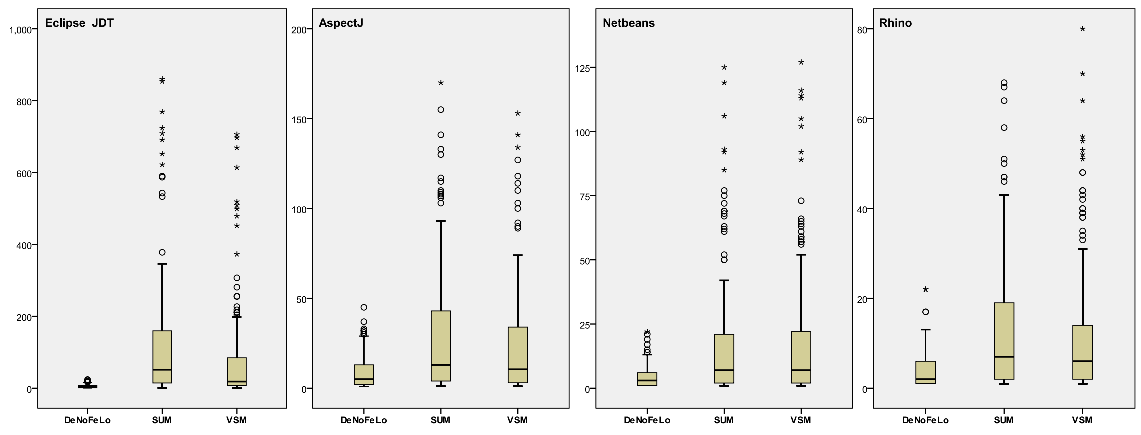


Figure 6.8: Results of ***De***NoFeLo, SUM and VSM for effectiveness metric

more effective than SUM and VSM.

The box plot graph of the effectiveness results are shown in Figure 6.8. Low values in the box plots, which represent the positions of relevant files, suggest potentially less effort is needed by a developer to locate relevant files, because the ranks are among the first results returned by the feature location approach. The most effective approach for feature location is the approach with the lowest effectiveness value. The graphs have different scales due to the different number of files in the subject systems. The top and bottom boxes in these graphs represent the upper and lower quartiles, respectively, and the line between the boxes represents the median. The whiskers above and below the boxes denote the maximum and minimum effectiveness values, respectively. Due to the non-normal distribution of the effectiveness values and very high number of outliers that contain extreme values, the "outer fences"[2] were temporary removed from the results to make the differences between the approaches visible in the graphs. Due to the high number of outliers, the numbers of outliers still remain in the effectiveness results that are shown by the cycle and star symbols in the box plot diagrams. As it is shown in Figure 6.8, the box plot of the effectiveness results of ***De***NoFeLo is smaller than that of SUM and VSM which indicates the superiority of the ***De***NoFeLo effectiveness over the

[2]The formula for calculating the outer fences is Q3+3×(Q3-Q1).

baseline approaches.

6.2.1.2 ***De*****NoFeLo versus TiNoFeLo**

As mentioned earlier, ***De***NoFeLo analyzes the data from the aspect of developer who created or modified the data in the repository. To further evaluate the developer-aspect analysis of data for feature location, the results of ***De***NoFeLo are assessed by comparing with the results of ***Ti***NoFeLo. As discussed in Section 6.1, the use of ***Ti***NoFeLo method for feature location makes a significant improvement in the accuracy, effectiveness and performance of location identification process. The aim of this assessment is evaluating the proposed developer-based method over the time-based method. Table 6.11 presents the results of ***De***NoFeLo and ***Ti***NoFeLo using the identified metrics.

The accuracy results reported in Table 6.11 shows that ***De***NoFeLo improves the accuracy of ***Ti***NoFeLo on all the TopN ranked locations of the JDT, Netbeans and Rhino projects. The accuracy improvements for Top10 ranked locations on these subject systems are by as much as 17%, 7%, and 12%, respectively. For the AspectJ project, the ***Ti***NoFeLo method obtained higher accuracy over the ***De***NoFeLo method. The reason for no improvement for the AspectJ project is due to the low number of developers working on this project: 8 developers (See Table 5.1). In projects that have a low number of developers, analyzing the source code data from the developer-aspect is more challenging and leads to poorer accuracy in the developer-based feature location approaches. This issue is further discussed in Section 6.2.3.

The investigation of MAP results, which highlights the performance of the feature location approaches, indicates that ***De***NoFeLo performs better than ***Ti***NoFeLo on the JDT, Netbeans and Rhino projects. Similar to the accuracy results, the ***De***NoFeLo performance on the AspectJ project is no better than ***Ti***NoFeLo performance. The accuracy and performance results are displayed in Figure 6.9. As shown in this figure, the accuracy

Table 6.11: Results of ***De***NoFeLo and ***Ti***NoFeLo for accuracy, performance and effectiveness metrics

Project	**Metric**		**DeNoFeLo**	**TiNoFeLo**	**Difference**
Eclipse JDT	**Accuracy**	**Top1 (%)**	21.5	10.5	11
		Top5 (%)	52	44	8
		Top10 (%)	71	54.5	16.5
	Performance	**MAP (%)**	9.9	9	0.9
	Effectiveness	**MRR (%)**	37.7	26.4	11.3
		Mean	66.1	71	-4.9
		Median	4.5	7	-2.5
AspectJ	**Accuracy**	**Top1 (%)**	13.5	14.5	-1
		Top5 (%)	47	57	-10
		Top10 (%)	64	77	-13
	Performance	**MAP (%)**	10.4	12.2	-1.8
	Effectiveness	**MRR (%)**	29.9	34.3	-4.4
		Mean	22.5	7.6	14.9
		Median	6	4.5	1.5
Netbeans	**Accuracy**	**Top1 (%)**	27.5	23.5	4
		Top5 (%)	57.5	55.5	2
		Top10 (%)	78.5	72	6.5
	Performance	**MAP (%)**	41.8	38.6	3.2
	Effectiveness	**MRR (%)**	46.6	43.4	3.2
		Mean	8.5	9.4	-0.9
		Median	3	3	0
Rhino	**Accuracy**	**Top1 (%)**	30.5	29	1.5
		Top5 (%)	65	62	3
		Top10 (%)	86	74	12
	Performance	**MAP (%)**	54.8	51.5	3.3
	Effectiveness	**MRR (%)**	49	47	2
		Mean	6	6.8	-0.8
		Median	3	3	0

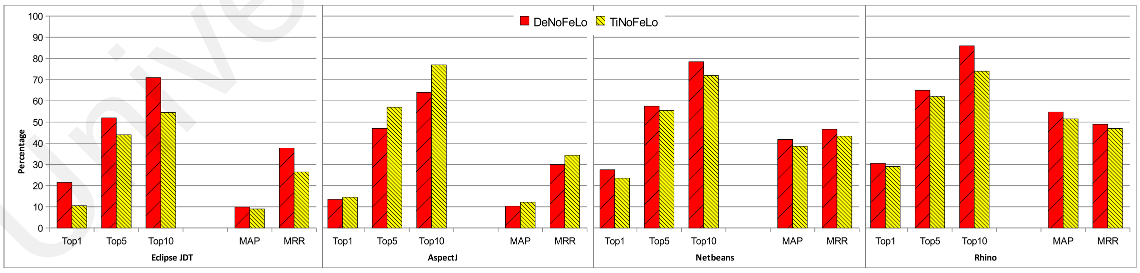


Figure 6.9: Results of ***De***NoFeLo and ***Ti***NoFeLo for TopN, MAP and MRR metrics

and performance evaluation on JDT, Netbeans, and Rhino show the outperformance of ***De***NoFeLo.

The ***De***NoFeLo effectiveness, assessed by analyzing the results of MRR, Mean, and Median metrics, follows the same line as the accuracy and performance of ***De***NoFeLo.

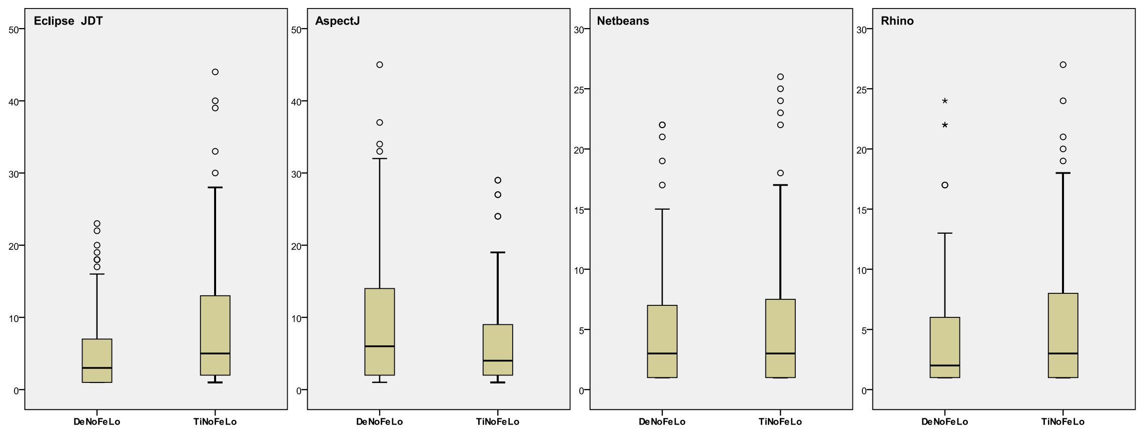


Figure 6.10: Results of ***De***NoFeLo and ***Ti***NoFeLo for effectiveness metric

The MRR results show up to 11% improvement over ***Ti***NoFeLo on the JDT, Netbeans, and Rhino projects. Figure 6.9 illustrates the bar chart diagrams of the MRR results as well as the accuracy and MAP results. As shown in this figure, the MRR results of ***De***NoFeLo are slightly higher than ***Ti***NoFeLo on the subject systems, except AspectJ.

Further investigation of the mean and median effectiveness metrics indicates a slight effectiveness improvement on JDT, Netbeans, and Rhino. Figure 6.10 presents the box plot graphs of the effectiveness results. As shown in this figure, the effectiveness of ***De***NoFeLo is slightly better than ***Ti***NoFeLo for all the subject systems, except AspectJ, which again is due to the low number of developers working on this project.

6.2.1.3 Impact of Time and Keywords Consideration

One of important considerations in this study is the effect of using time-metadata when analyzing the data from the developer-aspect for feature location. As mentioned in Section 4.2, since ***Ti***NoFeLo analyzes the data from the time-aspect, all times at which a term was used in the repository were taken into account when weighting the terms. In that case, the term could have been used at several different times in different commits of a file, and consequently, the term would have several different weights based on the different times it was used. Unlike ***Ti***NoFeLo, since ***De***NoFeLo analyzes the data from the developer-aspect, only the most recent time of use for a term is considered in ***De***NoFeLo. This avoids the combination of time and developer aspects analysis of data as well as simplifying the weighting process for identifying the related location.

To evaluate the effect of using only the most recent time of a term usage, as an important parameter in weighting the term by ***De***NoFeLo, the results of the method that uses time-metadata (i.e. ***De***NoFeLo) and the results of the same method but does not use time-metadata (i.e. ***De***NoFeLo$_{No-Time}$) were investigated. Furthermore, as mentioned in Section 4.2, an inverse of the developer frequency of use of a term was added to weights of the terms, as the keyword parameter, to have more weight for the keywords terms that were used by a developer further in the past. Accordingly, to assess the effect of keywords in ***De***NoFeLo, the results of the proposed method without the weighting of keywords, referred to as ***De***NoFeLo$_{No-Keywords}$, was also investigated.

Table 6.12 presents the results of the proposed method with and without these parameters, time and keyword. The results of the experiments show that when time-metadata is not taken into account (***De***NoFeLo$_{No-Time}$), the performance of ***De***NoFeLo for all of the metrics degraded for almost all the subject systems; the Rhino project being the exception. The results of ***De***NoFeLo for Rhino are approximately the same as the ***De***NoFeLo$_{No-Time}$ results (around 0% to 2% difference). The reasons of this issue are discussed in Section 6.2.3.

The results of the experiment indicate that by not considering time, the accuracy of ***De***NoFeLo is reduced between 6% and 9% for JDT, AspectJ, and Netbeans when recommending the Top10 locations. The performance and effectiveness of ***De***NoFeLo highlighted by the MAP and MRR metrics shows the same achievements. The MAP and MRR results indicate that the performance and effectiveness of ***De***NoFeLo were reduced by up to 8% and 6%, respectively, when the time-metadata was not considered

Table 6.12: Results of ***De***NoFeLo, ***De***NoFeLo$_{No-Time}$, and ***De***NoFeLo$_{No-Keywords}$ for accuracy, performance and effectiveness metrics

Project	Metric		DeNoFeLo	DeNoFeLo (No-Time)	DeNoFeLo (No-Keywords)	Difference on No-Time	Difference on No-Keywords
Eclipse JDT	Accuracy	Top1 (%)	21.5	21	10.5	0.5	11
		Top5 (%)	52	44	34.5	8	17.5
		Top10 (%)	71	63.5	56.5	7.5	14.5
	Performance	MAP (%)	9.9	12.2	6.6	-2.3	3.3
	Effectiveness	MRR (%)	37.7	35.2	24.4	2.5	13.3
		Mean	66	80.9	81.2	-14.9	-15.2
		Median	4.5	6	7	-1.5	-2.5
AspectJ	Accuracy	Top1 (%)	13.5	11	6	2.5	7.5
		Top5 (%)	47	41	36.5	6	10.5
		Top10 (%)	64	58.5	57	5.5	7
	Performance	MAP (%)	10.4	2	1.7	8.4	8.7
	Effectiveness	MRR (%)	29.9	25.4	21.9	4.5	8
		Mean	22.5	21.5	24.3	1	-1.8
		Median	6	8	9	-2	-3
Netbeans	Accuracy	Top1 (%)	27.5	23	16.5	4.5	11
		Top5 (%)	57.5	50.5	47.5	7	10
		Top10 (%)	78.5	69.5	66.5	9	12
	Performance	MAP (%)	41.8	35.5	30.8	6.3	11
	Effectiveness	MRR (%)	46.6	40.1	34.7	6.5	11.9
		Mean	8.5	12	14.4	-3.5	-5.9
		Median	3	4	5	-1	-2
Rhino	Accuracy	Top1 (%)	30.5	31	25	-0.5	5.5
		Top5 (%)	65	65	60.5	0	4.5
		Top10 (%)	86	86.5	79.5	-0.5	6.5
	Performance	MAP (%)	54.8	55.6	48.8	-0.8	6
	Effectiveness	MRR (%)	49	49.6	43.5	-0.6	5.5
		Mean	6	6	7.5	0	-1.5
		Median	3	3	3	0	0

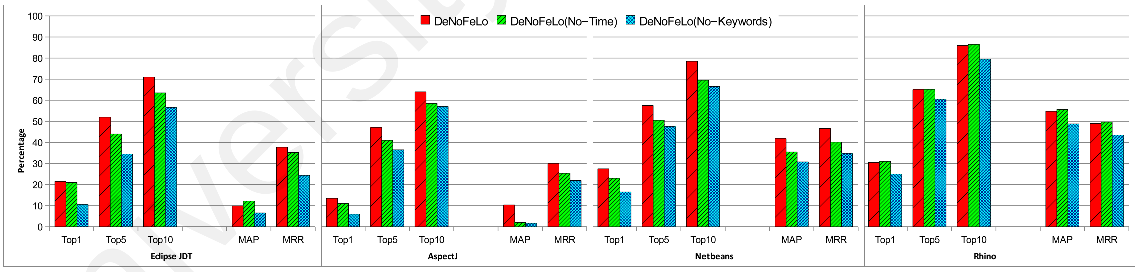


Figure 6.11: Results of ***De***NoFeLo, ***De***NoFeLo$_{No-Time}$, and ***De***NoFeLo$_{No-Keywords}$ for TopN, MAP and MRR metrics

(***De***NoFeLo$_{No-Time}$).

