A noun-based approach to feature location using time-aware term-weighting

Sima Zamani a,⁎, Sai Peck Lee a, Ramin Shokripour a, John Anvik b

a Faculty of Computer Science & Information Technology, University of Malaya, Kuala Lumpur, Malaysia
b Department of Computer Science, Central Washington University, Ellensburg, WA, USA

Abstract

Context: Feature location aims to identify the source code location corresponding to the implementation of a software feature. Many existing feature location methods apply text retrieval to determine the relevancy of the features to the text data extracted from the software repositories. One of the preprocessing activities in text retrieval is term-weighting, which is used to adjust the importance of a term within a document or corpus. Common term-weighting techniques may not be optimal to deal with text data from software repositories due to the origin of term-weighting techniques from a natural language context.

Objective: This paper describes how the consideration of when the terms were used in the repositories, under the condition of weighting only the noun terms, can improve a feature location approach.

Method: We propose a feature location approach using a new term-weighting technique that takes into account how recently a term has been used in the repositories. In this approach, only the noun terms are weighted to reduce the dataset volume and avoid dealing with dimensionality reduction.

Results: An empirical evaluation of the approach on four open-source projects reveals improvements to the accuracy, effectiveness and performance up to 50%, 17%, and 13%, respectively, when compared to the commonly-used Vector Space Model approach. The comparison of the proposed term-weighting technique with the Term Frequency-Inverse Document Frequency technique shows accuracy, effectiveness, and performance improvements as much as 15%, 10%, and 40%, respectively. The investigation of using only noun terms, instead of using all terms, in the proposed approach also indicates improvements up to 28%, 21%, and 58% on accuracy, effectiveness, and performance, respectively.

Conclusion: In general, the use of time in the weighting of terms, along with the use of only the noun terms, makes significant improvements to a feature location approach that relies on textual information.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

During software evolution, the existing source code of a project undergoes incremental modifications in order to satisfy software change requests [1–3]. A change request may result in adding a new software feature, removing a bug or defect, or improving existing software functionality [4]. More effective support for change requests is needed to obtain a sustainable, high-quality evolution of a software system. One of the key issues in addressing a change request is finding relevant locations in the source code of the project, such as files, classes, or methods, requiring modification to address the change request [2,5]. Performing this process manually in a large-scale software project is challenging and time-consuming.

Feature location [6,7] is a well-known technique used by software developers to address this challenge and it has become one of the most frequent program comprehension activities. Feature location aims to identify the initial location in the source code that is pertinent to a change request. It is also part of other software evolution tasks, such as the recovery of traceability links between software repositories, and the retrieval of software components for reuse [8].

A recent survey of feature location literature [9] found that more than 51% of the published literature in this research area is based, at least in part, on text retrieval. The primary text retrieval methods reported in this literature are: Pattern Matching (PM) [10,11], Information Retrieval (IR) [12,13] and Natural Language Processing (NLP) [14,15]. The use of these methods 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 [16]. However, these
methods originate from a natural language context such as the summarization of newspaper articles which are less structured than the text documents found in software repositories [17]. Furthermore, unlike the typical context in which these methods are applied, text documents in software repositories have a corresponding set of metadata [18]. In other words, the analysis of the textual data found in software repositories requires different techniques and methods than for those found in other text analysis domains.

As previously mentioned, text documents in software repositories are associated with a set of metadata. This metadata includes such items as developer identifiers, time stamps, and commit comments. This metadata associates answers of who, when, and why with the data in a software repository [19]. One important piece of metadata is the ‘when’, or the time at which the data was created or modified. According to the feature location literature, metadata has only been used by Sisman and Kak [20] for the weighting of files in a probabilistic IR model. As demonstrated in their work, the use of time-metadata can enhance feature location approaches and improve their results. In other words, the linking of the text data in a software repository with the associated time-metadata indicates that the importance of text varies over the different periods of the project’s life. In this paper, we propose a new feature location approach that includes weighting and ranking the source code locations based on both the textual similarity with a change request, and the use of time-metadata. The consideration of time-metadata in feature location is based on the following assumption. For a new change request, the source code entities with the highest textual similarity, and have also been most recently modified, lead to the most relevant source code locations. This assumption is based on two principles:

- **Defect localization:** It is known that the most recent modifications to a project are most likely the cause of future bugs or defects [21,22]. By considering recent modifications, this may lead to finding relevant locations that are the cause of a new change request [20].
- **Software evolution:** Each software project has different goals and requirements in different periods of the project’s life [23]. For a given change request, the requested modification to the source code in the same time period of project’s life will likely have the same goals or requirements. This principle bears further elaboration.

In every period of a project’s life cycle, the terms that are used across the different repositories, such as source code version and issue tracking, are consistent with the requirements of the project during that specific time period [23]. 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 request 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 JDT¹ project, around 490 files were modified in 490 commits to the project’s source code repository. Similarly, in the first day of the AspectJ² project, 87 files were modified in 2873 commits. The number of modified files and the extent of modifications gradually decreases over time and becomes stable. The time needed to reach this stability depends on the age of the project. Therefore, the time stamp of when a term was used in the project can play a significant role in determining the degree of relevancy of the term with a change request. This observation has motivated a new term-weighting technique that considers the value of the text over time.

In addition, traceability research conducted by Capobianco et al. [24,25] showed that using only the noun terms of the text greatly improves the accuracy of IR-based traceability recovery methods. Inspired by this result, the use of only noun terms in a bug assignment process was investigated and found to improve the accuracy of the developer recommendation [26].

This paper presents a feature location approach that uses only noun terms, called Noun-Based Feature Location (NBFL), and a time-aware weighting technique, called Time-Aware Term-Weighting (TATW). Text data is extracted from two software repositories of project, namely, Version Control System (VCS) and Issue Tracking System (ITS). The VCS is a software repository that manages the changes of source code and its relevant documents and the ITS is a software repository that tracks reported software change requests.

This paper also presents an empirical evaluation of the proposed approach. The evaluation was conducted on four open-source projects of varying sizes and development speeds. The result of using NBFL is compared to that of using the Vector Space Model (VSM) with the Term Frequency-Inverse Document Frequency (TF-IDF) technique [27]. VSM is a high-performing IR model in feature location that is used with common term-weighting techniques, and TF-IDF was chosen to assess the proposed term-weighting technique. The evaluation setup is similar to that used in the work of Rao and Kak [28] and Zhou et al. [29].

The evaluation showed a number of results. First, there are benefits to using only the noun terms from the text data. These benefits include:

- An independence of the approach from dimensionality reduction methods, which is one of the challenges with IR methods [30].
- A reduced amount of noise in the extracted entities from the data-sources [31], thereby enhancing the effectiveness of improvements made by other means.
- The use of only nouns provides enough information to make a feature location decision for a given change request (Section 4).

Second, the evaluation of the approach found that the NBFL approach using TATW outperformed the VSM approach in accuracy, effectiveness, and performance by as much as 50%, 17% and 13%, respectively. The evaluation also found that the TATW technique outperformed TF-IDF in accuracy, effectiveness, and performance by as much as 15%, 10% and 40%, respectively. In addition, an evaluation of the impact of using only noun terms in the proposed approach indicated an improvement on the accuracy, effectiveness, and performance of a feature location approach by up to 28%, 21%, and 58%, respectively. In summary, these results show that the use of time-metadata in a noun-based feature location approach is an improvement over the standard feature location approach using VSM and TF-IDF.

The remaining sections of this paper are organized as follows. In Section 2, the proposed feature location approach (NBFL) is described in detail. The setup for the empirical evaluation is presented in Section 3 and the evaluation results are given and discussed in Section 4. In Section 5, some of the threats to the validity of this study are outlined. An overview of the related research in feature location using text resources is presented in

---

¹ [http://www.eclipse.org/jdt/]
² [http://www.eclipse.org/aspectj/]
Section 6. Finally, the paper is concluded and future directions are discussed in Section 7.

2. The proposed approach

In this paper, a feature location approach with a new term-weighting technique is proposed. First, an overview of the proposed approach is given in Section 2.1. Next, the details of the proposed approach are presented in Section 2.2.

2.1. Overview of the proposed approach

In the proposed approach, information from two software repositories are used to determine the correct location for a new change request [20, 29]. The VCS is used to provide information about the source code locations and the ITS is used to provide information about previously fixed change requests [32]. Fig. 1 shows a high-level view of the proposed approach. As shown in the figure, the proposed approach is organized into three components:

1. Data Collecting and Integrating.
2. Weighting and Ranking.
3. Final Location Ranking.

In each of these three components, data is collected from both the VCS and the ITS, and is analyzed to identify the correct location for the new change request. In the Data Collecting and Integrating component, the collected data from the software repositories is preprocessed and analyzed for integration. The output of this component is the selected noun terms from both repositories that are linked to the corresponding source code locations. In the Weighting and Ranking component, the integrated data is used to weight and rank the source code locations. In the Final Location Ranking component, the ranked locations obtained from each of the data-sources are combined to create the final list of source code locations that have the highest relevancy for the new change request.

As the data from these two resources is in different formats, each requires different data extraction and analysis techniques in each component. In other words, each data-source follows its own path through the proposed approach. In the first path, referred
Change Request Analyzing path, separately. The locations are noun terms in the Source Code Analyzing path, and the code locations is computed as the sum of the weights of the terms that were created or modified. Next, the score of the source frequency and value of noun terms, including the time at which resources are weighted by applying the proposed term-weighting technique (Section 2.2.2). The term-weighting technique uses the relationships between the data-sources. The common noun terms in these highlighted. As shown in Figs. 2 and 3, two fields from the change request and the two entities from the software repositories are highlighted. As shown in Figs. 2 and 3, two fields from the change requests – Summary and Description – are used to find the similarities between the data-sources. The common noun terms in these resources are weighted by applying the proposed term-weighting technique (Section 2.2.2). The term-weighting technique uses the frequency and value of noun terms, including the time at which the terms were created or modified. Next, the score of the source code locations is computed as the sum of the weights of the common noun terms in the Source Code Analyzing path, and the Change Request Analyzing path, separately. The locations are ranked in descending order of relevancy in each path. Finally, the ranked locations in these two lists of locations are combined for create a final ranking to identify the most relevant source code location for the new change request.

2.2. Details of the proposed approach

This section provides a more detailed description of the components used in the approach. Fig. 5 shows an expanded view of Fig. 1 providing a detailed view of the approach and its components. It can be seen that each component has a set of subcomponents that address the goals of the specific component. In the rest of this section, these components and their subcomponents are further explained.

2.2.1. Data collecting and integrating component

The Data Collecting and Integrating component integrates the collected data and metadata from the software repositories based on the identified source code locations. It then provides the integrated data in a format that is used by the next component. This component uses a set of subcomponents called Data Collecting, Data Preprocessing, and Software Repositories Integration. Note that this component can be reused for other Mining Software Repositories (MSR) tasks. The remainder of this section discusses these subcomponents.

A. Data Collecting: The Data Collecting subcomponent collects the source code files referred to in the activity logs from the VCS and fixed change requests from the ITS. This subcomponent is supported by two functions, include the Source Code Entities Data Collecting and Fixed Change Request Data Collecting (see Fig. 5).

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 includes a number of items including a revision number, a commit date, a commit message, and the modified code. As mentioned previously, this metadata has an important role in integrating the data from the two software repositories and also for weighting the predicted source code locations. From the source code files, the identifiers which are used to name the 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 [33]. 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. This data-source is also used to provide the required data to identify source code locations in the Source Code Analyzing path. The time of creation or modification of each of the identifiers, which is used in the Weighting and Ranking component, is also extracted from the VCS log.

The second set of data is collected from the ITS and includes the reported change requests of the project. Specifically, a set of the reported change requests that are marked as FIXED is used. To resolve each of these change requests, a set of locations in the source code are modified. These sets of locations are used to identify the source code location for the new change request in the Change Request Analyzing path. As mentioned in Section 2.1, two fields of the change request are used – Summary and Description. These fields contain natural language text. The vocabulary used by the reporters to describe the change request provides a rich source of contextual information. Note that if
only data from change requests is used, the volume of data may not be sufficient and may affect the accuracy of the MSR tasks. Using both the change request information and the identifiers from source code provides a sufficiently rich data-set in order to more accurately identify the relevant locations.

B. Data Preprocessing: A rich set of content 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. In other words, different subcomponents are needed to preprocess the data from the VCS and the ITS.

- Source Code Preprocessing: From the source code locations found by 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 as 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. [34]. This results in a set of terms from which the identifiers are formed (hereafter referred as decomposed identifiers). Furthermore, reporters sometimes mention the name of a class or method directly in the explanation of the change requests [35]. Therefore, both the complete and decomposed identifiers are used to find similarities in the data of both of the two data-sources as well as the complete identifiers. More detail on the noun categorization and extraction is explained in Section 3.4.1. Also, nouns that contain symbols, digits or have less than three characters are filtered out as has been done in other mining repository approaches [26]. Finally, 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 of removal of inflectional endings only, and to return the base or dictionary form of a term. This form is known as the lemma [27,24].