Figure 6.11 shows the bar chart graphs of the Top1 to Top10, MAP and MRR results. As shown in this figure, for the JDT, AspectJ and Netbeans projects, the results of ***De***NoFeLo were reduced when no time-metadata was considered. Moreover, the effectiveness results on the mean and median show that using ***De***NoFeLo$_{No-Time}$ means checking an average of 15 more locations in the worst case for these subject systems,

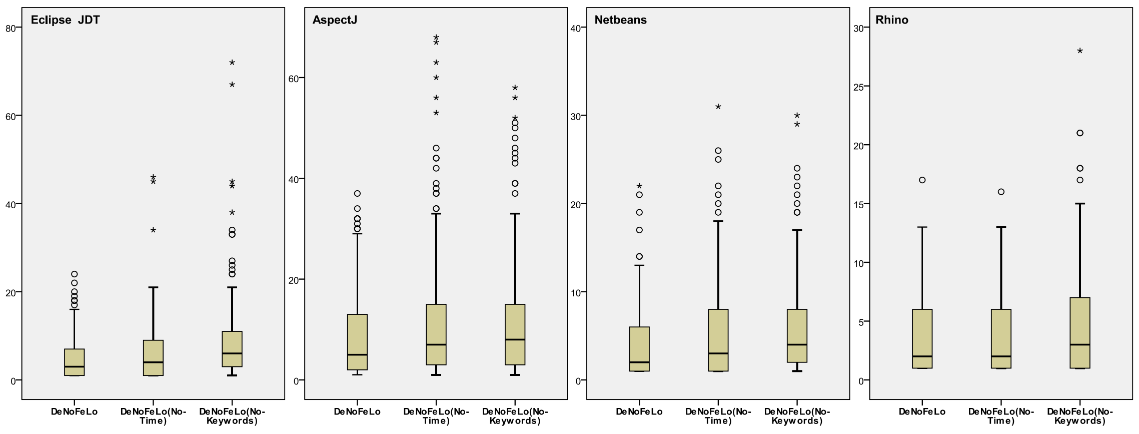


Figure 6.12: Results of ***De***NoFeLo, ***De***NoFeLo$_{No-Time}$, and ***De***NoFeLo$_{No-Keywords}$ for effectiveness metric

compared to ***De***NoFeLo. Figure 6.12 presents the box plot graphs of the effectiveness results and shows the need for more developer effort when using the ***De***NoFeLo$_{No-Time}$ method on most of the subject systems.

Examination of the ***De***NoFeLo results when both the time and keyword parameters are removed in term weighting shows that the accuracy, performance and effectiveness are further reduced on all the subject systems. According to the results, ***De***NoFeLo$_{No-Keywords}$ is up to 15% less accurate than ***De***NoFeLo and 7% less accurate than ***De***NoFeLo$_{No-Time}$. The performance and effectiveness of ***De***NoFeLo$_{No-Keywords}$ are respectively up to 11% and 13% lower than ***De***NoFeLo. It is also up to 7% and 11% lower than ***De***NoFeLo$_{No-Time}$. Comparing the results presented in Table 6.12 and Figure 6.11 show a degradation in the performance for ***De***NoFeLo$_{No-Keywords}$ over both ***De***NoFeLo and ***De***NoFeLo$_{No-Time}$ for all the subject systems including the Rhino project. Furthermore, the comparison of the median of the effectiveness results in Figure 6.12 shows a considerable difference between ***De***NoFeLo and ***De***NoFeLo$_{No-Keywords}$ on all the subject systems as there is a need for checking an average of 15 more locations in the worst case when using ***De***NoFeLo$_{No-Keywords}$ for feature location over ***De***NoFeLo.

On the other hand, the comparison of the ***De***NoFeLo$_{No-Time}$ with the feature location baseline approaches, SUM and VSM, also shows the significant improvement in feature location. The experiment revealed that ***De***NoFeLo$_{No-Time}$ outperforms the accuracy of both SUM and VSM by as much as 47% and 40%, respectively. Furthermore, the performance of the baseline approaches was outperformed by up to 39% and 23%, respectively. Additionally, ***De***NoFeLo$_{No-Time}$ is around 27% and 22% more effective than SUM and VSM, respectively. In general, the results indicate the significant effect of the considering developers' expertise for improving the accuracy of feature location.

6.2.2 Statistical Results

In this section, the statistical analyses of all of the obtained results from the descriptive analysis of the ***De***NoFeLo method are discussed in detail. First, normality tests were conducted to help determining a suitable statistical comparison test. To determine the normality of the results, the value of the Skewness and Kurtosis and also the p-value of the Shapiro-Wilk normality test were investigated. For the normal distributed data the Skewness and Kurtosis values are between -2 and +2 and the Shapiro-Wilk p-value is higher than α ($\alpha = 0.05$). Otherwise, the data is considered as the non-normal distributed.

Table 6.13 reports the normality test results for the experiment that performed to evaluate the ***De***NoFeLo method. The values of the Skewness and Kurtosis for the accuracy results are between -2 and +2, thus indicating that the accuracy results are normally distributed. Furthermore, the p-value of the Shapiro-Wilk confirms the normality of the accuracy results. Since, the accuracy results are normally distributed, the parametric paired T-test was conducted for a statistical comparison of the two approaches. Since, the ***De***NoFeLo method is evaluated in three different forms[3], ***De***NoFeLo, ***De***NoFeLo$_{No-Time}$, ***De***NoFeLo$_{No-Keywords}$, for the statistical analysis of this method, the repeated measure ANOVA with a Least Significant Difference (LSD) test as the post-hoc test was conducted.

[3]The results of the ***De***NoFeLo with and without two of main parameters, time and keywords is analyzed.

Table 6.13: Results of normality test for ***De***NoFeLo's experiment

Metric	Project	Method/Approach	Skewness	Kurtosis	Shapiro-Wilk
Accuracy	**Eclipse JDT**	DeNoFeLo	-0.649	-0.485	0.508
		DeNoFeLo$_{No-Time}$	-0.451	-0.61	0.794
		DeNoFeLo$_{No-Keywords}$	-0.304	-1.297	0.566
	AspectJ	DeNoFeLo	-0.8	-0.163	0.39
		DeNoFeLo$_{No-Time}$	-0.603	-0.785	0.528
		DeNoFeLo$_{No-Keywords}$	-0.643	-0.154	0.758
	Netbeans	DeNoFeLo	-0.587	-0.419	0.781
		DeNoFeLo$_{No-Time}$	-0.572	-0.665	0.625
		DeNoFeLo$_{No-Keywords}$	-0.622	-0.496	0.604
	Rhino	DeNoFeLo	-0.653	-0.259	0.683
		DeNoFeLo$_{No-Time}$	-0.723	-0.259	0.515
		DeNoFeLo$_{No-Keywords}$	-0.754	-0.033	0.611
Effectiveness	**Eclipse JDT**	DeNoFeLo	12.496	164.814	0
		DeNoFeLo$_{No-Time}$	11.461	144.42	0
		DeNoFeLo$_{No-Keywords}$	11.502	145.02	0
	AspectJ	DeNoFeLo	10.519	116.669	0
		DeNoFeLo$_{No-Time}$	8.186	73.777	0
		DeNoFeLo$_{No-Keywords}$	11.413	141.147	0
	Netbeans	DeNoFeLo	6.199	43.405	0
		DeNoFeLo$_{No-Time}$	4.285	19.623	0
		DeNoFeLo$_{No-Keywords}$	3.56	12.599	0
	Rhino	DeNoFeLo	5.469	35.579	0
		DeNoFeLo$_{No-Time}$	5.318	33.369	0
		DeNoFeLo$_{No-Keywords}$	4.006	18.943	0

For the effectiveness metric, the Skewness and Kurtosis values indicate the non-normality of the effectiveness results. The results of the Shapiro-Wilk normality test on the effectiveness metric show the p-value to be lower than α (p-value < 0.05), meaning that the effectiveness results are not normally distributed. With respect to the normality test results, the non-parametric Wilcoxon test was used for comparing the results of two approaches and the Friedman test was used for comparing multiple results.

Table 6.14 shows the results of the statistical tests for the comparison of the ***De***NoFeLo with the SUM, VSM, and ***Ti***NoFeLo. The statistical test of the results of ***De***NoFeLo for both the accuracy and effectiveness metrics indicates the significance of the differences with both the SUM and the VSM, as the p-values for all cases are less than 0.05. Thus, all the corresponding null hypotheses are rejected and the alternative hypotheses are accepted

Table 6.14: Results of statistical test for ***De***NoFeLo comparing with the baseline approaches and ***Ti***NoFeLo method

Metric	Project	Two Sample			Effect Size	
		Hypothesis	**Test Type**	**P-value**	**Type**	**Value**
Accuracy	**Eclipse JDT**	$H_{0,DeNoFeLovs.SUM}$	T-test	0	Hedgens' g	3.278
		$H_{0,DeNoFeLovs.VSM}$		0		2.72
		$H_{0,DeNoFeLovs.TiNoFeLo}$		0		0.751
	AspectJ	$H_{0,DeNoFeLovs.SUM}$		0		1.289
		$H_{0,DeNoFeLovs.VSM}$		0		0.806
		$H_{0,DeNoFeLovs.TiNoFeLo}$		0*		0.463
	Netbeans	$H_{0,DeNoFeLovs.SUM}$		0		1.718
		$H_{0,DeNoFeLovs.VSM}$		0		1.432
		$H_{0,DeNoFeLovs.TiNoFeLo}$		0.002		0.208
	Rhino	$H_{0,DeNoFeLovs.SUM}$		0		1.658
		$H_{0,DeNoFeLovs.VSM}$		0		1.269
		$H_{0,DeNoFeLovs.TiNoFeLo}$		0.008		0.284
Effectiveness	**Eclipse JDT**	$H_{0,DeNoFeLovs.SUM}$	Wilcoxon	0	Cliff's delta	0.657
		$H_{0,DeNoFeLovs.VSM}$		0		0.537
		$H_{0,DeNoFeLovs.TiNoFeLo}$		0		0.175
	AspectJ	$H_{0,DeNoFeLovs.SUM}$		0		0.335
		$H_{0,DeNoFeLovs.VSM}$		0		0.233
		$H_{0,DeNoFeLovs.TiNoFeLo}$		0*		0.157
	Netbeans	$H_{0,DeNoFeLovs.SUM}$		0		0.413
		$H_{0,DeNoFeLovs.VSM}$		0		0.352
		$H_{0,DeNoFeLovs.TiNoFeLo}$		0.025		0.055
	Rhino	$H_{0,DeNoFeLovs.SUM}$		0		0.38
		$H_{0,DeNoFeLovs.VSM}$		0		0.348
		$H_{0,DeNoFeLovs.TiNoFeLo}$		0.002		0.052

with high confident level. The effect size results suggest a large practical significance in the differences of accuracy between ***De***NoFeLo and the feature location baseline approaches, SUM and VSM. In terms of the effectiveness assessment, the effect size results for ***De***NoFeLo suggest that the significance level of the differences is mostly in the range of medium to large for the subject systems.

For the comparison of the accuracy and effectiveness of ***De***NoFeLo and ***Ti***NoFeLo, the statistical test shows the significance of the differences on all the subject systems that leads to the rejection of the $H_{0,DeNoFeLovs.TiNoFeLo}$ and acceptance of the corresponding alternative hypothesis. However, the statistical test indicates the significance of the differences between ***De***NoFeLo and ***Ti***NoFeLo on the AspectJ project, which is marked with an asterix, the comparison of the mean of the methods' results indicates that ***Ti***NoFeLo obtained better results. As shown in previous section, this exception is due to the low number of the developers working on this project. The discussion on the reasons of this issue is presented in Section 6.2.3.

The effect size results for the accuracy and effectiveness assessment on the AspectJ project indicate a small practical significance level between the results for ***De***NoFeLo and ***Ti***NoFeLo. For the comparison of the accuracy of ***De***NoFeLo and ***Ti***NoFeLo on the other subject systems, the effect size results indicate the significance level to be between small and medium. The effect size results for effectiveness results suggest that the significance level of the differences between the ***De***NoFeLo and ***Ti***NoFeLo is in the range of small to negligible for the JDT, Netbeans, and Rhino projects. As mentioned earlier, the effect sizes for the effectiveness measurements suggest a lower significance level when compared with the accuracy measurements, which is due to the non-normal distribution of the effectiveness results. The effect size value has a direct relationship with the outliers in the data and the outliers lead to a misleading effect size calculation. In these cases, the effectiveness results contain a large set of outliers, and these cannot be removed due to their usefulness and importance for evaluating the effectiveness metric. Therefore, the effect size cannot reflect the real level of improvement for the effectiveness results.

Table 6.15 presents the results of the statistical analysis for considering the use of time and keywords in ***De***NoFeLo. Since, ***De***NoFeLo is investigated in three different forms, the repeated measure ANOVA was conducted to investigate the significance of the difference between adding the two parameters of time and keywords. As shown in Table 6.15, all the p-values for the multiple group comparison of the accuracy results indicate a significance of the differences. For the two sample comparison, the p-values of the pair wise comparisons are reported. All the p-values on the accuracy measurement

Table 6.15: Results of statistical test for ***De***NoFeLo when time and keywords are not taken into account

Metric	Project	Multiple Sample		Two Sample			Effect Size	
		Test Type	P-value	Hypothesis	Test Type	P-value	Type	Value
Accuracy	**Eclipse JDT**	Repeated Measure ANOVA	0	$H_{0,}$DeNoFeLo vs. DeNoFeLo(No-Time)	LSD	0	Hedgens' g	0.396
				$H_{0,}$DeNoFeLo vs. DeNoFeLo(No-Keywords)		0		0.915
				$H_{0,}$DeNoFeLo(No-Time) vs. DeNoFeLo(No-Keywords)		0		0.597
	AspectJ		0	$H_{0,}$DeNoFeLo vs. DeNoFeLo(No-Time)		0		0.378
				$H_{0,}$DeNoFeLo vs. DeNoFeLo(No-Keywords)		0		0.556
				$H_{0,}$DeNoFeLo(No-Time) vs. DeNoFeLo(No-Keywords)		0.003		0.186
	Netbeans		0	$H_{0,}$DeNoFeLo vs. DeNoFeLo(No-Time)		0		0.424
				$H_{0,}$DeNoFeLo vs. DeNoFeLo(No-Keywords)		0		0.621
				$H_{0,}$DeNoFeLo(No-Time) vs. DeNoFeLo(No-Keywords)		0		0.214
	Rhino		0	$H_{0,}$DeNoFeLo vs. DeNoFeLo(No-Time)		0.005*		0.067
				$H_{0,}$DeNoFeLo vs. DeNoFeLo(No-Keywords)		0		0.305
				$H_{0,}$DeNoFeLo(No-Time) vs. DeNoFeLo(No-Keywords)		0		0.37
Effectiveness	**Eclipse JDT**	Friedman	0.001	$H_{0,}$DeNoFeLo vs. DeNoFeLo(No-Time)	Wilcoxon	0	Cliff's delta	0.075
				$H_{0,}$DeNoFeLo vs. DeNoFeLo(No-Keywords)		0		0.223
				$H_{0,}$DeNoFeLo(No-Time) vs. DeNoFeLo(No-Keywords)		0.003		0.122
	AspectJ		0	$H_{0,}$DeNoFeLo vs. DeNoFeLo(No-Time)		0		0.107
				$H_{0,}$DeNoFeLo vs. DeNoFeLo(No-Keywords)		0.007		0.126
				$H_{0,}$eNoFeLo(No-Time) vs. DeNoFeLo(No-Keywords)		0.208*		0.004
	Netbeans		0	$H_{0,}$DeNoFeLo vs. DeNoFeLo(No-Time)		0		0.118
				$H_{0,}$DeNoFeLo vs. DeNoFeLo(No-Keywords)		0		0.194
				$H_{0,}$DeNoFeLo(No-Time) vs. DeNoFeLo(No-Keywords)		0.011		0.006
	Rhino		0.003	$H_{0,}$DeNoFeLo vs. DeNoFeLo(No-Time)		0.139*		0.016
				$H_{0,}$DeNoFeLo vs. DeNoFeLo(No-Keywords)		0.001		0.092
				$H_{0,}$DeNoFeLo(No-Time) vs. DeNoFeLo(No-Keywords)		0		0.162

show significance in the differences. However, the p-value of the statistical test on the Rhino project indicates the significance of the differences, the comparison of the mean of accuracy results reveals that ***De***NoFeLo$_{No-Time}$ works more accurate than ***De***NoFeLo on this project. This issue is due to the low evolution speed of the project which was discussed in Section 6.1.3. The effect size for the accuracy results on ***De***NoFeLo versus ***De***NoFeLo$_{No-Time}$ suggests a small practical significance for JDT, AspectJ, and Netbeans. For the Rhino project, the effect size is negligible. The accuracy effect size indicates the significance level between small and large for ***De***NoFeLo versus ***De***NoFeLo$_{No-Keywords}$.