C. Software Repository Integration: A project's software repositories are usually maintained separately [36]. Establishing links between these repositories is an important step in effective software maintenance activities [37]. In general, the actual source code locations which are modified to support the fixed change request are not known. To link the fixed change request to the corresponding source code locations, two different sources, include patches and commit messages, have been used [38,39,26]. First, the patches attached to the change requests
were examined to extract the location information. If the change request does not have an attached patch, the commit messages 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. After integrating the two repositories, a location-based integrated repository is created from the output of the Data Preprocessing subcomponent. This repository contains the extracted information from the source code revisions and fixed change requests, along with their associated metadata, and is linked to their corresponding source code locations. In other words, each source code location is connected to the extracted and preprocessed data, including:

- Extracted identifiers from the corresponding source code location.
- Extracted and refined nouns from the decomposed identifiers.
- Extracted and refined nouns from the Summary and Description of fixed change requests.

2.2.2. Weighting and ranking component

In this component, the data linked to source code locations by the Data Collecting and Integrating component is weighted using the proposed term-weighting technique, TATW. The data is weighted based on the textual similarity to the new change request, taking into consideration the value of similar text over time. Two factors are taken into account in the proposed term-weighting
technique. The first is term frequency with respect to either the source code identifiers or the text from a fixed change request. The second is the time at which 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 [27]. It focuses on the number of appearances of the term in a document or corpus. In the TF-IDF technique, perhaps the most-used term-weighting technique in IR, the weight of a term is related to the frequency of the term in the individual document, and the inverse frequency of the term across the corpus. As previously mentioned, the terms that are used in the proposed approach are restricted to only nouns. To be more specific, noun frequency is used instead of the more general term frequency in the proposed term-weighting technique.

In addition to considering the noun frequency, the proposed term-weighting technique considers the 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 and the classic term-weighting techniques. More specifically, in the proposed term-weighting technique, 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 a 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 over time.
- Frequency of a noun term \( i \) in a specific document. The specific document can be either a source code file \( (F_0) \) or consist of a set of fixed change requests that were reported for a source code file \( (FCR) \).
- Frequency of a noun term \( i \) in the corpus containing all the source code files of the project \( ((FPrj)_{res}) \) or all the fixed change requests reported to the project \( ((FPrj)_{res}) \). This parameter represents the generality of the noun term in the corpus.
- Time interval between when the new change request was reported \( (Date_{NCR}) \) and the time of creating or modifying the noun term \( i \).

The first three parameters are the frequencies of a noun term \( i \) at different levels of documentation. Existing term-weighting techniques primarily focus on using the frequency of a term in source code files across the project [29,27]. In the proposed technique, however, the value of the noun term over time is considered for the new change request, the fixed change requests, and the project source code files. This set of noun terms is referred to as the common nouns.

In the proposed term-weighting technique, the time difference between uses of a noun term in the new change request and the data sources is negatively correlated with the weight of the noun. In other words, a noun that is recently used in these resources (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 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 a logarithm strongly reduces the time difference value when compared with using a square root. On the other hand, the differences between the logarithms of larger values for the number of days is 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 difference 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 terms-weighting in the proposed approach. The decision to use the square root is also supported by the use with the time differences in other MSR fields [4,40].

The weighting of terms is done by two subcomponents: Identifier Weighting and Change Request Weighting. Based on the calculated weights for the noun terms, the scores of the source code locations are computed first by the Initial Location Ranking function, and then the source code locations are ranked in descending order. The details of these subcomponents are as follows.

A. Identifier Weighting: In this subcomponent, the source code identifiers and the selected noun terms of the decomposed identifiers (both referred to as \( N_i \)) are weighted with respect to those four parameters mentioned above. Accordingly, Eq. (1) is used to calculate the weight of noun term \( i (N_i) \) in the changset \( j (C_j) \) of the source code file \( k (F_k) \). A changset 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 changset \( j \). \( Freq_{N_i,F_k} \) is the frequency of the noun term \( i \) in the source code file \( k \) and is used in existing term-weighting techniques. It enumerates the occurrences of the noun term among all of the noun terms of a file [27]. \( Freq_{N_i,(FPrj)_{res}} \) indicates the frequency of the noun term \( i \) across all of the project's source code files in the VCS. Finally, \( Date_{NCR} - Date_{B,C_j} \) is the time difference between when a new change request was reported \( (Date_{NCR}) \) and the time that a noun term \( i \) was used in changset \( j (Date_{C_j}) \).

\[
\text{Weight}(N_i,C_j,F_k) = \frac{Freq_{N_i,C_j} + Freq_{N_i,F_k}}{Freq_{N_i,(FPrj)_{res}} \times \sqrt{Date_{NCR} - Date_{B,C_j}}}
\]

B. Change Request Weighting: For a fixed change request, the extracted nouns from the Summary and Description fields are weighted using Eq. (2). In this equation, \( Freq_{N_i,FCR_k} \) is the frequency of noun \( i (N_i) \) in the fixed change request \( j (FCR_j) \) and \( Freq_{N_i,FCR_k} \) is the frequency of \( N_i \) in the set of the fixed change requests' reported affecting source code file \( k (FCR_{k}) \). \( Freq_{N_i,(FPrj)_{res}} \) is the frequency of \( N_i \) among all of the fixed change requests reported for the project. Lastly, \( Date_{NCR} - Date_{NCR} \) is the time difference between the time when the new change request was reported and the resolution of the fixed change request \( j (\text{symbolized with } Date_{E,FCR_j}) \).

\[
\text{Weight}(N_i,FCR_j,F_k) = \frac{Freq_{N_i,FCR_j} \times Freq_{N_i,FCR_k}}{Freq_{N_i,(FPrj)_{res}} \times \sqrt{Date_{NCR} - Date_{E,FCR_j}}}
\]

C. Initial Location Ranking: Having calculated the weights of the nouns appearing in both the new change request, the fixed change requests 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 [27] and the summation of the term weights is the score of the file. (For more details, see Section 3.1). However, using the proposed technique, the weight of a noun term is related to the creation or modification time of the term in the data-sources. In other words, the noun can have more than one weight depending on when it was used.

Eq. (3) is used to compute the score of a source code file \( F_k \) based on the weights of the nouns collected from the change sets in the VCS \( \text{Score}(F_k)_{\text{VCS}} \) and the weights of the common nouns (i.e., those nouns appearing in both the new change request and the source code file). 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 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 \( k \).

\[
\text{Score}(F_k)_{\text{VCS}} = \sum_{j=1}^{\#\text{Changesets for } F_k} \sum_{i=1}^{\#\text{CommonNouns}} \text{Weight}(N_i, C_j, F_k)
\]

Eq. (4) is used to calculate the score of a source code file based on the fixed change requests \( \text{Score}(F_k)_{\text{FCR}} \). The weights of the nouns appearing in both the new change request and the fixed change requests for file \( k \) are summed up to determine the score of the file. As with Eq. (3), the weights of the common nouns between the new change request and the fixed change request \( j \) reported for file \( k \) are summed up, and these weights are then summed up again for all fixed change requests for file \( k \).

The output of this subcomponent is two ranked lists of source code locations in descending order, based on the calculated score for the file.

\[
\text{Score}(F_k)_{\text{FCR}} = \sum_{j=1}^{\#\text{FCR for } F_k} \sum_{i=1}^{\#\text{CommonNouns}} \text{Weight}(N_i, FCR_j, F_k)
\]

2.2.3. Final location ranking

This component has two subcomponents to combine the two ranked source code location lists from the previous component. These subcomponents are as follows:

A. Normalizing: Having computed the scores from the analysis of the source code files and the fixed change requests, the scores are combined for each source code location. However, the calculated scores of the source code locations from these subcomponents have different value ranges. The values therefore need to be normalized for comparison. 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 [29]. This mapping of values is done using Eq. (5). In this equation, \( \text{Score}(\text{max}) \) and \( \text{Score}(\text{min}) \) are the maximum and minimum scores obtained for the locations in each subcomponent of the Weighting and Ranking component.

\[
\text{NormalScore}(F_k) = \frac{\text{Score}(F_k) - \text{Score}(\text{min})}{\text{Score}(\text{max}) - \text{Score}(\text{min})}
\]

\[
\text{NormalScore}(F_k) \leq 1
\]

B. Final Location Ranking: Having normalized the scores, the scores are then used to again rank the locations. For locations that appear in ranked lists, the score is the summation of the scores calculated by each subcomponent (see Eq. (6)). The final scores are then ranked in descending order so that the most relevant locations for the new change request are ranked at the top.

\[
\text{FinalScore}(F_k) = \text{NormalScore}(F_k)_{\text{FCR}} + \text{NormalScore}(F_k)_{\text{VCS}}
\]

3. Experimental evaluation setup

In this section, the setup for empirically evaluating the proposed feature location approach (NBFL) and the proposed term-weighting technique (TATW) is presented. This section follows the same organization for empirical evaluation of software systems recommended by Wohlin et al. [41]. The first section presents how open-source projects were reviewed to find subject systems (See Section 3.1.1) and how the feature location literature was reviewed to identify previously evaluated approaches and/or proposed term-weighting techniques (See Section 3.1.2). Next, the hypotheses and research questions are presented in Section 3.2. Section 3.3 presents the experimental design to assess the hypotheses. Finally, how the data was collected and preprocessed, as well as an implementation of the proposed approach, is explained in Section 3.4.

3.1. Context selection

This section first presents the methodology by which the four open-source subject systems were selected. Next, a description of how VSM was chosen as the baseline feature location approach against which NBFL is compared.

3.1.1. Subject systems

To evaluate the proposed feature location approach and term-weighting technique, a set of real-world projects is needed. As data from open-source projects is more readily available, such projects were investigated to identify the subject systems for the evaluation of NBFL and TATW. Although, there are many open-source projects that could be used, subject systems were selected based on three 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 volumes of data in their repositories.** 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 in Section 2.2.2, time is an important factor in the proposed term-weighting technique, and correspondingly in the proposed feature location approach. It is assumed that the volume 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 weighting of the source code locations.

Based on these criteria, four subject systems – JDT, AspectJ, NetBeans, and Rhino - were chosen as the subject systems. The JDT project provides the tool plug-ins that implement a Java integrated development environment (IDE). AspectJ is a simple and practical extension of the Java programming language that adds aspect-oriented programming (AOP) capabilities to Java. NetBeans is an IDE for Java development. It also supports other languages, in particular PHP, C/C++, and HTML. Rhino is an implementation of JavaScript 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 and change requests, are shown in Table 2. According to the assumption that the volume of data represents

\[\text{http://www.eclipse.org/jdt/} \]
\[\text{http://www.eclipse.org/aspectj/} \]
\[\text{https://netbeans.org/} \]
the scale of the project, JDT, AspectJ, Netbeans, and Rhino can be treated as large (JDT), medium (Netbeans) and small-scale (AspectJ and Rhino) projects. Using these four subject systems, it is expected that the applicability of the NBFL approach on different scales of projects will be shown. Also, the different average volumes of changes per day for the four subject systems indicates differences in their rates of project evolution. Therefore, the effect of time in weighting the terms for projects with different speeds of evolution can be evaluated. From each subject system, a set of change requests were selected as the test set to evaluate the proposed approach and technique. The details of the test set selection are presented in Section 3.4.1.

3.1.2. Baseline approach for comparison

The results obtained from applying the proposed approach and term-weighting technique on the subject systems need to be compared with existing feature location approaches. Therefore, the feature location literature was reviewed to find suitable feature location approaches and term-weighting techniques for an unbiased comparison of these two aspects. According to the recent feature location survey in [9], IR is the most commonly used feature location method for analyzing the text resources of historical software repositories. Many models have been used in prior IR-based feature location research. Some of the important models include: Smoothed Unigram Model (SUM) [28], Latent Dirichlet Allocation (LDA) [42], Latent Semantic Indexing (LSI) [12], and Vector Space Model (VSM) [29]. SUM and LDA are probabilistic models, while VSM and LSI are algebraic models. However, probabilistic models do not often use term-weighting techniques [17] and therefore are not appropriate for comparison to the proposed approach and term-weighting technique. This means that algebraic models that use terms-weighting techniques were selected for comparison. Rao and Kak [28] evaluated the performance and accuracy of the IR models for locating software features and found that VSM is the best algebraic model. Correspondingly, to evaluate the proposed feature location approach, VSM was selected for comparison.