For the effectiveness measurement, all the p-values for the multiple group comparison show a significance of the differences. For the comparison of the two samples, almost all the p-values except ***De***NoFeLo$_{No-Time}$ versus ***De***NoFeLo$_{No-Keywords}$ for AspectJ, show the significance of the differences. However, the p-value of the statistical test on the Rhino project indicates the significance of the differences, the comparison of the mean of effectiveness results reveals that ***De***NoFeLo$_{No-Time}$ works more accurate than ***De***NoFeLo on this project. For the case of comparing ***De***NoFeLo$_{No-Time}$ and ***De***NoFeLo$_{No-Keywords}$ for AspectJ, which is marked with an asterix in the table, the reasons of not significance of the difference will be discussed in the next section. The effect size for the effectiveness values shows the significance level of difference between negligible and small which is likely due to the non-normal distribution of the results.

For the effectiveness results on the Rhino project, the p-value is greater than 0.05, indicating that the differences are not significant. The effect size also shows a negligible improvement. In terms of an accuracy assessment for the Rhino project, the statistical results show a significance of differences. Recall that the accuracy results for ***De***NoFeLo$_{No-Time}$ are around 0% to 2% better than ***De***NoFeLo, which is very low. Also, the effect size shows a negligibility of the differences. In general, the statistical analysis of the results shows that both the improvements in effectiveness and in accuracy for almost all of the cases are remarkably significant.

6.2.3 Discussion

As mentioned in Chapter 1, in a well-organized software project, each requirement is implemented in a specific set of source code files and there is a set of project developers who are related to the source code files and the corresponding requirements. On the other hand, in the typical text analysis based feature location process, the context of the source code files is analyzed to find the similarity between the source code files and a desired requirement or software feature. According to the relationship between the source code files and the project developers who are working on the files, the context of the files can also be analyzed from the aspect of developers.

Having two or more developers working on a set of related requirements/features on the same file shows a concentration within the corresponding file for these requirements, as well as the importance of the requirements within the file. To find the relation between the requirements and the developers, the context of the developer's expertise on the source code files are analyzed. Consideration of the context of developers' expertise not only provides the capability for investigating changes to the source code files during a project lifetime, but also helps to identify important requirements in these files.

The experimental evaluation of the proposed developer-based method indicates that ***De***NoFeLo obtained better results in all respects i.e. accuracy, performance, and effectiveness, when compared to the state-of-the-art of IR-based feature location approaches, SUM and VSM. These approaches treated as the location-based baseline approaches. More specifically, the statistical analysis of the accuracy and effectiveness results of the experiments indicates the significance of differences that lead to the rejection of the null hypotheses of $H_{0,DeNoFeLovs.SUM}$ and $H_{0,DeNoFeLovs.VSM}$. Accordingly, the corresponding alternative hypotheses are supported with respect to the significance of the differences between the proposed developer-based method, ***De***NoFeLo, and the location-based baseline approaches, SUM and VSM. Moreover, the effect sizes mostly suggest a significance level in the range of medium to large. In general, these results emphasize the positive effect that investigating developers' expertise has for feature location. These results lead to answering **De.RQ**$_1$ and concluding that ***De***NoFeLo outperforms the feature location baseline approaches. In other words, analyzing the developers' expertise for locating the software features results in an improvement in feature location process over the location-based baseline approaches.

Furthermore, the investigation of time complexity of ***De***NoFeLo and VSM shows that both of these approaches are in order of $|F| \times |T|$ (F is the number of source code files in the corpus and T is the average of the number of terms in the files). In comparison with SUM that uses Dirichlet smoothing method, SUM is in the order of $|F| \times |q|$ (F is the number of source code files in the corpus and q is the average of the number of terms in the given query) (Zhai & Lafferty, 2004). This means that SUM can be considered as efficient as ***De***NoFeLo in terms of time complexity.

Moreover, the results of ***De***NoFeLo are compared with the results of ***Ti***NoFeLo in order to evaluate the proposed developer-based method compared to the time-based baseline method. The investigation of these results enhances to find the answer of **De.RQ**$_2$. The comparison of the experimental results obtained from ***De***NoFeLo and ***Ti***NoFeLo shows an improvement in all aspects for most of the cases with two exceptions.

The first exception found in the comparison of ***De***NoFeLo and ***Ti***NoFeLo is related to the results of the experiment on the AspectJ project. These results show that ***Ti***NoFeLo performs better than ***De***NoFeLo on AspectJ. This exception is due to the low number of developers working on the AspectJ project as shown in Table 5.1. In this project, eight developers are working on a large number of source code files, around 6000 Java files. This means that the ratio of the developers over the source code locations in this software project is very low. This makes it challenging to analyze the source code data from the developer-aspect. In this case, the average size of data in developer's expertise profiles is large and the likelihood of similarity of the context of a given change request with the context of a large size of developers' expertise will be high. Thus, it is possible to lead to miscalculation of high similarity between the given change request and irrelevant source code locations. This issue results in misleading of the feature location method in identifying the correct source code location and consequently, reduction of the accuracy of the feature location process.

In the other words, in the developer-based feature location process, the data in source code file is classified in the developer expertise profiles based on the developers who are working on the files. Accordingly, a low ratio of the project developers to the source code files leads the developer-based method to create large sizes of profiles. Finding the similarity between the data in large size profiles and the desired change request will challenge the method to identify the related developer expertise profile, hence results in a low accuracy of ***De***NoFeLo compared to ***Ti***NoFeLo.

The second exception is related to the results of the methods on the Netbeans project. The results of ***De***NoFeLo and ***Ti***NoFeLo for this subject system are close. Further investigation of the properties of this project reveals that the number of developers to the source code files of the project is very high. As mentioned in Section 4.2, in time-based feature location process, the data recorded in source code files is classified based on time of usage. Since, the source code files are modified several times during project's lifetime, the number of classes in time-based process is high and the size of classes is typically small. On the other hand, in the project with high ratio of the developers over the source code files, a developer-based feature location process leads to small sizes of developer expertise profiles (small sizes of classes). In this case, the size of the classes in developer-based method will be close to the one in time-based method, and consequently, the performance of these two methods will be close.

Another interesting finding is related to the obtained results from the Rhino project. As discussed in Sections 6.1.1.2 and 6.1.3, ***Ti***NoFeLo was not able to obtain highly accurate results for Rhino which is due to the project having a gradual and slow evolution speed. In contrast, ***De***NoFeLo obtained a notably good result for this subject system. In ***De***NoFeLo, however, time is considered, the classification of the source code data is based on the developer, not time. In the other words, in ***De***NoFeLo, the data of Rhino project is analyzed from the developer-aspect, not the time-aspect. As mentioned earlier, in the ***Ti***NoFeLo method that analyzes the data from the time-aspect, all the times that a term was used in a file were considered and the value of each term was determined based on the importance of the term at the usage time. In this case, a term that was used several times in different commits of a file was separately weighted based on its values in the associated commits. It was demonstrated in Section 6.1 that consideration of the changes of the data over time leads to significant improvements in the accuracy of feature location process.

Similar to ***Ti***NoFeLo, the ***De***NoFeLo method considers the changes of the developer's expertise over time in developer-based feature location process. However, unlike ***Ti***NoFeLo, only the latest time that a term was used by a developer is considered in the ***De***NoFeLo method. In this case, the most recent time when the developer works on the related subject within a file is taken into account without dealing with the complexity of determining the value of the term for every usage time. Consideration of the changes in the source code made by the developer provides the ability of giving more value to the developer's recent activities and lower value to the developer's activities which were recorded further in the past. This issue avoids the need for threshold determination for the developer activity which is one of the challenges in bug assignment research area that results in missing part of developer's expertise (Shokripour et al., 2014).

The impact of time consideration is further investigated by evaluating the proposed method with and without time consideration (***De***NoFeLo versus ***De***NoFeLo$_{No-Time}$). The experimental results showed the superiority of ***De***NoFeLo on the JDT, AspectJ, and Netbeans projects. On the Rhino project, consideration of time did not improve the accuracy due to the slow evolution speed of the project. While time of the developer's activities is considered in identifying the location, the amount of changes of data over time affects the accuracy of ***De***NoFeLo (See Section 6.1). When the changes to the project over time are small, the use of time in weighting the terms may result in a degradation of the accuracy. As it was discussed in Section 6.1.3, Rhino has a small average number of changes per day in comparison to the other subject systems. Furthermore, as discussed in Section 6.2.2, the results of the statistical analysis reveal the significance of the differences between ***De***NoFeLo and ***De***NoFeLo$_{No-Time}$ on the subject systems except Rhino. Moreover, the effect sizes suggest a significance level of small for the accuracy metric and negligible for the effectiveness metric. As mentioned earlier, due to the non-normality of the effectiveness results, effect size suggests a little level of significance. In general, the descriptive and statistical results help to answer the **De.RQ**$_3$ regarding the investigation of the impact of time consideration in ***De***NoFeLo. With respect to the results for projects like JDT, AspectJ, and Netbeans that have a higher evolution speed, the use of time shows a significant improvement. However, for projects with low evolution speed, e.g. Rhino, time consideration results in reduction of the accuracy of the location identification process.

In order to investigate and find the answer of the last research question, **De.RQ**$_4$, the results of ***De***NoFeLo are compared to the results of ***De***NoFeLo$_{No-Keywords}$. As mentioned in Section 4.3.2, the keyword refers to the terms that appear in a small set of developer's expertise profiles. The descriptive results of the experiments indicate the reduction of the performance of the proposed method when the keyword factor is removed from the method. The degradation in the performance is for all the subject systems including the Rhino and AspectJ projects. The statistical analysis of the obtained results emphasizes the significance of the differences and the effect size suggests the significance level in the range of medium to large. The results of this experiment confirm the importance of giving a higher value to these terms which are used by a small set of developers, even if they have been used further in the past.

In general, the results of the experiments indicate the significant effect of analyzing the data from the aspect of developers, who recorded data in the source code files, for improving the accuracy, performance and effectiveness of feature location process. On the other hand, the consideration of time enhances a feature location process not only in a time-based method that analyzes the data from the time-aspect (***Ti***NoFeLo), but also in the developer-based method that analyzes the data from the developer-aspect (***De***NoFeLo). In summary, it can be concluded that the experiments' results indicate the superiority of the developer-based feature location over the time-based and location-based feature location.

6.3 Evaluation of *TiDe*NoFeLo Approach

The main goal of this thesis is proposing a feature location approach that considers the differences of the text data recorded in the repository and text data in a natural language context. As mentioned in Chapter 1, three perspectives are considered in order to improve the text analysis based feature location. These perspectives deal with the consideration of time-metadata, and developer-metadata and also the use of noun-only in text analysis process of feature location. The identified perspectives are separately applied and assessed in two proposed feature location methods, ***Ti***NoFeLo and ***De***NoFeLo. These two methods embody the proposed approach, ***TiDe***NoFeLo, to apply all the perspectives to the feature location in one approach. Accordingly, the proposed approach analyzes the data from both the aspects of time and developer, and also uses only the noun terms in order to more accurately locate the software features.

The impacts of considering each of the identified perspectives were assessed in the previous sections. In this section, the proposed approach, ***TiDe***NoFeLo, is evaluated over the location-based baseline approaches, i.e. SUM and VSM, the proposed time-based method, ***Ti***NoFeLo, and the proposed developer-based method, ***De***NoFeLo. To assess the ***TiDe***NoFeLo approach, a set of experiments were conducted based on the identified settings in Chapter 5. The obtained descriptive and statistical results of the experiments are reported in Sections 6.3.1 and 6.3.2, respectively. The discussion on the obtained results is presented in Section 6.3.3.

6.3.1 Descriptive Results

In this section, the results of the experiments of ***TiDe***NoFeLo are assessed based on the metrics explained in Section 5.2.1. The descriptive results are analyzed based on the identified research questions for ***TiDe***NoFeLo in Section 5.3.3. To remind, these research questions are represented in Table 6.16.

Table 6.16: Research questions of ***TiDe***NoFeLo approach

TiDe.RQ1	Does ***TiDe***NoFeLo outperform SUM and VSM as the location-based feature location approaches?
TiDe.RQ2	Does ***TiDe***NoFeLo outperform ***Ti***NoFeLo as a time-based feature location method?
TiDe.RQ3	Does ***TiDe***NoFeLo outperform ***De***NoFeLo as a developer-based feature location method?

According to these research questions, the results of the proposed approach are assessed against the results of SUM, VSM, ***Ti***NoFeLo and ***De***NoFeLo. The goal of these comparisons is assessing the proposed approach against the location-based, time-based, and developer-based approaches/methods, respectively. According to the research questions, in Section 6.3.1.1, the results of ***TiDe***NoFeLo are compared to the results of the location-based baseline approaches, i.e. SUM and VSM. Next, in Section 6.3.1.2, the proposed approach is evaluated against the proposed time-based method, ***Ti***NoFeLo, and the proposed developer-based method, ***De***NoFeLo.

6.3.1.1 ***TiDe*NoFeLo versus Baseline Approaches**

In the first step of evaluating the proposed method, the results of ***TiDe***NoFeLo are compared with the results obtained from the state-of-the-art IR-based approaches, SUM and VSM. Table 6.17 reports the results of ***TiDe***NoFeLo, SUM, and VSM for the accuracy (Top1 to Top10), performance (MAP) and effectiveness (MRR, Mean and Median) metrics. The last two columns of this table show the differences of the results of ***TiDe***NoFeLo with SUM and VSM, respectively, to enhance the comparison process.

As shown in Table 6.17, ***TiDe***NoFeLo is notably more accurate than SUM and VSM for the Top1 to Top10 ranked locations on all subject systems. The ***TiDe***NoFeLo outperforms the accuracy of SUM on Top10 location recommendation as much as 62%, 48%, 42%, and 37%, respectively on the JDT, AspectJ, Netbeans, and Rhino projects. In comparing to the VSM, the accuracy improvement is up to 54%, 44%, 36%, and 32%, respectively, for those subject systems. The accuracy results are also illustrated in Figure 6.13. As it can be seen in this figure, ***TiDe***NoFeLo significantly outperforms SUM and VSM on all the subject systems.