In VSM, each document is represented as a vector of term weights. These term weights are usually calculated using a specific term-weighting technique [27]. The relevancy score between a query and the source code locations is computed as the cosine similarity between the corresponding vector representations of the source code locations. Although there are several term-weighting techniques used with VSM, the most common one is TF-IDF [27]. This technique computes a weight for each term in each document of a corpus as a combination of the term frequency (TF) and the inverse document frequency (IDF). TF is the number of appearances of term $t$ in document $d$ $(TF_{t,d})$. The document frequency for term $t$ is the number of documents in the corpus that contains the term $(DF_{t})$. IDF is the inverse of document frequency calculated using Eq. (7). In this equation, $N$ is the total number of documents in a corpus. TF-IDF, a combination of TF and IDF, is calculated by Eq. (8).

$$IDF_{t} = \log \frac{N}{DF_{t}}$$

$$TF-IDF_{t,d} = TF_{t,d} \times IDF_{t}$$

$$Score(d) = \sum_{t}^{A(t,d)} TF-IDF_{t,d}$$

In this study, TraceLab\(^7\) [43] – a recently published experimentation framework – was 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 that utilizes VSM with a TF-IDF term-weighting technique. Hereafter, VSM refers to the VSM approach that utilizes the TF-IDF term-weighting technique. Fig. 6 shows the setup of the VSM experiment in TraceLab implemented by Dit et al. [43]. This experimental setup was used to evaluate the results of VSM against those of the proposed approach, NBFL. First, the input data was converted into an acceptable format for use with Tracelab. Next, the common preprocessing steps of cleaning.

\(^7\) http://coest.org/coest-projects/projects/tracelab.
stemming and stopword filtering for the queries were done. Lastly, the similarity score between the new change request and the source code locations was calculated.

3.2. Hypotheses and research questions

The goal of evaluating the proposed approach, NBFL, and the proposed technique, TATW, was to address the following research questions:

- **RQ1**: Does NBFL outperform VSM, a baseline feature location approach?
- **RQ2**: What is the impact of the use of time-metadata in the term-weighting process for feature location?
- **RQ3**: What is the impact of using only noun terms instead of using all of the terms in a feature location approach?

These research questions led to the following set of hypotheses based on accuracy and effectiveness metrics (Section 3.3.1). Statistical testing of these hypotheses can demonstrate possible significance of the improvements, for the proposed feature location approach. The null hypotheses for this research are as follows:

- **H0,NBFL,VSM**: There is no significant difference between the effectiveness/accuracy of NBFL and VSM.
- **H0,TATW,TF-IDF**: There is no significant difference between the effectiveness/accuracy of TATW and TF-IDF.
- **H0,All-Terms,Noun-Terms**: There is no significant difference between the effectiveness/accuracy of NBFL when using all of the terms, and when using only noun terms.

The null hypothesis for either accuracy or effectiveness can be derived analogously from the above list of hypothesis. If a null hypothesis can be rejected with high confidence (95%), the corresponding alternative hypothesis is supported. The alternative hypotheses are as follows:

- **H1,NBFL,VSM**: The effectiveness/accuracy of NBFL is significantly better than the effectiveness/accuracy of VSM.
- **H1,TATW,TF-IDF**: The effectiveness/accuracy of TATW is significantly better than the effectiveness/accuracy of TF-IDF.
- **H1,All-Terms,Noun-Terms**: The effectiveness/accuracy of NBFL using only noun terms as its input is significantly better than the effectiveness/accuracy of using all of the terms.

3.3. Experimental design

The proposed feature location approach and term-weighting technique were applied to test sets for each of the subject systems. The test sets selected for each subject system are considered as the independent variables. The results of the evaluation were determined using the identified metrics, presented below. These metrics are taken as the dependent variables. For further analysis of the dependent and independent variables, statistical tests were conducted. These tests are also described below.

3.3.1. Descriptive analysis and metrics

To perform a descriptive analysis of the experiments, a set of metrics was identified to evaluate the results. The metrics used are as follows:

- **Top N Rank or Likelihood [28,29]**: For a new change request, if the 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 [24,41]**: 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) [45]**: 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. The MRR is the average of the reciprocal ranks of a set of queries and is calculated using Eq. (10).

\[
MRR = \frac{1}{Q} \sum_{i=1}^{Q} \frac{1}{R_i}
\]  

- **Mean Average Precision (MAP) [45]**: The 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 Eq. (11).

\[
P(k) = \frac{\text{Number of Positive Instances In Top k Positions}}{k}
\]  

3.3.2. Statistical analysis

The descriptive analysis of whether one feature location approach or term-weighting technique significantly outperforms another one is not enough. A statistical analysis is also needed to determine whether the difference between the obtained effectiveness and accuracy results is significant or not. To analyze the accuracy, the results from recommending the Top 1 to Top 10 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 [41]. As this research is dealing with paired data, paired statistical tests are needed. Based on the results of testing for a normal distribution, the paired t-test was used for statistical comparison of the two approaches or techniques for normally distributed results, and a two-way ANOVA was conducted to compare multiple results. For results that are not normally distributed, the permutation Wilcoxon test was conducted for paired data comparison and the Permutation 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 \( \alpha \), it can be said that the null hypothesis is rejected with 95% confidence, obtained by subtracting \( \alpha \) from \( \alpha (1 - \alpha) \).

In addition to the statistical significance (the test’s p-value), the effect size of the difference between the approaches or techniques 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 normal distributed data, the effect size of the Hedges’ g is used, and for the non-normal distributed data, the Cliff’s delta is calculated.\(^\text{4}\) Hedges’

\(^{4}\) http://softeng.polito.it/software/effsize/.
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 [46]. The thresholds are \(|g| < 0.2\) for “negligible”, \(|g| < 0.5\) for “small”, \(|g| > 0.8\) for “medium”, and “large” for \(|g| > 0.8\). The magnitude of the Clfif’s delta is assessed using the thresholds provided by Romano [47] that consider \(|\delta| < 0.147\) as “negligible”, \(|\delta| < 0.33\) as “small”, \(|\delta| < 0.474\) as “medium”, and \(|\delta| > 0.474\) as “large”.

Note for some of the experiments, like VSM and the proposed approach in the case of using all of the terms, the approaches were not able to identify a correct location for a few of the change requests in the test set (Section 4). This may be the result of a high volume of noisy data when all of the terms are used by either approach. In these cases, the approaches would have no data to be paired with the data from the other approaches for statistical comparison. The percentage of the correct locations identified by each approach is given in Section 4. Therefore, only cases where both approaches or techniques found a correct location are used for the statistical test. This means that some of the change requests for which NBFL determined a correct location are not used in the statistical test.