In terms of performance evaluation, the MAP results indicate a higher performance for ***TiDe***NoFeLo over SUM and VSM for all of the subject systems. For example, on the Eclipse JDT project, ***TiDe***NoFeLo's MAP is around six times that of the MAP of SUM and almost two times the MAP of VSM. Also, on the Netbeans project, the ***TiDe***NoFeLo's

Table 6.17: Results of ***TiDe***NoFeLo, SUM and VSM for accuracy, performance and effectiveness metrics

Project	**Metric**		**TiDeNoFeLo**	**SUM**	**VSM**	**Difference *TiDe*NoFeLo /SUM**	**Difference *TiDe*NoFeLo /VSM**
Eclipse JDT	**Accuracy**	**Top1** (%)	19.7	3.5	5.5	16.2	14.2
		Top5 (%)	53.7	11	17.5	42.7	36.2
		Top10 (%)	78.2	16.5	24	61.7	54.2
	Performance	**MAP** (%)	10.4	1.8	5.8	8.6	4.6
	Effectiveness	**MRR** (%)	38.2	8.4	12.9	29.8	25.3
		Mean	59.8	290.2	193.2	-230.4	-133.4
		Median	4.5	75.5	40	-71	-35.5
AspectJ	**Accuracy**	**Top1** (%)	16	7	15	9	1
		Top5 (%)	62.2	28	36.5	34.2	25.7
		Top10 (%)	89.4	41	45.5	48.4	43.9
	Performance	**MAP** (%)	14.4	6.3	9.2	8.1	5.2
	Effectiveness	**MRR** (%)	37.5	17.8	25.4	19.7	12.1
		Mean	6.7	38.9	43.1	-32.2	-36.4
		Median	4	14.5	13.5	-10.5	-9.5
Netbeans	**Accuracy**	**Top1** (%)	33.5	10	16	23.5	17.5
		Top5 (%)	64.4	34.5	37	29.9	27.4
		Top10 (%)	85.6	44	50	41.6	35.6
	Performance	**MAP** (%)	48.7	10.2	25.3	38.5	23.4
	Effectiveness	**MRR** (%)	50.9	24.8	29.1	26.1	21.9
		Mean	6.5	30.2	32.3	-23.7	-25.8
		Median	3	11	8	-8	-5
Rhino	**Accuracy**	**Top1** (%)	38.3	18.5	14	19.8	24.3
		Top5 (%)	73.4	37.5	43.5	35.9	29.9
		Top10 (%)	91	54.5	59	36.5	32
	Performance	**MAP** (%)	59.3	16.3	32.6	43	26.7
	Effectiveness	**MRR** (%)	52.1	30.4	29.7	21.7	22.4
		Mean	5.4	18	16.8	-12.6	-11.4
		Median	3	9	6	-6	-3

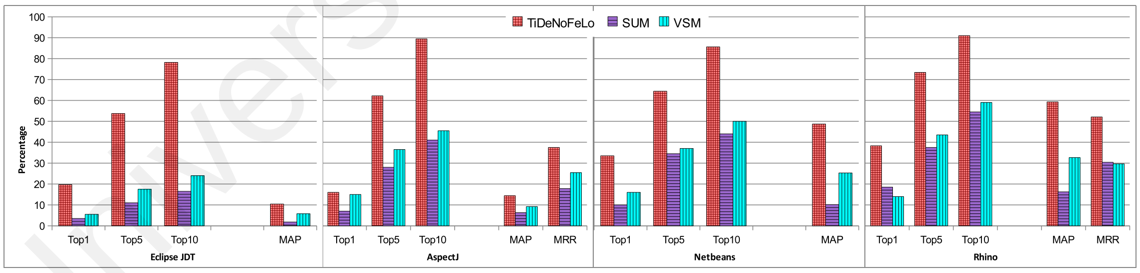


Figure 6.13: Results of ***TiDe***NoFeLo, SUM and VSM for TopN, MAP and MRR metrics

MAP is almost five times the MAP of SUM and two times the MAP of VSM.

For the MRR metric, which highlights the effectiveness of methods/approaches, ***TiDe***NoFeLo outperforms SUM and VSM. The improvement in comparing to the SUM is by as much as 30%, 20%, 26%, and 22%, on the JDT, AspectJ, Netbeans, and Rhino projects, respectively. In comparing to the VSM, the improvement is by up to 25%, 12%,

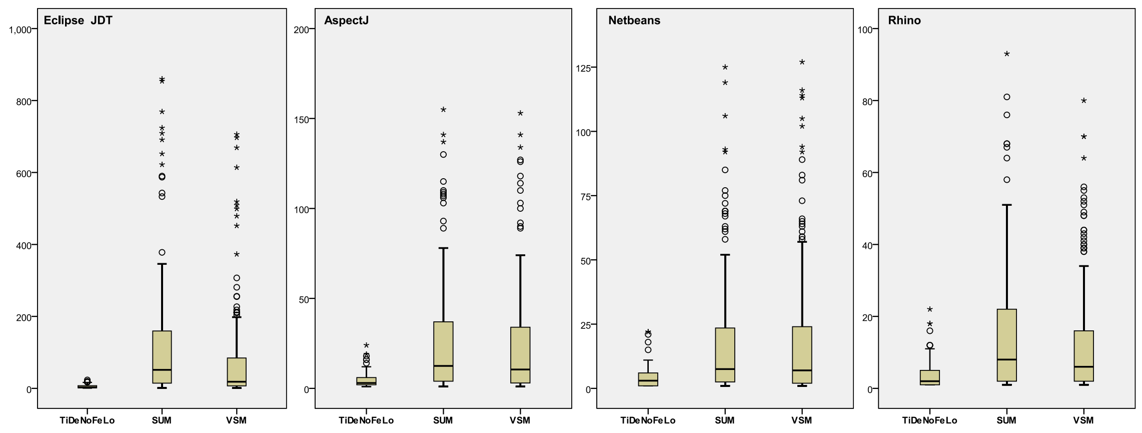


Figure 6.14: Results of ***TiDe***NoFeLo, SUM and VSM for effectiveness metric

22%, and 22%, on these projects. Figure 6.13 displays the bar chart diagrams of the results obtained from the MAP and MRR metrics for ***TiDe***NoFeLo, SUM, and VSM. The superiority of ***TiDe***NoFeLo over both the SUM and VSM is visible in this figure.

Further analysis of the approach effectiveness is performed by investigating the mean effectiveness values. The results show that identifying the location using SUM or VSM requires checking around 13 to 230 and 11 to 133 more locations, respectively, for the subject systems, when compared to ***TiDe***NoFeLo. Due to the non-normal distribution of the effectiveness results, in addition to examining the average number of locations that will need to be checked, the median of the effectiveness results also needs to be investigated. The median of the number of locations to check for SUM is 6 to 71 locations and, for VSM, it is 3 to 36 locations more than the median of ***TiDe***NoFeLo. Both the mean and median of the effectiveness indicate that ***TiDe***NoFeLo is more effective than SUM and VSM.

The box plot graph of the effectiveness results are shown in Figure 6.14. Low values in the box plots, which represent the positions of relevant files, suggest potentially less effort is needed by a developer to locate relevant files, because the ranks are among the first results returned by the feature location approach. The most effective approach for feature location is the approach with the lowest effectiveness value. The graphs have different scales due to the different number of files in the subject systems. To remind, the top and bottom boxes in these graphs represent the upper and lower quartiles, respectively, and the line between the boxes represents the median. The whiskers above and below the boxes denote the maximum and minimum effectiveness values, respectively. Due to the non-normal distribution of the effectiveness values and very high number of outliers that contain extreme values, the "outer fences"[4] were temporary removed from the results to make the differences between the approaches visible in the graphs. Due to the high number of outliers, the numbers of outliers still remain in the effectiveness results that are shown by the cycle and star symbols in the box plot diagrams. According to the effectiveness results presented in Table 6.17 and Figure 6.14, ***TiDe***NoFeLo is significantly more effective than SUM and VSM.

6.3.1.2 *TiDe*NoFeLo versus *Ti*NoFeLo and *De*NoFeLo

In this section, the results of the proposed approach are assessed by comparing with the proposed time-based and the proposed developer-based methods that embody the approach. Table 6.18 reports the results of ***TiDe***NoFeLo, ***Ti***NoFeLo, and ***De***NoFeLo for the accuracy (Top1 to Top10), performance (MAP) and effectiveness (MRR, Mean and Median) metrics. The last two columns of this table show the differences of the results of ***TiDe***NoFeLo with the ***Ti***NoFeLo and ***De***NoFeLo, respectively, to enhance the comparison process.

In Table 6.18, the experimental results of ***TiDe***NoFeLo indicate the outperformance of this approach over both the ***Ti***NoFeLo and ***De***NoFeLo methods on almost all the cases for accuracy, performance and effectiveness. The accuracy results of ***TiDe***NoFeLo on Top10 ranked locations shows the improvements of the accuracy of ***Ti***NoFeLo by as much as 24%, 12%, 14%, and 17% on the Eclipse JDT, AspectJ, Netbeans and Rhino projects,

[4]The formula for calculating the outer fences is Q3+3×(Q3-Q1).

Table 6.18: Results of ***TiDe***NoFeLo, ***De***NoFeLo and ***Ti***NoFeLo for accuracy, performance and effectiveness metrics

Project	**Metric**		**TiDeNoFeLo**	**DeNoFeLo**	**TiNoFeLo**	**Difference TiDe-NoFeLo /DeNoFeLo**	**Difference TiDe-NoFeLo /TiNoFeLo**
Eclipse JDT	**Accuracy**	**Top1 (%)**	19.7	21.5	10.5	-1.8	9.2
		Top5 (%)	53.7	52	44	1.7	9.7
		Top10 (%)	78.2	71	54.5	7.2	23.7
	Performance	**MAP (%)**	10.4	9.9	9	0.5	1.4
	Effectiveness	**MRR (%)**	38.2	37.7	26.4	0.5	11.8
		Mean	59.8	66.1	71	-6.3	-11.2
		Median	4.5	4.5	7	0	-2.5
AspectJ	**Accuracy**	**Top1 (%)**	16	13.5	14.5	2.5	1.5
		Top5 (%)	62.2	47	57	15.2	5.2
		Top10 (%)	89.4	64	77	25.4	12.4
	Performance	**MAP (%)**	14.4	10.4	12.2	4	2.2
	Effectiveness	**MRR (%)**	37.5	29.9	34.3	7.6	3.2
		Mean	6.7	22.5	7.6	-15.8	-0.9
		Median	4	6	4.5	-2	-0.5
Netbeans	**Accuracy**	**Top1 (%)**	33.5	27.5	23.5	6	10
		Top5 (%)	64.4	57.5	55.5	6.9	8.9
		Top10 (%)	85.6	78.5	72	7.1	13.6
	Performance	**MAP (%)**	48.7	41.8	38.6	6.9	10.1
	Effectiveness	**MRR (%)**	50.9	46.6	43.4	4.3	7.5
		Mean	6.5	8.5	9.4	-2	-2.9
		Median	3	3	3	0	0
Rhino	**Accuracy**	**Top1 (%)**	38.3	30.5	29	7.8	9.3
		Top5 (%)	73.4	65	62	8.4	11.4
		Top10 (%)	91	86	74	5	17
	Performance	**MAP (%)**	59.3	54.8	51.5	4.5	7.8
	Effectiveness	**MRR (%)**	52.1	49	47	3.1	5.1
		Mean	5.4	6.1	6.8	-0.7	-1.4
		Median	3	3	3	0	0

respectively. Compared to the ***De***NoFeLo method, the accuracy is improved by up to 7%, 25%, 7%, and 5% on the subject systems, respectively. Figure 6.15 shows the bar chart of the accuracy results on all the subject systems. The accuracy diagrams show that in some cases such as Eclipse JDT, ***De***NoFeLo works more accurately than ***Ti***NoFeLo, and in some cases such as AspectJ, ***Ti***NoFeLo works better. The bar charts of ***TiDe***NoFeLo results show that the proposed approach is more accurate than both the ***De***NoFeLo and ***Ti***NoFeLo methods.

In terms of performance evaluation, the MAP results of ***TiDe***NoFeLo indicate an improvement from 1% to 10% compared to ***Ti***NoFeLo and improvement from 0.5% to 7% compared to ***De***NoFeLo. The effectiveness evaluation on the MRR metric shows improvements around 3% to 12% compared to ***Ti***NoFeLo. The comparison of the MRR

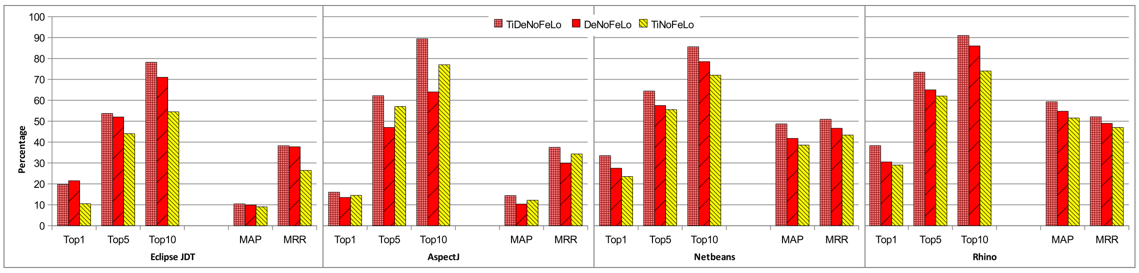


Figure 6.15: Results of ***TiDe***NoFeLo, ***De***NoFeLo and ***Ti***NoFeLo for TopN, MAP and MRR metrics

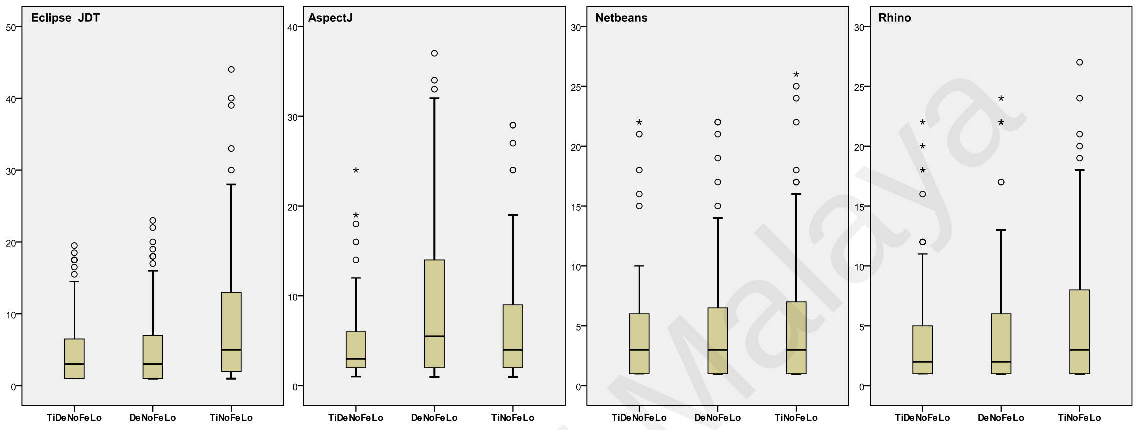


Figure 6.16: Results of ***TiDe***NoFeLo, ***De***NoFeLo and ***Ti***NoFeLo for effectiveness metric

results of ***TiDe***NoFeLo and ***De***NoFeLo indicates an improvement from 1% to 8% on the subject systems. Figure 6.15 also displays the MAP and MRR diagrams of the obtained results for this experiment. As shown in this figure the MAP and MRR results of the ***TiDe***NoFeLo approach are better than that of both the ***Ti***NoFeLo and ***De***NoFeLo methods in most of the cases.

Further investigation of effectiveness results obtained from the Mean and Median effectiveness metrics indicates a slight improvement over both the ***Ti***NoFeLo and ***De***NoFeLo methods. Figure 6.16 presents the box plot diagrams of the effectiveness results. As shown in this figure, the ***TiDe***NoFeLo approach performs slightly more effective than ***Ti***NoFeLo and ***De***NoFeLo on the subject systems. This means that the project developer needs to check an average of 11 more locations in the worst case when using ***Ti***NoFeLo, and needs checking an average of 16 more locations in the worst case when using ***De***NoFeLo.

6.3.2 Statistical Results

In this section, the statistical analysis of the obtained results from the ***TiDe***NoFeLo experiments is reported. First, the results of the normality test are reported in Table 6.19. As reported in this table, the results indicate the normality of the accuracy results and non-normality of the effectiveness results.

Table 6.19: Results of normality test for ***TiDe***NoFeLo's experiment

Metric	Project	Skewness	Kurtosis	Shapiro-Wilk
Accuracy	**Eclipse JDT**	-0.666	-0.247	0.694
	AspectJ	-0.826	-0.004	0.496
	Netbeans	-0.659	-0.375	0.704
	Rhino	-0.874	-0.257	0.319
Effectiveness	**Eclipse JDT**	6.866	51.127	0
	AspectJ	3.79	16.233	0
	Netbeans	5.049	30.68	0
	Rhino	6.357	55.272	0

As discussed in Section 5.2.2, the paired T-test is used for comparing two sets of normal distributed results. In the case of comparing multiple normal distributed results, the two-way ANOVA was conducted and a Least Significant Difference (LSD) test was used as the post hoc test. For the non-normal distributed results, the Wilcoxon test was used for comparing the results of two methods/approaches/techniques and the Friedman test was used for comparing multiple cases.

Table 6.20 reports the results of the statistical test conducted for comparing ***TiDe***NoFeLo with baseline approaches. The statistical test for the accuracy results shows the rejection of the null hypothesis. This means an acceptance of the corresponding alternative hypothesis. In the other words, there is a significant difference in the accuracy of ***TiDe***NoFeLo comparing to SUM and VSM. Furthermore, the values of the effect size for the accuracy measurement suggest a large practical significance on all of the subject systems. Similar results were found for the effectiveness results – which means the ef-

Table 6.20: Results of statistical test for ***TiDe***NoFeLo comparing with baseline approaches, SUM and VSM

Metric	**Project**	**Two Sample**			**Effect Size**	
		Hypothesis	**Test Type**	**P-value**	**Type**	**Value**
Accuracy	**Eclipse JDT**	$H_{0,TiDeNoFeLovs.SUM}$	T-test	0	Hedgens' g	3.118
		$H_{0,TiDeNoFeLovs.VSM}$		0		2.632
	AspectJ	$H_{0,TiDeNoFeLovs.SUM}$		0		1.842
		$H_{0,TiDeNoFeLovs.VSM}$		0		1.48
	Netbeans	$H_{0,TiDeNoFeLovs.SUM}$		0		2.135
		$H_{0,TiDeNoFeLovs.VSM}$		0		1.851
	Rhino	$H_{0,TiDeNoFeLovs.SUM}$		0		2.117
		$H_{0,TiDeNoFeLovs.VSM}$		0		1.68
Effectiveness	**Eclipse JDT**	$H_{0,TiDeNoFeLovs.SUM}$	Wilcoxon	0	Cliff's delta	0.707
		$H_{0,TiDeNoFeLovs.VSM}$		0		0.592
	AspectJ	$H_{0,TiDeNoFeLovs.SUM}$		0		0.534
		$H_{0,TiDeNoFeLovs.VSM}$		0		0.398
	Netbeans	$H_{0,TiDeNoFeLovs.SUM}$		0		0.462
		$H_{0,TiDeNoFeLovs.VSM}$		0		0.402
	Rhino	$H_{0,TiDeNoFeLovs.SUM}$		0		0.405
		$H_{0,TiDeNoFeLovs.VSM}$		0		0.379

fectiveness of ***TiDe***NoFeLo over SUM and VSM is remarkably significant for all subject system. The effect size values for the effectiveness metric indicated a medium to large practical significance for the subject systems. In general, the statistical analysis of the obtained results shows that both the improvements in effectiveness and the accuracy of ***TiDe***NoFeLo over SUM and VSM are significant and the effect sizes mostly suggest a large practical significance.

The statistical analysis of the results of ***TiDe***NoFeLo compared to ***Ti***NoFeLo and ***De***NoFeLo is reported in Table 6.21. First, the results are compared in general with a multiple group analysis. Note that all of the statistical results on both the accuracy and effectiveness for the multiple group analysis are less than α (p-value < 0.05). This shows the overall superiority of ***TiDe***NoFeLo with respect to accuracy and effectiveness for feature location. The post hoc analysis on the accuracy results of ***TiDe***NoFeLo compared to the both of ***Ti***NoFeLo and ***De***NoFeLo shows the significance of the differences on all the cases. For the effectiveness results, the post hoc analysis indicates that the differences

Table 6.21: Results of statistical test for ***TiDe***NoFeLo comparing with ***Ti***NoFeLo and ***De***NoFeLo

Metric	**Project**	Multiple Sample		Two Sample			Effect Size	
		Test Type	P-value	Hypothesis	Test Type	P-value	Type	Value
Accuracy	**Eclipse JDT**	Two-way ANOVA	0	$H_{0,TiDeNoFeLovs.TiNoFeLo}$ $H_{0,TiDeNoFeLovs.DeNoFeLo}$	LSD	0 0.015	Hedgens' g	0.859 0.154
	AspectJ		0	$H_{0,TiDeNoFeLovs.TiNoFeLo}$ $H_{0,TiDeNoFeLovs.DeNoFeLo}$		0 0		1.842 0.773
	Netbeans		0	$H_{0,TiDeNoFeLovs.TiNoFeLo}$ $H_{0,TiDeNoFeLovs.DeNoFeLo}$		0 0		0.6 0.372
	Rhino		0	$H_{0,TiDeNoFeLovs.TiNoFeLo}$ $H_{0,TiDeNoFeLovs.DeNoFeLo}$		0 0		0.686 0.359
Effectiveness	**Eclipse JDT**	Friedman	0	$H_{0,TiDeNoFeLovs.TiNoFeLo}$ $H_{0,TiDeNoFeLovs.DeNoFeLo}$	Wilcoxon	0 0.051*	Cliff's delta	0.216 0.001
	AspectJ		0	$H_{0,TiDeNoFeLovs.TiNoFeLo}$ $H_{0,TiDeNoFeLovs.DeNoFeLo}$		0 0		0.094 0.226
	Netbeans		0	$H_{0,TiDeNoFeLovs.TiNoFeLo}$ $H_{0,TiDeNoFeLovs.DeNoFeLo}$		0 0		0.353 0.067
	Rhino		0.000018	$H_{0,TiDeNoFeLovs.TiNoFeLo}$ $H_{0,TiDeNoFeLovs.DeNoFeLo}$		0 0		0.088 0.04

are significant in almost all the cases. On the Eclipse JDT project, it can be said that ***TiDe***NoFeLo is significantly more effective than ***De***NoFeLo with 94.9% confidence.

Furthermore, to complement inferential statistics, the effect size values (Hedges' g and Cliff's delta) are reported in Table 6.21. The effect size values for the accuracy of ***TiDe***NoFeLo against ***Ti***NoFeLo suggest a medium to large practical significance for the subject systems. Compared to ***De***NoFeLo, the effect size on the accuracy shows the significance level in the range of negligible to medium. For the effectiveness measurement, the effect size value reveals negligible to medium effect sizes in the case of comparing ***TiDe***NoFeLo and ***Ti***NoFeLo. Compared to ***De***NoFeLo, the effect size on the effectiveness suggests a negligible to small practical significance for the subject systems.

6.3.3 Discussion

As mentioned earlier, the proposed approach combined a time-based method, ***Ti***NoFeLo, and a developer-based method, ***De***NoFeLo. With respect to the identified research questions, to assess the robustness of the proposed approach, ***TiDe***NoFeLo was assessed over the location-based, time-based, and developer-based feature location approaches and methods. Accordingly, first, the ***TiDe***NoFeLo approach was assessed over the location-based baseline approaches, SUM and VSM. The analysis of the descriptive results for the ***TiDe***NoFeLo approach compared to the location-based baseline approaches shows the superiority of ***TiDe***NoFeLo on all identified metrics. Moreover, the statistical analysis of the results emphasizes the significance of the differences over the location-based baseline approaches, SUM and VSM. It means, the rejection of the null hypotheses of $H_{0,TiDeNoFeLovs.SUM}$ and $H_{0,TiDeNoFeLovs.VSM}$ and consequently, acceptance of the corresponding alternative hypotheses. Furthermore, the values of the effect size mostly suggest a medium to large practical significance for ***TiDe***NoFeLo when compared with SUM and VSM. These results lead to answering **TiDe.RQ**$_1$and concluding that ***TiDe***NoFeLo outperforms the location-based baseline approaches.

Furthermore, the time complexity of the proposed approach is evaluated against the baseline approaches. The time complexity of the ***TiDe***NoFeLo approach is the summation of time complexity of ***Ti***NoFeLo and ***De***NoFeLo. The summation of two similar time complexity results in the same complexity. Accordingly, ***TiDe***NoFeLo has same time order with VSM. Also, the comparison of ***TiDe***NoFeLo and SUM shows that SUM can be considered as efficient as ***TiDe***NoFeLo in terms of time complexity.

Regarding the second research questions identified for the ***TiDe***NoFeLo approach, the proposed approach is evaluated over the time-based baseline approach, ***Ti***NoFeLo. The descriptive results show that ***TiDe***NoFeLo improves the accuracy, performance, and effectiveness of ***Ti***NoFeLo on all the subject systems. The statistically analysis of the results indicates the significance of the differences between ***TiDe***NoFeLo and ***Ti***NoFeLo. This means the rejection of the null hypothesis of $\mathbf{H}_{0,TiDeNoFeLovs.TiNoFeLo}$ and acceptance of the corresponding alternative hypothesis. The effect size for the accuracy assessment suggests a medium to large practical significance and for the effectiveness measurement indicates the significance level of negligible to medium. The descriptive and statistical analyses of these results lead to the answer of the **TiDe.RQ**$_2$ that indicates the superiority of ***TiDe***NoFeLo over ***Ti***NoFeLo that treated as the time-based baseline method.

On the other hand, the comparison of the proposed approach with the developer-based baseline method, ***De***NoFeLo, also shows the outperformance of ***TiDe***NoFeLo over ***De***NoFeLo. The descriptive results on all the subject systems indicate the improvement of the results by using ***TiDe***NoFeLo over ***De***NoFeLo and the statistical tests confirm the significance of the differences. Accordingly, the null hypothesis of $\mathbf{H}_{0,TiDeNoFeLovs.DeNoFeLo}$ is rejected and the alternative hypothesis of $\mathbf{H}_{a,TiDeNoFeLovs.DeNoFeLo}$ is supported. The effect size suggests a practical significance in the range of negligible to medium for the accuracy evaluation and a significance level in the range of negligible to small for the effectiveness measurement. All these results help to find the answer of the last research question of **TiDe.RQ**$_3$ which means that the ***TiDe***NoFeLo approach outperforms ***De***NoFeLo as a developer-based baseline feature location method.

The other interesting issues are related to the weaknesses of ***Ti***NoFeLo and ***De***NoFeLo in addressing the feature location process on different subject systems. First issue is related to the results of ***Ti***NoFeLo on the Rhino project. As it was mentioned in Section 6.1.1.2, the results of the experiments show that the consideration of time is not able to improve the feature location process for projects such as Rhino with a low evolution speed. This exception is due to the low ratio of modified data and the low number of fixed change requests over time that affect a time-based method. Therefore, the use of time-based methods is not recommended for projects with a low evolution speed, such as Rhino.

The second issue is related to the results of ***De***NoFeLo on the AspectJ project. The experiments on ***De***NoFeLo assessment, reported in Section 6.2.1.2, reveal the weakness of this method on locating the features in the source code of the projects with the low ratio of the developers over the source code files such as the AspectJ project. In this case, the average size of classes in developer expertise profiles is large and the likelihood of similarity of a given change request with a large size of profiles will be high. This issue leads to the reduction of the accuracy of the feature location process.

Both these weaknesses are addressed in the proposed approach by combining ***Ti***NoFeLo and ***De***NoFeLo. It is due to the analysis of data from both the aspects of time and developer in the ***TiDe***NoFeLo approach. Based on the experimental results, it can be concluded that the combination of time and developer-aspect of data analysis in a noun-based feature location approach makes a significant improvement in the accuracy, performance and effectiveness of feature location process applied on a large set of different software projects with various evolution speeds and various ratio of developers over source code files.

6.4 Summary

In this thesis, to assess the impacts of analyzing the data from the aspects of time and developer and also using only the nouns, a large set of experiments were conducted. In these experiments the impact of time, developer and noun consideration was analyzed on several software projects with different scales, different number of developers, and various evolution speeds. The obtained results from the experiments were investigated from different points of view and the findings were analyzed in depth. Discussion on the analysis of the findings was explained in Section 6.1.3, 6.2.3, and 6.3.3. Summary of the results and the interesting findings were reviewed in the rest of this section.

Table 6.22 summarizes the comparison of the results obtained from all the experiments conducted for evaluating the proposed methods and approach. As explained earlier, a set of experiments were conducted to evaluate each of the proposed methods and approach. Thus, the corresponding boxes in the table are filled up with the comparison values and the other boxes are empty. The reported results in this table are estimated from subtracting the values of the accuracy (for Top10 ranked locations), performance (for MAP metric), and effectiveness (for MRR metric) on the obtained results for each of the methods/approaches. In this table, the best values of the comparison of the results obtained from the methods and approaches on different subject systems are reported.

Table 6.22: Comparison of the results for all the experiments

Approach /Method	Metrics	SUM	VSM	TiNoFeLo	TiNoFeLo (TF-IDF)	TiNoFeLo (All-Term)	DeNoFeLo	DeNoFeLo (No-Time)	DeNoFeLo (No-Keywords)
TiNoFeLo	**Accuracy(%)**	38	32	-	14	26	-	-	-
	Performance(%)	35	19	-	9	49	-	-	-
	Effectiveness(%)	19	17	-	12	23	-	-	-
DeNoFeLo	**Accuracy(%)**	55	47	17	-	-	-	9	17
	Performance(%)	39	22	3	-	-	-	8	11
	Effectiveness(%)	29	25	11	-	-	-	6	13
TiDeNoFeLo	**Accuracy(%)**	62	54	24	-	-	25	-	-
	Performance(%)	43	27	10	-	-	7	-	-
	Effectiveness(%)	30	25	12	-	-	8	-	-

As reported in Table 6.22, the experiment's results indicates a positive impact of using noun-only for improving the accuracy, performance and effectiveness of a feature location process on the projects of different scales. Furthermore, using only the nouns, instead of all types of terms, not only provides enough information to identifying the correct location for software features, but also significantly reduces the size of the datasets. Also, using the noun terms summarizes the source code data and enhances MSR tasks (Haiduc et al., 2010). In summary, the use of only the noun terms has a remarkable positive impact in the feature location process.