3.4. Execution

3.4.1. Data collection and preparation

As mentioned in Section 3.1.1, four open-source projects were selected as the subject systems for evaluating NBFL. Two repositories of these projects – the VCS and the ITS – were mined to provide the required data. Different tools and techniques were used to collect the data and metadata because of the different formats used to record the data in these two repositories. To collect the data from the VCS, the commit log was first downloaded from the source code repositories of the projects. Different projects used different version control systems, such as CVS [9], Git [10] and Mercurial [11]. Our subject systems used CVS (JDT and AspectJ), Git (Rhino), and Mercurial (Netbeans) as their version control systems. We used CVSANALY [12] to transfer the data from a project’s VCS to our local repository. CVSANALY organizes this data based on the source code revisions which includes the revision number, the commit date, and the commit message. Based on the source code revision numbers, the identifiers were collected 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. Depending on the project’s programming language, a suitable tool was employed to extract the identifiers from the generated source code files. In this research, all of the subject systems were written in Java. Therefore, the JELDoclet [13] 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 \((jclass), (jfield), (jmethod), \text{and} (jparams)\) tags are extracted to collect the name of class, fields, methods, and method parameters. As mentioned in Section 2.2.1, identifiers are typically a concatenation of a set of terms. To decompose an identifier, the approach recommended by Butler et al. [34] was used to produce a set of terms used to create the identifier. To extract the noun terms from the decomposed identifiers, the ANNI [14] plug-in of GATE [48] was employed. To determine the roles of the decomposed identifiers, the part-of-speech (POS) tagger of the ANNI 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 ruleset which are used to tag the input terms. From the output of the POS tagger, the terms which are tagged as a noun (such as NN and NNP) were extracted. Recall that both the complete identifier 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 file were added.

From the project’s ITS, only the change requests marked as FIXED were collected. First, a list of ids for the project’s fixed change requests was created using a search on the status of the reports in the repository. Next, the corresponding change requests were downloaded and stored in XML format. For each fixed change request, the text of the Summary and the Description fields was extracted. To analyze the text, Named Entity Recognition (NER) [31] was used. To apply NER, the ANNI plug-in of GATE was employed. First of all, the whole text was split into sentences by the sentence splitter component of ANNI. Next, the tokenizer component was used to separate the terms and symbols into tokens. Then, the POS tagger component was employed to determine the roles of the terms in the sentences. The output of the POS tagger is a list of tagged terms as a noun, verb and so forth. From these tagged terms, the ones that were tagged as nouns (such as NN or NNP), were selected to create a noun index for each change request. All of the nouns in the identifier noun indexes and the change request noun indexes were then lemmatized using the Stanford CoreNLP API [15]. Recall that these noun indexes were used as the data-sources in the feature location approaches.

For evaluating the proposed approach and technique, for each subject system, a randomly selected set of fixed change requests was chosen as the test set. The remaining set of change requests was used as the training set. Note that as these change requests were already fixed, the set of the source code locations that were modified can be determined and used as the oracle for the test set. These source code locations were determined using the same technique used in [26]. In this technique, the locations of a change request can be determined in two ways. One way is from patches attached to the change requests and the another way is change request’s ids that appear in the commit messages of logs from the VCS [38]. First, we examine any patch(es) attached to the change request and extract the location in the source code. If the change request does not have an attached patch, the commit messages in the project’s VCS are used to determine the link. This technique is used to link and integrate the VCS and ITS of the project.

There is no standard for the optimal test set size for evaluating a feature location approach. In most of the previous feature location researches, a few change requests from a specific release, meaning a specific period of project life cycle, were selected as a test set to evaluate an approach [49,3]. The number of change requests

9 http://sourceforge.net/apps/trac/sourceforge/wiki/CVS.
11 http://sourceforge.net/p/forge/documentation/Mercurial/.
12 http://metricsgrimoire.github.com/CVSANALY/.
14 http://www.aktors.org/technologies/annie/.
15 http://gate.ac.uk/.
randomly selected for use as the test set in this study was 200 change requests. This value was chosen as it is more than that used by Poshyvanyk et al. [12] (three from Eclipse and five from Mozilla) but less than that used by Zhou et al. [29] (3000 from Eclipse). As mentioned in Section 1, the change requests reported in different periods of the project life cycle may have different goals and requirements. Therefore, to assess the applicability of the proposed approach at different periods of project life cycle, the 200 change requests were chosen randomly from all of the fixed change requests of subject systems.

From each of these test sets, a subset of change requests was selected that were reported at roughly the same time. This was done to evaluate the proposed feature location approach in comparison with VSM. This test set is called the VSM test set. The VSM test sets for JDT, AspectJ, Netbeans, and Rhino had 78, 74, 32, and 25 change requests, respectively. All of the data-sets and test sets that were used in the evaluation are freely available online, and other researchers are welcome to use this information in replicating this work.

3.4.2. Experimental implementation

The proposed approach discussed in Section 2 was implemented as a TraceLab experiment. Fig. 7 shows an overview of our TraceLab experiment. This experiment is available online for those wanting to replicate this work. The data collected from the source code files and fixed change requests of each subject system was preprocessed and integrated using the methods found in [26]. As mentioned in Section 2.2.1, the output of this step is a set of noun terms extracted from the text resources of the VCS and ITS software repositories. The noun terms, along with their associated metadata, form the input of our TraceLab experiment. This data was imported using the SEMERU-ICPC’12-CorpusImporter and SEMERU-ICPC’12-QueriesImporter components, created by Moritz and his colleagues [43]. The imported data was used to calculate the scores of the source code files by the Identifier Weighting subcomponent and the Change Request Weighting subcomponent (see Section 2.2). These scores were then normalized, as described in Section 2.2.3, and ranked by the Final Location Ranking subcomponent based on their relevancy for each change request in the test set. The output of this experiment was a ranked list of files in descending order of the scores.

The results of these experiments were evaluated using the identified metrics (Section 3.3.1), and compared with the results of VSM implemented in TraceLab (see Section 3.1). To compare the effectiveness and accuracy of the proposed term-weighting technique with TF-IDF, NBFL was executed with TATW, referred to as NBFL_TATW, and with TF-IDF, referred to as NBFL_TF-IDF. To assess the effect of using only nouns, the results of the proposed approach were compared with the results of the proposed approach when using all of the extracted terms.

4. Experimental results and analysis

This section presents the descriptive results of the experiment and the outcomes of the statistical analysis. As mentioned in Section 3, a set of evaluations was performed to answer the research questions. First, Section 4.1 presents the results of the evaluation of the proposed approach and the baseline VSM-based feature location approach. Next, a comparison of using the proposed term-weighting technique versus the TF-IDF technique is presented in Section 4.2. Finally, the impact of using only noun terms is analyzed in Section 4.3. The statistical results of these evaluations are given in Section 4.4 and a discussion of both the descriptive and statistical results is presented in Section 4.5.