In terms of time consideration, the analysis of experimental results reported in Table 6.22 indicates the significant improvement made by the time-aspect analysis of the data for feature location. Moreover, the time-aspect analysis of data provides the ability of considering the evolution of the text data over time. An effective factor on the accuracy of time-aspect analysis of data is the evolution speed of the software project over time. This is due to the volume of modified data over time, which is shown by the number of terms that were recorded at the same time in a source code file, impacts the value of a term in the time-aspect analysis of data. Accordingly, the performance of the time-based feature location process is affected by the evolution speed of software projects. This means that for the projects with high evolution speed, the performance of time-based feature location process is better. When the volume of the data over time is small, the time-aspect analysis of data may not be able to provide considerable improvement over a location-based feature location process.

In terms of developer consideration, the investigation of experimental results shows the significant improvement made by the developer-based feature location process over both the time-based and location-based feature location (See Table 6.22). Developer-aspect analysis of data not only provides the capability for investigating changes in the source code made by the developer during a project lifetime, but also helps to identify important requirements in these elements. However, the developer-aspect of data analysis on the projects with low ratio of developers over source code files is not able to provide considerable improvement in feature location process due to the large sizes of classes in the IR sense. Furthermore, the high ratio of the number of developers over source code files leads to a small size of classes. In this case, the performance of developer-based and time-based feature location process will be close. Accordingly, it can be said that the performance of the developer-based feature location process is impacted by the ratio of developers over source code files.

Finally, as it is shown in Table 6.22, the combination of the developer-based and time-based feature location methods results in an improved feature location approach. The ***TiDe***NoFeLo approach addresses the limitations of both the developer-based and time-based process since it analyzes the data from both the aspects of developer and time, then sum up the retrieved results. Accordingly, the proposed approach is able to accurately locate the software feature in a large set of projects with various properties, i.e. scales, evolution speeds, and number of developers. All in all, the proposed approach is able to provide high accuracy, performance and effectiveness on a large set of projects with different evolution speeds and different ratios of developers over source code files. Thus, it can be concluded that consideration of time-metadata and developer-metadata as two important pieces of metadata and the use of only the nouns significantly improves the accuracy, performance and effectiveness of text analysis-based feature location. It means that the identified research problem regarding with the low accuracy of text analysis of feature location is addressed by proposing the TiDeNoFeLo approach.

CHAPTER 7: CONCLUSION

Feature location aims at identifying an initial location in the source code of a software system pertinent to a particular software feature (Biggerstaff et al., 1993; Rajlich & Wilde, 2002). According to the recent survey on feature location literature (Dit et al., 2013), most of the existing feature location approaches are based, at least in part, on text analysis. Text analysis process is a fundamental step in most of the feature location approaches. Thus, the overall accuracy of feature location would be directly affected by the accuracy of the text analysis process.

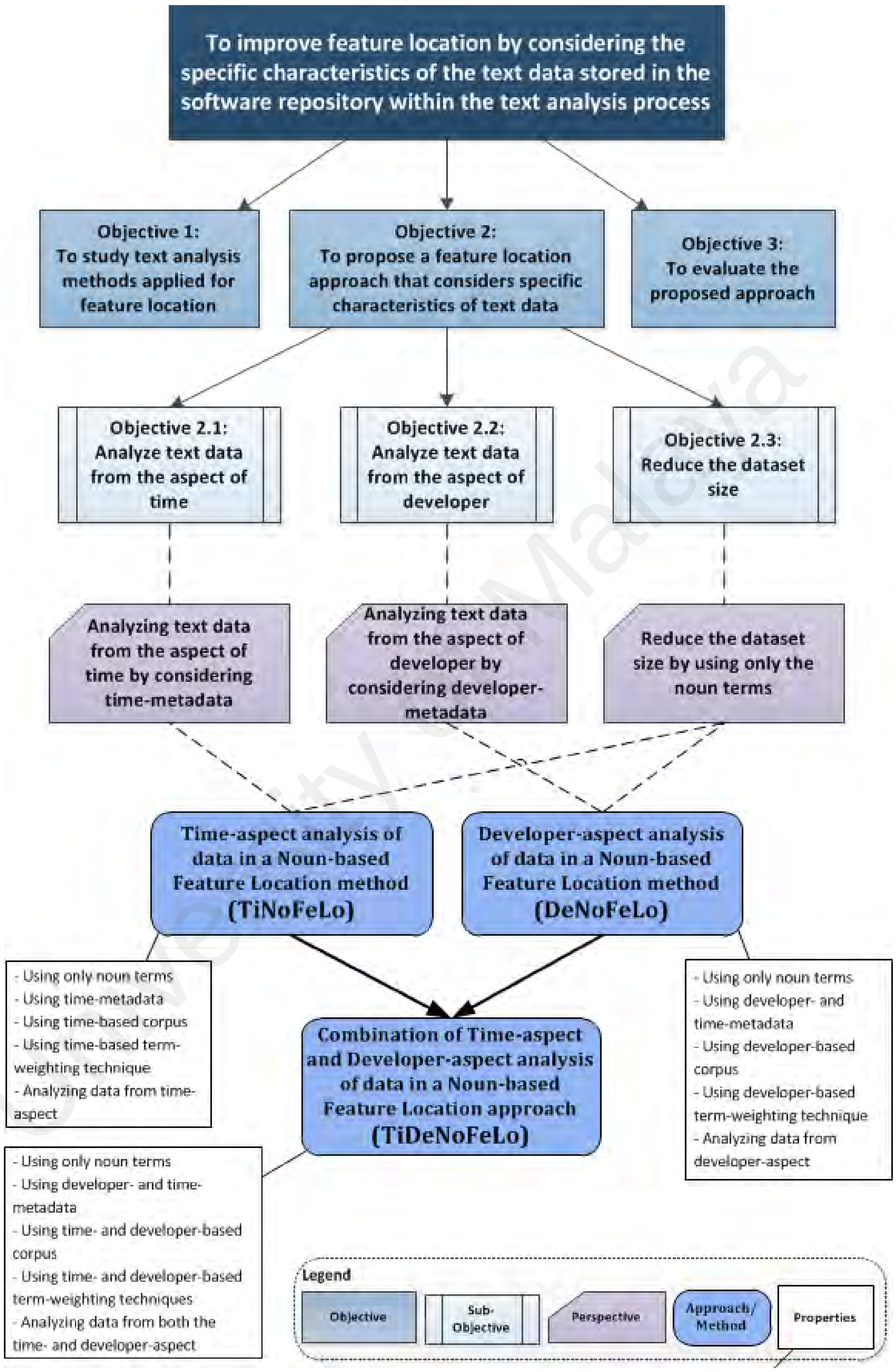
Investigation of the feature location literature indicates that the origination of the applied text analysis methods for feature location mainly comes from a natural language context (Bassett & Kraft, 2013). Thus, it is likely that text analysis may not utilize all possible potential for identifying the correct source code location pertinent to a software feature. Low accuracy of the text analysis is due to the lack of considering the specific characteristics of software repository's text data – i.e. incorporation of the data with metadata and large dataset size – in analyzing the historical text data. Addressing this issue in analyzing the repository's text data enhances the location identification process. Since text analysis process is foundational to the most of feature location approaches, the main goal of this thesis is improving feature location accuracy by considering the specific characteristics of the text data recorded in the software repository within the text analysis process. In the next section, a summary of the approach of addressing this goal is explained.

7.1 Summary of Research Work and Contributions

In order to address the main objective of this thesis, a set of objectives were identified which are summarized in Figure 7.1. Addressing these objectives results in satisfaction of the main goal. In the rest of this section, the achievement of the identified objectives, as well as the contributions of this thesis is briefly explained.

- **Objective 1**: The first objective of this research aims to study the existing text analysis methods applied for feature location and identify the current problems in the existing text analysis methods for feature location. To address this objective, the literature in feature location research area and the other related research areas was broadly investigated to gain an insightful understanding of typical text analysis methods and their weaknesses (See Chapter 2). The investigation of the literature indicates that the origination of current applied text analysis methods for feature location mostly comes from a natural language context. As mentioned earlier, incorporation of the data with metadata, i.e. time and developer, as well as the large dataset size are considered as two important characteristics that differentiate the repository's text data from the text data in a natural language context. However, the researchers' efforts to improve the text analysis process have been mainly focused on the use of additional information such as static and dynamic information with text analysis. It was found that the incorporation of the data with the metadata is not properly addressed in feature location literature. Furthermore, there is no simple and applicable way to reduce the dataset size that results in properly summarizing the data and reducing the noises for feature location.

- **Objective 2**: The second objective of this study aims at proposing a new feature location approach that considers the identified characteristics of repository's text data in order to make an improvement in text analysis process. These characteristics were considered in text analysis process in three perspectives, i.e. (i) analysis of the text data from the aspect of time at which the data was recorded in the



To improve feature location by considering the specific characteristics of the text data stored in the software repository within the text analysis process
Objective 1: To study text analysis methods applied for feature location
Objective 2: To propose a feature location approach that considers specific characteristics of text data
Objective 3: To evaluate the proposed approach
Objective 2.1: Analyze text data from the aspect of time
Objective 2.2: Analyze text data from the aspect of developer
Objective 2.3: Reduce the dataset size
Analyzing text data from the aspect of time by considering time-metadata
Analyzing text data from the aspect of developer by considering developer-metadata
Reduce the dataset size by using only the noun terms
Time-aspect analysis of data in a Noun-based Feature Location method (TiNoFeLo)
Developer-aspect analysis of data in a Noun-based Feature Location method (DeNoFeLo)
- Using only noun terms - Using time-metadata - Using time-based corpus - Using time-based term-weighting technique - Analyzing data from time-aspect
- Using only noun terms - Using developer- and time-metadata - Using developer-based corpus - Using developer-based term-weighting technique - Analyzing data from developer-aspect
Combination of Time-aspect and Developer-aspect analysis of data in a Noun-based Feature Location approach (TiDeNoFeLo)
- Using only noun terms - Using developer- and time-metadata - Using time- and developer-based corpus - Using time- and developer-based term-weighting techniques - Analyzing data from both the time- and developer-aspect
Legend
Objective
Sub-Objective
Perspective
Approach/ Method
Properties

Figure 7.1: Achievement of the main research objective through proposing a feature location approach

repository, (ii) analysis of the text data from the aspect of developers who recorded the data in the repository, (iii) reducing the dataset size by using only the noun terms that exist in the text data recorded in the repository. Corresponding to each of these perspectives, a set of sub-objectives were identified (See Figure 7.1). Applying the identified perspectives in text analysis process resulted in proposing two feature location methods that were finally combined to embody the proposed feature location approach. The proposed methods and approach are briefly explained in the following.

– The first proposed method identifies the source code files by analyzing the data from the aspect of time when the data was recorded in the repository. In addition, this method uses only the nouns extracted from the source code files as the dataset and weights the noun terms using a time-based term-weighting technique. This method is referred as ***Ti***me-aspect analysis of data in a Noun-based Feature Location method (***Ti***NoFeLo).

– The second proposed method analyzes the repository's data from the aspect of developer who recorded the data in the repository. Similar to ***Ti***NoFeLo, this method also uses only the nouns as the dataset and weights the noun terms using a developer-based term-weighting technique. This method is referred as ***De***veloper-aspect analysis of data in a Noun-based Feature Location method (***De***NoFeLo).

– The combination of ***Ti***me-aspect and ***De***veloper-aspect analysis of data is used to propose a Noun-based Feature Location approach (named ***TiDe***NoFeLo). The proposed approach addresses all the identified perspectives as a whole in order to enhance the text analysis based feature location process. Figure 7.1

displays the main objectives, sub-objectives and the identified perspectives in relation with the proposed methods and approach. This figure also summarizes the main properties of the proposed methods and the proposed approach.

- **Objective 3**: The third objective of this thesis investigates the impacts of considering the identified characteristics of the repository's data in the text analysis process. Accordingly, the proposed methods and the proposed approach were experimentally evaluated with respect to the guidelines recommended by Wohlin et al. (Wohlin et al., 2012) (See Chapter 5). In this case, the impacts of considering time and developer-metadata as well as using noun-only for feature location were assessed in a comprehensive experimental evaluation, and the obtained results were reported and analyzed in Chapter 6.

With respect to the research work that was accomplished to achieve the identified objectives, the contributions of this thesis are:

- **Introducing three types of feature location approaches, i.e. location based, time-based and developer-based**: In location-based feature location approach, the data is analyzed from the aspect of source code location where the data is stored. According to the feature location literature, almost all the existing feature location approaches are location-based. Time-based feature location approach analyzes the data from the aspect of time when the data was recorded, and developer-based feature location approach analyzes the data from the aspect of developer who recorded the data in the repository. Time and developer-based types of feature location approaches are new and they are proposed in this thesis.

- **Considering time-metadata in feature location process by proposing a time-based feature location method**: As mentioned in Chapter 1, time is an important piece of metadata which is incorporated with the repository's data. Consideration of time-metadata in feature location process provides the ability of analyzing the data from the aspect of time when the data is recorded in the repository. As mentioned above, the first proposed method in this thesis, ***Ti***NoFeLo, analyzes the data from the time-aspect. In this method, the data in the repository is classified based on the time of usage. This approach significantly improves the accuracy, performance and effectiveness of feature location process over the location-based baseline approaches.

- **Considering developer-metadata in feature location process by proposing a developer-based feature location method**: As mentioned in Chapter 1, developer-metadata is another important piece of metadata. Consideration of developer-metadata provides the ability of analyzing the data from the aspect of developer who recorded the data in the repository. The second proposed method in this thesis, ***De***NoFeLo, classified the data in the developer expertise profiles and valued the data based on the developer who recorded the data in the repository. ***De***NoFeLo outperforms the location-based and time-based feature location baseline approaches.

- **Using only the noun terms as the dataset for feature location**: In this thesis, the dataset, which is used to identify the related source code locations to a desired software feature, is restricted to only the noun terms extracted from the repository. The use of only the nouns leads to significantly reduction of dataset size and summarizing the data. As demonstrated in Chapter 6, the use of noun-only notably improves the accuracy, performance and effectiveness of feature location process.

- **Combining the consideration of time-metadata and developer-metadata, and the use of noun-only for feature location**: As mentioned above, consideration of each of time and developer-metadata and also use of noun-only separately improve the accuracy of feature location process. In the last part of this thesis, these contributions were combined to obtain a robust feature location approach, called ***TiDe***NoFeLo. ***TiDe***NoFeLo significantly improves the accuracy, performance and effectiveness of feature location process over the location-based, time-based, and developer-based baseline approaches. In ***TiDe***NoFeLo, the data is analyzed from both the time-aspect and developer-aspect, and also the dataset is limited to noun-only.

7.2 Limitations and Future Works

The experimental evaluation of the proposed methods, ***Ti***NoFeLo and ***De***NoFeLo, and proposed approach indicates a significant improvement in feature location process which is made by considering time and developer-metadata, and using only the nouns. However, the use of the proposed methods on different software systems reveals few limitations. As discussed in Chapter 6, the proposed time-based method, ***Ti***NoFeLo, is challenged on the project with a low evolution speed such as Rhino. This exception is due to the low volume of modified data over time that affects the accuracy of a time-based method. Therefore, the use of time-based methods is not recommended for software projects with a low evolution speed, such as Rhino.

On the other hand, the performance of the proposed developer-based method, ***De***NoFeLo, depends on the ratio of the project developers over the source code files. The experimental results reveal lower accuracy of the method on a project, such as AspectJ, with a low ratio of the developers over the source code files. In this case, the average size of classes in developer expertise profiles is large and the likelihood of similarity of a given software feature with a large size of profiles will be high. This results in the reduction of the accuracy of the feature location process.

However, all these limitations are addressed in the proposed approach, ***TiDe***NoFeLo, by combining ***Ti***NoFeLo and ***De***NoFeLo. This is due to the analysis of data from both the aspects of time and developer in the ***TiDe***NoFeLo approach. Based on the experimental results, it can be concluded that the combination of time and developer-aspect of data analysis in a noun-based feature location approach makes a significant improvement in the accuracy, performance and effectiveness of feature location process applicable on a large set of different software projects with various evolution speeds and various ratios of developers over source code files.

For the future work, a number of directions can be explored.

- First, investigating the other metadata incorporated with the repository's text data that has the potential to make an improvement on the accuracy of text analysis based feature location.
- Second, examining other possible ways of analyzing the data from the aspects of time and developer.
- Third, considering the existing feature location approaches to find possible ways of adapting them to apply time-metadata or developer-metadata in order to improve their accuracy.
- Fourth, combining the time-aspect and developer-aspect of data analysis with the dynamic and static analysis of data to further improve feature location accuracy.
- Fifth, considering the synonym of the noun terms recorded in source code files to improve the accuracy of feature location process.
- Sixth, investigating different smoothing methods (Zhai & Lafferty, 2004) to resolve the problem of zero value for the developer-based term-weighting technique.

7.3 Final Remarks

In this thesis, a new feature location approach, named ***TiDe***NoFeLo, is proposed that considers the metadata of time and developer, and uses only the nouns as the dataset. In this approach, the data is analyzed from the aspect of time when the data was recorded and the aspect of developer who recorded the data in the repository. Hence two new feature location methods were proposed applying each of these aspects of data analysis separately.

The first proposed method, a time-based method named ***Ti***NoFeLo, considers time-metadata in text analysis process and analyzes the repository's data from the aspect of time when the data was recorded in the repository. The second proposed method, a developer-based method called ***De***NoFeLo, considers the time and developer-metadata in text analysis process and analyzes the data from the aspect of developer who recorded the data in the repository. Both the methods use only the noun terms, extracted from the text resources, as the dataset.

The experimental results indicate that the proposed approach outperforms the location-based, time-based, and developer-based baseline approaches and consequently improves feature location accuracy. Since text analysis process is a fundamental step in most of the feature location approaches, improving the accuracy of text analysis process individually impacts the overall accuracy of feature location process that used text data separately or in combination with the additional information.

REFERENCES

Abadi, A., Nisenson, M., & Simionovici, Y. (2008). A traceability technique for specifications [Conference Proceedings]. In *The 16th ieee international conference on program comprehension (icpc 2008)* (pp. 103–112). IEEE Computer Society.

Abebe, S. L., & Tonella, P. (2010). Natural language parsing of program element names for concept extraction [Conference Proceedings]. In *Ieee 18th international conference on program comprehension (icpc 2010)* (pp. 156–159). IEEE.

Alhindawi, N., Dragan, N., Collard, M. L., & Maletic, J. (2013). Improving feature location by enhancing source code with stereotypes [Conference Proceedings]. In *The 29th ieee international conference on software maintenance (icsm 2013)* (pp. 300–309). IEEE.

Andrieu, C., De Freitas, N., Doucet, A., & Jordan, M. I. (2003). An introduction to mcmc for machine learning [Journal Article]. *Machine learning*, *50*(1-2), 5–43.

Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., & Merlo, E. (2002). Recovering traceability links between code and documentation [Journal Article]. *IEEE Transactions on Software Engineering*, *28*(10), 970–983.

Antoniol, G., & Gueheneuc, Y.-G. (2006). Feature identification: An epidemiological metaphor [Journal Article]. *IEEE Transactions on Software Engineering*, *32*(9), 627–641.

Anvik, J. (2006). Automating bug report assignment [Conference Proceedings]. In *Proceedings of the 28th international conference on software engineering (icse 2006)* (pp. 937–940).

Anvik, J., Hiew, L., & Murphy, G. C. (2006). Who should fix this bug? [Conference Proceedings]. In *Proceedings of the 28th international conference on software engineering (icse 2006)* (pp. 361–370). ACM.

Anvik, J., & Murphy, G. C. (2007). Determining implementation expertise from bug reports [Conference Proceedings]. In *The fourth international workshop on mining software repositories, 2007. icse workshops msr 2007.* (pp. 2–2). IEEE.

Bacchelli, A., Lanza, M., & Robbes, R. (2010). Linking e-mails and source code artifacts [Conference Proceedings]. In *Proceedings of the 32nd acm/ieee international* *conference on software engineering (icse 2010)* (Vol. 1, pp. 375–384).

Baeza-Yates, R. A., & Ribeiro-Neto, B. (1999). *Modern information retrieval* (Vol. 463) [Book]. New York: ACM press.

Bai, J., Nie, J.-Y., & Paradis, F. (2004). Using language models for text classification [Conference Proceedings]. In *Asia information retrieval symposium (airs)* (p. 6). Beijing, China.

Bassett, B., & Kraft, N. A. (2013). Structural information based term weighting in text retrieval for feature location [Conference Proceedings]. In *Ieee 21st international conference on program comprehension (icpc 2013)* (pp. 133–141).

Berry, M. W. (1992). Large-scale sparse singular value computations [Journal Article]. *International Journal of Supercomputer Applications*, *6*(1), 13–49.

Biggers, L. R., Bocovich, C., Capshaw, R., Eddy, B. P., Etzkorn, L. H., & Kraft, N. A. (2014). Configuring latent dirichlet allocation based feature location [Journal Article]. *Empirical Software Engineering*, *19*(3), 465–500.

Biggerstaff, T. J., Mitbander, B. G., & Webster, D. (1993). The concept assignment problem in program understanding [Conference Proceedings]. In *Proceedings of the 15th international conference on software engineering (icse 1993)* (pp. 482–498).

Bird, C., Pattison, D., D'Souza, R., Filkov, V., & Devanbu, P. (2008). Latent social structure in open source projects [Conference Proceedings]. In *Proceedings of the 16th acm sigsoft international symposium on foundations of software engineering* (pp. 24–35). ACM.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation [Journal Article]. *Journal of Machine Learning Research*, *3*, 993–1022.

Bohner, S. A. (1996). Software change impact analysis [Journal Article]. *IEEE Computer Society*, 196–205.

Bouras, C., & Tsogkas, V. (2010). Noun retrieval effect on text summarization and delivery of personalized news articles to the user's desktop. *Data & Knowledge Engineering*, *69*(7), 664–677. (Advanced Knowledge-based Systems) doi: http://dx.doi.org/10.1016/j.datak.2010.02.005

Briand, L. C., Labiche, Y., & Yue, T. (2009). Automated traceability analysis for uml model refinements [Journal Article]. *Information and Software Technology*, *51*(2), 512–527.

Brin, S., & Page, L. (2012). Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, *56*(18), 3825 - 3833. (The {WEB} we live in)

Butler, S., Wermelinger, M., Yu, Y., & Sharp, H. (2011). Improving the tokenisation of identifier names [Conference Proceedings]. In *Proceedings of the 25th european conference on object-oriented programming (ecoop 2011)* (pp. 130–154).

Capobianco, G., De Lucia, A., Oliveto, R., Panichella, A., & Panichella, S. (2009). On the role of the nouns in ir-based traceability recovery [Conference Proceedings]. In *Proceedings of the 17th international conference on program comprehension (icpc 2009)* (pp. 148–157). IEEE.

Capobianco, G., Lucia, A. D., Oliveto, R., Panichella, A., & Panichella, S. (2012). Improving ir-based traceability recovery via noun-based indexing of software artifacts [Journal Article]. *Journal of Software: Evolution and Process*, *25*(7), 743–762.

Chen, K., & Rajlich, V. (2000). Case study of feature location using dependence graph [Conference Proceedings]. In *Proceedings 8th international workshop on program comprehension (iwpc 2000)* (pp. 241–247). IEEE Computer Society Press.

Chen, K., & Rajlich, V. (2010). Case study of feature location using dependence graph, after 10 years [Conference Proceedings]. In *Proceedings of the 2010 ieee 18th international conference on program comprehension (icpc 2010)* (pp. 1–3). IEEE.

Chowdhury, G. (2010). *Introduction to modern information retrieval* [Book]. Facet Publishing.

Cleary, B., & Exton, C. (2007). assisting concept location in software comprehension [Journal Article]. *Psychology of Programming Interest Group (PPIG 2007)*, 42–55.

Cleary, B., Exton, C., Buckley, J., & English, M. (2009). An empirical analysis of information retrieval based concept location techniques in software comprehension [Journal Article]. *Empirical Software Engineering*, *14*(1), 93–130.

Cohen, J. (1992). A power primer [Journal Article]. *Psychological bulletin*, *112*(1), 155.

Corley, C. S., Kraft, N. A., Etzkorn, L. H., & Lukins, S. K. (2011). Recovering traceability links between source code and fixed bugs via patch analysis [Conference Proceedings]. In *Proceedings of the 6th international workshop on traceability in emerging forms of software engineering* (pp. 31–37).

Cornelissen, B., Zaidman, A., Van Deursen, A., Moonen, L., & Koschke, R. (2009). A systematic survey of program comprehension through dynamic analysis [Journal Article]. *IEEE Transactions on Software Engineering*, *35*(5), 684–702.

Crain, S. P., Zhou, K., Yang, S.-H., & Zha, H. (2012). Dimensionality reduction and topic modeling: from latent semantic indexing to latent dirichlet allocation and beyond [Journal Article]. *Mining Text Data*, 129–161.

Cunningham, H., Maynard, D., Bontcheva, K., & Tablan, V. (2002). Gate: an architecture for development of robust hlt applications. In *Proceedings of the 40th annual meeting on association for computational linguistics* (pp. 168–175).

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis [Journal Article]. *Journal of the American society for information science*, *41*(6), 391–407.

De Lucia, A., Oliveto, R., & Tortora, G. (2009). Assessing ir-based traceability recovery tools through controlled experiments [Journal Article]. *Empirical Software Engineering*, *14*(1), 57–92.

Diaz, D., Bavota, G., Marcus, A., Oliveto, R., Takahashi, S., & De Lucia, A. (2013). Using code ownership to improve ir-based traceability link recovery [Conference Proceedings]. In *Proceedings of the 21th international conference on program comprehension (icpc 2013)* (pp. 123–132). IEEE.

Dit, B., Moritz, E., & Poshyvanyk, D. (2012). A tracelab-based solution for creating, conducting, and sharing feature location experiments [Conference Proceedings]. In *Proceedings of the 20th international conference on program comprehension (icpc 2012)* (pp. 203–208). IEEE.

Dit, B., Revelle, M., Gethers, M., & Poshyvanyk, D. (2013). Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process*, *25*(1), 53–95. Retrieved from `http://dx.doi.org/10.1002/smr.567` doi: 10.1002/smr.567

Eddy, B. P., & Kraft, N. A. (2014). Using structured queries for source code search [Conference Proceedings]. In *International conference on software maintenance and evolution (icsme 2014)* (pp. 431–435). IEEE.

Eisenberg, A. D., & De Volder, K. (2005). Dynamic feature traces: Finding features in unfamiliar code [Conference Proceedings]. In *Proceedings of the 21st ieee international conference on software maintenance (icsm 2005)* (p. 337-346).

Gay, G., Haiduc, S., Marcus, A., & Menzies, T. (2009). On the use of relevance feedback in ir-based concept location [Conference Proceedings]. In *Proceedings of the ieee international conference on software maintenance (icsm 2009)* (p. 351-360). IEEE.

Gethers, M., Kagdi, H., Dit, B., & Poshyvanyk, D. (2011). An adaptive approach to impact analysis from change requests to source code [Conference Proceedings]. In *Proceedings of the 26th ieee/acm international conference on automated software engineering* (p. 540-543). IEEE Computer Society.

Gómez, V. U., Kellens, A., Brichau, J., & D'Hondt, T. (2009). Time warp, an approach for reasoning over system histories. In *Proceedings of the joint international and annual ercim workshops on principles of software evolution (iwpse) and software evolution (evol) workshops* (pp. 79–88). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1595808.1595825` doi: 10.1145/1595808.1595825

Haiduc, S., Aponte, J., & Marcus, A. (2010). Supporting program comprehension with source code summarization [Conference Proceedings]. In *Proceedings of the acm/ieee 32th international conference on software engineering (icse 2010)* (Vol. 2, p. 223-226).

Hassan, A. E. (2006, Sept). Mining software repositories to assist developers and support managers. In *Proceedings of the 22nd ieee international conference on software maintenance (icsm 2006)* (p. 339-342). doi: 10.1109/ICSM.2006.38

Hassan, A. E. (2008). The road ahead for mining software repositories [Conference Proceedings]. In *Frontiers of software maintenance (fosm 2008)* (p. 48-57).

Hassan, A. E., & Holt, R. C. (2005). The top ten list: Dynamic fault prediction [Conference Proceedings]. In *Proceedings of the 21st ieee international conference on software maintenance (icsm 2005)* (p. 263-272).

Hassan, A. E., & Xie, T. (2010). Mining software engineering data [Conference Proceedings]. In *Proceedings of the 32nd acm/ieee international conference on software engineering (icse 2010)* (Vol. 2, p. 503-504). ACM.

Heinrich, G. (2005). *Parameter estimation for text analysis* (Report). University of Leipzig.

Hill, E., Pollock, L., & Vijay-Shanker, K. (2007). Exploring the neighborhood with dora to expedite software maintenance [Conference Proceedings]. In *Proceedings of the twenty-second ieee/acm international conference on automated software engineering* (p. 14-23).

Hill, E., Pollock, L., & Vijay-Shanker, K. (2009). Automatically capturing source code context of nl-queries for software maintenance and reuse [Conference Proceedings]. In *Proceedings of the 31st international conference on software engineering (icse 2009)* (p. 232-242).

Hossen, K., Kagdi, H. H., & Poshyvanyk, D. (2014). Amalgamating source code authors, maintainers, and change proneness to triage change requests [Conference Proceedings]. In *Proceedings of the 22nd international conference on program comprehension (icpc 2014)* (p. 130-141).

Jacobs, R. A. (1995, September). Methods for combining experts' probability assessments. *Neural Comput.*, *7*(5), 867–888.

Jurafsky, D., & Martin, J. H. (2014). *Speech and language processing* [Book]. Pearson.

Kagdi, H., Collard, M. L., & Maletic, J. I. (2007). A survey and taxonomy of approaches for mining software repositories in the context of software evolution [Journal Article]. *Journal of Software Maintenance and Evolution: Research and Practice*, *19*(2), 77-131.

Kagdi, H., Gethers, M., Poshyvanyk, D., & Hammad, M. (2012). Assigning change requests to software developers [Journal Article]. *Journal of Software: Evolution and Process*, *24*(1), 3-33.

Kagdi, H., Maletic, J. I., & Sharif, B. (2007). Mining software repositories for traceability links [Conference Proceedings]. In *15th ieee international conference on program comprehension (icpc 2007)* (p. 145-154).

Karahasanović, A., Levine, A. K., & Thomas, R. (2007). Comprehension strategies and difficulties in maintaining object-oriented systems: An explorative study. *Journal of Systems and Software*, *80*(9), 1541–1559.

Kim, S., Whitehead, E. J., Jr., & Zhang, Y. (2008, March). Classifying software changes: clean or buggy? *IEEE Transactions on Software Engineering*, *34*(2), 181–196. Retrieved from `http://dx.doi.org/10.1109/TSE.2007.70773` doi: 10.1109/TSE.2007.70773

Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment [Journal Article]. *Journal of the ACM (JACM)*, *46*(5), 604-632.

Landauer, T. K., & Dumais, S. T. (1997). A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge [Journal Article]. *Psychological review*, *104*(2), 211.