4.1. NBFL vs. VSM

This section presents the descriptive results of the proposed approach, NBFL, and VSM (with TF-IDF) using the VSM test sets of the subject systems. Table 3 shows the results of using the NBFL approach and the VSM approach on the subject systems. The first two columns list the subject systems and the approaches. The Top1, Top5, and Top10 columns show the average accuracy of the two approaches when recommending the top 1, 5 and 10 entries from the ranked list of recommendations. As shown in this table, NBFL has a better average accuracy than VSM for the top 10
ranked locations on the JDT, AspectJ, Netbeans, and Rhino projects. The improvement over VSM for these projects was 28%, 34%, 50%, and 12%, respectively. Fig. 8 shows graphs of the top N ranked results for the NBFL and the VSM approaches. As shown by this figure, the accuracy of the NBFL approach is better than the VSM approach for all subject systems. The sixth column of Table 3 presents the MAP metric results. These results show a performance improvement up to 13% when using NBFL. The last three columns of the table present the effectiveness metrics – Average Effectiveness, Median Effectiveness and MRR. The results of the average and median effectiveness for NBFL show that less developer effort is needed to find a correct source code location when using NBFL. Specifically, if VSM is used for feature location, the developer would need to check on average 220 more locations, in the worst case (AspectJ). Finally, the results for the MRR metric indicate an effectiveness improvement of as much as 17% (Rhino).

Table 3
The results of NBFL and VSM.

<table>
<thead>
<tr>
<th>Project</th>
<th>Approach</th>
<th>Top1 (%)</th>
<th>Top5 (%)</th>
<th>Top10 (%)</th>
<th>MAP (%)</th>
<th>Effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDT</td>
<td>NBFL</td>
<td>15.38</td>
<td>53.84</td>
<td>67.94</td>
<td>34.13</td>
<td>48.99</td>
</tr>
<tr>
<td></td>
<td>VSM</td>
<td>11.53</td>
<td>33.33</td>
<td>39.74</td>
<td>20.89</td>
<td>130.88</td>
</tr>
<tr>
<td>AspectJ</td>
<td>NBFL</td>
<td>4.05</td>
<td>39.18</td>
<td>56.75</td>
<td>3.99</td>
<td>43.41</td>
</tr>
<tr>
<td></td>
<td>VSM</td>
<td>5.4</td>
<td>14.86</td>
<td>22.97</td>
<td>1.03</td>
<td>263.49</td>
</tr>
<tr>
<td>Netbeans</td>
<td>NBFL</td>
<td>12.5</td>
<td>53.12</td>
<td>71.87</td>
<td>4.9</td>
<td>15.83</td>
</tr>
<tr>
<td></td>
<td>VSM</td>
<td>6.25</td>
<td>12.5</td>
<td>21.87</td>
<td>0.45</td>
<td>73.97</td>
</tr>
<tr>
<td>Rhino</td>
<td>NBFL</td>
<td>36</td>
<td>76</td>
<td>84</td>
<td>4.96</td>
<td>5.96</td>
</tr>
<tr>
<td></td>
<td>VSM</td>
<td>24</td>
<td>56</td>
<td>72</td>
<td>2.89</td>
<td>8.6</td>
</tr>
</tbody>
</table>

Fig. 8. The accuracy of NBFL and VSM for recommending the top N ranked locations.

Fig. 9 shows box plots for the effectiveness metric (i.e. the location of relevant source code files) for the subject systems. In this figure, the diamonds indicate the average effectiveness measure, and the yellow and red boxes are the upper and lower quartiles, respectively. The line between the boxes represents the median effectiveness value. The whiskers above and below the boxes denote the maximum and minimum effectiveness values, respectively. The percentages on the top of the effectiveness box plot indicates the percentage of the change requests having at least one relevant file in the list of recommendations. Note that the graphs for JDT, Aspect, Netbeans, and Rhino have different scales as a result of a different number of files in their respective repositories. This figure shows that for NBFL, the box plots are lower than VSM. This suggests that developers would potentially require less effort to locate relevant files for a change request when using NBFL.

Fig. 9. Effectiveness evaluation results for NBFL and VSM for the subject systems.
The figure also shows that, in general, the effectiveness of NBFL is significantly better than the VSM for all of the subject systems, with one exception. For the Rhino project, the top whisker shows that NBFL has a larger maximum effectiveness value of the source code file location when compared with VSM. However, in all other aspects (i.e. median, average, first and third quartiles) NBFL is significantly better than the VSM for this project.

4.2. TATW vs. TF-IDF

The NBFL approach 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 NBFL using the TATW technique is applied to the subject systems NBFL<sub>TATW</sub>. Then, TATW is replaced by TF-IDF and the same experiment runs NBFL<sub>TF-IDF</sub>. For the evaluation, the main test set, which includes randomly selected change requests from subject systems, was used.

Table 4 shows the results of evaluation comparing the use of TATW, and TF-IDF in NBFL on the test sets. For JDT, AspectJ and Netbeans, the use of TATW shows a notable improvement. However, for the Rhino project, the result of using TF-IDF is better than that of using TATW. This is likely the result of the low evolution speed of this project, which is further discussed in Section 4.5.

![Fig. 10. The accuracy evaluation of NBFL, TATW, and TF-IDF in recommending top N ranked locations.](image-url)

<table>
<thead>
<tr>
<th>Project</th>
<th>Technique</th>
<th>Top1 (%)</th>
<th>Top5 (%)</th>
<th>Top10 (%)</th>
<th>MAP (%)</th>
<th>Effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDT</td>
<td>TATW</td>
<td>24</td>
<td>53.5</td>
<td>69</td>
<td>41.43</td>
<td>61.15</td>
</tr>
<tr>
<td></td>
<td>TF-IDF</td>
<td>21.5</td>
<td>45.5</td>
<td>54.5</td>
<td>1.02</td>
<td>99.02</td>
</tr>
<tr>
<td>AspectJ</td>
<td>TATW</td>
<td>12</td>
<td>36.5</td>
<td>51.5</td>
<td>13.72</td>
<td>82.29</td>
</tr>
<tr>
<td></td>
<td>TF-IDF</td>
<td>11</td>
<td>31</td>
<td>41.5</td>
<td>7.02</td>
<td>91.16</td>
</tr>
<tr>
<td>Netbeans</td>
<td>TATW</td>
<td>28.5</td>
<td>62</td>
<td>76.5</td>
<td>43.22</td>
<td>15.02</td>
</tr>
<tr>
<td></td>
<td>TF-IDF</td>
<td>21</td>
<td>51</td>
<td>62.5</td>
<td>33.22</td>
<td>19.97</td>
</tr>
<tr>
<td>Rhino</td>
<td>TATW</td>
<td>31</td>
<td>70.5</td>
<td>84.5</td>
<td>60.45</td>
<td>6.25</td>
</tr>
<tr>
<td></td>
<td>TF-IDF</td>
<td>36.5</td>
<td>77.5</td>
<td>87.5</td>
<td>65.99</td>
<td>4.95</td>
</tr>
</tbody>
</table>