Le, T.-D. B., Oentaryo, R. J., & Lo, D. (2015). Information retrieval and spectrum based bug localization: Better together. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering* (pp. 579–590).

Linares-Vasquez, M., Hossen, K., Dang, H., Kagdi, H., Gethers, M., & Poshyvanyk, D. (2012). Triaging incoming change requests: Bug or commit history, or code authorship? *Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM 2012)*, *0*, 451-460. doi: http://doi.ieeecomputersociety.org/10.1109/ICSM.2012.6405306

Liu, D., Marcus, A., Poshyvanyk, D., & Rajlich, V. (2007). Feature location via information retrieval based filtering of a single scenario execution trace [Conference Proceedings]. In *Proceedings of the twenty-second ieee/acm international conference on automated software engineering* (p. 234-243).

Lukins, S. K., Kraft, N. A., & Etzkorn, L. H. (2008). Source code retrieval for bug localization using latent dirichlet allocation [Conference Proceedings]. In *15th working conference on reverse engineering (wcre 2008)* (p. 155-164).

Lukins, S. K., Kraft, N. A., & Etzkorn, L. H. (2010). Bug localization using latent dirichlet allocation [Journal Article]. *Information and Software Technology*, *52*(9), 972-990.

Mader, P., & Gotel, O. (2012). Towards automated traceability maintenance [Journal Article]. *Journal of Systems and Software*, *85*(10), 2205-2227.

Mader, P., Gotel, O., & Philippow, I. (2008). Enabling automated traceability maintenance by recognizing development activities applied to models [Conference Proceedings]. In *Proceedings of the 2008 23rd ieee/acm international conference on automated software engineering* (p. 49-58).

Manning, C. D., Raghavan, P., & Schutze, H. (2008). *Introduction to information retrieval* (Vol. 1) [Book]. Cambridge University Press Cambridge.

Marcus, A., & Haiduc, S. (2013). Text retrieval approaches for concept location in source code [Book Section]. In *Software engineering* (pp. 126–158). Springer.

Marcus, A., Maletic, J. I., & Sergeyev, A. (2005). Recovery of traceability links between software documentation and source code [Journal Article]. *International Journal of Software Engineering and Knowledge Engineering*, *15*(5), 811-836.

Marcus, A., Sergeyev, A., Rajlich, V., & Maletic, J. I. (2004). An information retrieval approach to concept location in source code [Conference Proceedings]. In *Proceedings of 11th working conference on reverse engineering (wcre 2004)* (p. 214-223).

McCandless, M., Hatcher, E., & Gospodnetic, O. (2010). *Lucene in action, second edition: covers apache lucene 3.0*. Greenwich, CT, USA: Manning Publications Co.

Moreno, L., Treadway, J. J., Marcus, A., & Shen, W. (2014). On the use of stack traces to improve text retrieval-based bug localization [Conference Proceedings]. In *Proceedings of the ieee international conference on software maintenance and evolution (icsme 2014)* (p. 151-160). IEEE.

Ngomo, A.-C. (2010). Low-bias extraction of domain-specific concepts [Journal Article]. *Informatica*, *34*(1), 37-44.

Ngomo, A.-C. N., & Schumacher, F. (2009). Borderflow: A local graph clustering algorithm for natural language processing [Book Section]. In *Computational linguistics and intelligent text processing* (p. 547-558). Springer.

Noice, H. (2013). *The nature of expertise in professional acting: A cognitive view*. Psychology Press.

Omoronyia, I., Sindre, G., & Stå, T., lhane. (2011). Exploring a bayesian and linear approach to requirements traceability [Journal Article]. *Information and Software Technology*, *53*(8), 851-871.

Petrenko, M., Rajlich, V., & Vanciu, R. (2008). Partial domain comprehension in software evolution and maintenance [Conference Proceedings]. In *Proceedings of the 16th ieee international conference on program comprehension (icpc 2008)* (p. 13-22).

Poshyvanyk, D. (2008). *Using information retrieval to support software maintenance tasks* (Thesis). Wayne State University.

Poshyvanyk, D., Gethers, M., & Marcus, A. (2012). Concept location using formal concept analysis and information retrieval [Journal Article]. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *21*(4), 23.

Poshyvanyk, D., Gueheneuc, Y.-G., Marcus, A., Antoniol, G., & Rajlich, V. (2006). Combining probabilistic ranking and latent semantic indexing for feature identification [Conference Proceedings]. In *Proceedings of the 14th ieee international conference on program comprehension (icpc 2006)* (p. 137-148).

Poshyvanyk, D., Gueheneuc, Y.-G., Marcus, A., Antoniol, G., & Rajlich, V. (2007). Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval [Journal Article]. *IEEE Transactions on Software Engineering*, *33*(6), 420-432.

Poshyvanyk, D., & Marcus, A. (2007). Combining formal concept analysis with information retrieval for concept location in source code [Conference Proceedings]. In *Proceedings of the 15th ieee international conference on program comprehension (icpc 2007)* (p. 37-48).

Rajlich, V., & Wilde, N. (2002). The role of concepts in program comprehension [Conference Proceedings]. In *Proceedings of the 10th international workshop on program comprehension* (p. 271-278).

Rao, S., & Kak, A. (2011). Retrieval from software libraries for bug localization: a comparative study of generic and composite text models [Conference Proceedings]. In *Proceeding of the 8th working conference on mining software repositories (msr 2011)* (p. 43-52).

Ratanotayanon, S., Choi, H. J., & Sim, S. E. (2010). Using transitive changesets to support feature location [Conference Proceedings]. In *Proceedings of the ieee/acm international conference on automated software engineering* (p. 341-344).

Ratiu, D., & Deissenboeck, F. (2007). From reality to programs and (not quite) back again [Conference Proceedings]. In *Proceeding of the 15th ieee international conference on program comprehension (icpc 2007)* (p. 91-102).

Revelle, M., Dit, B., & Poshyvanyk, D. (2010). Using data fusion and web mining to support feature location in software [Conference Proceedings]. In *Proceedings of the 18th international conference on program comprehension (icpc 2010)* (pp. 14–23). IEEE.

Rilling, J., Witte, R., Gasevic, D., & Pan, J. Z. (2008). Semantic technologies in system maintenance (stsm 2008) [Conference Proceedings]. In *Proceedings of the 2008 the 16th ieee international conference on program comprehension* (p. 279-282). IEEE Computer Society.

Robillard, M. P. (2008, August). Topologyanalysis of software dependencies. *ACM Trans. Softw. Eng. Methodol.*, *17*(4), 1–36. Retrieved from `http://doi.acm.org/10.1145/13487689.13487691` doi: 10.1145/13487689.13487691

Romano, J., Kromrey, J. D., Coraggio, J., & Skowronek, J. (2006). Appropriate statistics for ordinal level data: Should we really be using t-test and cohend́ for evaluating group differences on the nsse and other surveys [Conference Proceedings]. In *Annual meeting of the florida association of institutional research* (p. 1-33).

Saha, R. K., Lease, M., Khurshid, S., & Perry, D. E. (2013). Improving bug localization using structured information retrieval [Conference Proceedings]. In *Proceedings of the 28th international conference on automated software engineering (ase'13)* (p. 345-355). IEEE/ACM.

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval [Journal Article]. *Information processing & management*, *24*(5), 513-523.

Salton, G., & McGill, M. J. (1986). *Introduction to modern information retrieval*. New York, NY, USA: McGraw-Hill, Inc.

Salton, G., Wong, A., & Yang, C.-S. (1975). A vector space model for automatic indexing [Journal Article]. *Communications of the ACM*, *18*(11), 613-620.

Salton, G., & Yang, C.-S. (1973). On the specification of term values in automatic indexing [Journal Article]. *Journal of documentation*, *29*(4), 351-372.

Sarawagi, S. (2008). Information extraction [Journal Article]. *Foundations and trends in databases*, *1*(3), 261-377.

Scanniello, G., & Marcus, A. (2011). Clustering support for static concept location in source code [Conference Proceedings]. In *Proceeding of the ieee 19th international conference on program comprehension (icpc 2011)* (p. 1-10).

Scanniello, G., Marcus, A., & Pascale, D. (2015). Link analysis algorithms for static concept location: an empirical assessment. *Empirical Software Engineering*, *20*(6), 1666–1720.

Schuler, D., & Zimmermann, T. (2008). Mining usage expertise from version archives [Conference Proceedings]. In *Proceedings of the 2008 international working conference on mining software repositories* (p. 121-124).

Servant, F., & Jones, J. A. (2012). Whosefault: Automatic developer-to-fault assignment through fault localization [Conference Proceedings]. In *proceeding of the 34th international conference on software engineering (icse 2012)* (p. 36-46).

Shepherd, D., Fry, Z. P., Hill, E., Pollock, L., & Vijay-Shanker, K. (2007). Using natural language program analysis to locate and understand action-oriented concerns [Conference Proceedings]. In *Proceedings of the 6th international conference on aspect-oriented software development* (p. 212-224).

Shepherd, D., Pollock, L., & Vijay-Shanker, K. (2006). Towards supporting on-demand virtual remodularization using program graphs [Conference Proceedings]. In *Proceedings of the 5th international conference on aspect-oriented software development* (p. 3-14). ACM.

Shokripour, R., Anvik, J., Kasirun, Z. M., & Zamani, S. (2013). Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation [Conference Proceedings]. In *Proceedings of the tenth international workshop on mining software repositories* (p. 2-11).

Shokripour, R., Anvik, J., Kasirun, Z. M., & Zamani, S. (2014). Improving automatic bug assignment using time-metadata in term-weighting [Journal Article]. *Institution of Engineering and Technology, IET*, *8*(6), 269-278.

Sillito, J., Murphy, G. C., & De Volder, K. (2008). Asking and answering questions during a programming change task [Journal Article]. *IEEE Transactions on Software Engineering*, *34*(4), 434-451.

Sisman, B., & Kak, A. C. (2012). Incorporating version histories in information retrieval based bug localization [Conference Proceedings]. In *Proceedings of the 9th ieee working conference on mining software repositories (msr 2012)* (p. 50-59).

Strohman, T., Metzler, D., Turtle, H., & Croft, W. B. (2005). Indri: A language model-based search engine for complex queries [Conference Proceedings]. In *Proceedings of the international conference on intelligent analysis* (Vol. 2, p. 2-6).

Tao, Q., Lu, Z., Zhiying, Z., Dan, H., & Jiasu, S. (2003). Discovering use cases from source code using the branch-reserving call graph [Conference Proceedings]. In *Tenth asia-pacific software engineering conference* (p. 60-67). IEEE Computer Society. doi: 10.1109/APSEC.2003.1254358

Voinea, L., & Telea, A. (2006). An open framework for cvs repository querying, analysis and visualization [Conference Proceedings]. In *Proceedings of the 2006 international workshop on mining software repositories* (p. 33-39).

Wang, J., Peng, X., Xing, Z., & Zhao, W. (2013). Improving feature location practice with multi-faceted interactive exploration. In *Proceedings of the 2013 international conference on software engineering* (p. 762-771).

Wang, S., & Lo, D. (2014). Version history, similar report, and structure: Putting them together for improved bug localization. In *Proceedings of the 22nd international conference on program comprehension* (pp. 53–63).

Wang, S., Lo, D., & Lawall, J. (2014). Compositional vector space models for improved bug localization. In *Proceedings of the 2014 ieee international conference on software maintenance and evolution (icsme 2014)* (pp. 171–180).

Wang, S., Lo, D., Xing, Z., & Jiang, L. (2011). Concern localization using information retrieval: An empirical study on linux kernel [Conference Proceedings]. In *Proceedings of the 2011 18th working conference on reverse engineering* (p. 92-96). IEEE Computer Society.

Wilde, N., & Scully, M. C. (1995). Software reconnaissance: mapping program features to code [Journal Article]. *Journal of Software Maintenance: Research and Practice*, *7*(1), 49-62.

Wille, R. (2005). Formal concept analysis as mathematical theory of concepts and concept hierarchies [Book Section]. In *Formal concept analysis* (p. 1-33). Springer.

Wilson, L. A. (2010). Using ontology fragments in concept location. In *Proceedings of the 2010 ieee international conference on software maintenance* (pp. 1–2). Washington, DC, USA: IEEE Computer Society. doi: 10.1109/ICSM.2010.5609555

Winkler, S., & von Pilgrim, J. (2010). A survey of traceability in requirements engineering and model-driven development [Journal Article]. *Software and Systems Modeling*, *9*(4), 529-565.

Witte, R., Li, Q., Zhang, Y., & Rilling, J. (2007). Ontological text mining of software documents. In *Proceedings of the 12th international conference on applications of natural language to information systems* (pp. 168–180). Berlin, Heidelberg: Springer-Verlag.

Witte, R., Li, Q., Zhang, Y., & Rilling, J. (2008). Text mining and software engineering: an integrated source code and document analysis approach [Journal Article]. *IET software*, *2*(1), 3 -16.

Witten, I. H., & Bell, T. C. (1991). The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression [Journal Article]. *IEEE Transactions on Information Theory*, *37*(4), 1085-1094.

Wohlin, C., Runeson, P., Hst, M., Ohlsson, M. C., Regnell, B., & Wessln, A. (2012). *Experimentation in software engineering* [Book]. Springer Publishing Company, Incorporated.

Yang, Y., & Pedersen, J. O. (1997). Acomparative study on feature selection in text categorization [Conference Proceedings]. In *Proceedings of the fourteenth international conference on machine learning* (pp. 412–420). Morgan Kaufmann Publishers Inc.

Ye, X., Bunescu, R., & Liu, C. (2014). Learning to rank relevant files for bug reports using domain knowledge. In *Proceedings of the 22nd acm sigsoft international symposium on foundations of software engineering* (pp. 689–699).

Zhai, C. (2008). Statistical language models for information retrieval [Journal Article]. *Synthesis Lectures on Human Language Technologies*, *1*(1), 1-141.

Zhai, C., & Lafferty, J. (2004). A study of smoothing methods for language models applied to information retrieval [Journal Article]. *Proceedings of the ACM Transactions on Information Systems (TOIS)*, *22*(2), 179-214.

Zhang, T., & Lee, B. (2013). A hybrid bug triage algorithm for developer recommendation. In *Proceedings of the 28th annual acm symposium on applied computing* (pp. 1088–1094). New York, NY, USA: ACM. doi: 10.1145/2480362.2480568

Zhang, Y., Witte, R., Rilling, J., & Haarslev, V. (2006). An ontology-based approach for traceability recovery [Conference Proceedings]. In *Proceedings of the 3rd international workshop on metamodels, schemas, grammars, and ontologies for reverse engineering (atem 2006)* (p. 36-43).

Zhao, W., Zhang, L., Liu, Y., Sun, J., & Yang, F. (2006). Sniafl: Towards a static noninteractive approach to feature location [Journal Article]. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *15*(2), 195-226.

Zhou, J., Zhang, H., & Lo, D. (2012). Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports [Conference Proceedings]. In *Proceedings of the 34th international conference on software engineering (icse 2012)* (p. 14-24). IEEE.

Zimmermann, T., Weisgerber, P., Diehl, S., & Zeller, A. (2005). Mining version histories to guide software changes [Journal Article]. *IEEE Transactions on Software Engineering*, *31*(6), 429-445.

Zou, X., Settimi, R., & Cleland-Huang, J. (2006). Phrasing in dynamic requirements trace retrieval [Conference Proceedings]. In *Proceedings of the 30th annual international computer software and applications conference (compsac 2006)* (Vol. 1, p. 265-272). IEEE Computer Society.

Zou, X., Settimi, R., & Cleland-Huang, J. (2008). Evaluating the use of project glossaries in automated trace retrieval [Conference Proceedings]. In *Software engineering research and practice* (p. 157-163).