The effectiveness comparison shows that TATW is more effective than TF-IDF on all subject systems, again except for Rhino. This indicates that the developer, in the worst case (JDT), needs to check 38 locations more if TF-IDF is used than if TATW is used. In other words, the use of TATW leads to less developer effort to find the correct source code location. Additionally, the MRR results show that the TATW technique is up to 10% more effective than the TF-IDF technique, except for Rhino.

Fig. 11 shows box plots of the effectiveness results. Note that the effectiveness results show the same pattern as that for accuracy. Once again, TATW is shown to be more effective than TF-IDF for all of the subject systems, except for Rhino. Again, this 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 the TATW technique produces better results for projects with a higher speed of evolution, such as JDT. This conclusion is further discussed in Section 4.5.

4.3. Impact of noun usage

To evaluate the use of only noun terms, first, all of the terms were extracted from the text resources in the software repositories and used in the NBFL approach with the TATW technique (NBFL<sub>All</sub>). Then only the noun terms were extracted from the text resources and the same experiment was run using those terms (NBFL<sub>TATW</sub>). Again, the set of randomly selected change requests, the main test set, was used in the evaluation.

Table 5 presents the descriptive results of NBFL<sub>TATW</sub> when using either all of the terms or only the noun terms. As shown in this table, the use of only nouns improves the average accuracy of the top 10 ranked locations for JDT, AspectJ, Netbeans, and Rhino by around 16%, 12%, 21%, and 28%, respectively. Fig. 12 shows the accuracy graphs from 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. The results for the MAP metric show a large performance improvement by as much as 58% (Rhino). The effectiveness results suggest that using only noun terms results in 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 71 more locations in the worst case (JDT). Finally, the MRR result reveals an improvement of as much as 21%, and indicates that the use of only noun terms is more effective than using all of the terms.

Fig. 11. The effectiveness results of TATW and TF-IDF.

Table 5
The results of NBFL-TATW using either all of the terms or using only noun terms.

<table>
<thead>
<tr>
<th>Project</th>
<th>Input</th>
<th>Top1 (%)</th>
<th>Top5 (%)</th>
<th>Top10 (%)</th>
<th>MAP (%)</th>
<th>Effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Average</td>
<td>Median</td>
<td>MRR (%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JDT</td>
<td>Noun terms</td>
<td>24</td>
<td>53.5</td>
<td>69</td>
<td>41.43</td>
<td>43.67</td>
</tr>
<tr>
<td></td>
<td>All terms</td>
<td>14</td>
<td>39</td>
<td>49</td>
<td>0.82</td>
<td>115.21</td>
</tr>
<tr>
<td>AspectJ</td>
<td>Noun terms</td>
<td>12</td>
<td>36.5</td>
<td>51.5</td>
<td>13.72</td>
<td>58.75</td>
</tr>
<tr>
<td></td>
<td>All terms</td>
<td>8</td>
<td>26</td>
<td>39</td>
<td>6.03</td>
<td>66.58</td>
</tr>
<tr>
<td>Netbeans</td>
<td>Noun terms</td>
<td>28.5</td>
<td>62</td>
<td>76.5</td>
<td>43.22</td>
<td>10.42</td>
</tr>
<tr>
<td></td>
<td>All terms</td>
<td>22</td>
<td>41.5</td>
<td>55.5</td>
<td>31.13</td>
<td>14.37</td>
</tr>
<tr>
<td>Rhino</td>
<td>Noun terms</td>
<td>31</td>
<td>70.5</td>
<td>84.5</td>
<td>60.45</td>
<td>3.7</td>
</tr>
<tr>
<td></td>
<td>All terms</td>
<td>15.5</td>
<td>43</td>
<td>56.5</td>
<td>2.79</td>
<td>11.35</td>
</tr>
</tbody>
</table>

Fig. 12. The results of the accuracy evaluation of NBFL with TATW using all terms and using only noun terms.

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 shows the number of terms extracted from the data-sources for each project dataset, for both extracting only the noun terms and extracting all of the terms. As would be expected, the Difference row of this table shows that the volume of data decreased significantly when extracting only nouns. For JDT, AspectJ, Netbeans, and Rhino, the number of terms in the dataset was reduced by 60%, 27%, 38%, and 36%, respectively. With the reduction in the amount of data that needs to be analyzed, the execution time of the feature location approach naturally decreases.
4.4. Statistical analysis

In this section, the statistical analysis of the obtained results is presented. 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 value of two criteria, Skewness and Kurtosis, and one normality test, Shapiro–Wilk, are investigated. The Skewness and Kurtosis values used were between −2 and +2 and the Shapiro–Wilk p-value was higher than 0.05.

Table 7 presents the results of the normality evaluation of NBFL using either only the noun terms, NBFL_{TATW}, or all of the terms, NBFL_{TF-IDF}, NBFL, using the TF-IDF weighting technique, NBFL_{TF-IDF}, and VSM. These results indicate that the descriptive accuracy results are mostly normal distributed, and therefore a parametric test can be used for statistical analysis. For example, the p-value of the Shapiro–Wilk for the accuracy of VSM on JDT is less than 0.05, and with a 98% confidence level, the results can be said to be normal. Also, due to the Skewness and Kurtosis results which are between −2 and 2, the results are almost normal. Therefore, the paired permutation T-test was used for statistical comparison of the two approaches or techniques. In the case of comparing multiple results, the two-way ANOVA was conducted and a 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 mostly non-normal. 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 permutation Wilcoxon test was used for comparing the results of two approaches or techniques and the Permutation Friedman test was used for comparing multiple results.

In comparing NBFL and VSM, the statistical test for the accuracy results shows the rejection of the null hypothesis, that there is a significant difference in accuracy between the two approaches. This means an acceptance of the alternative hypothesis. In other words, there is a significant difference in the accuracy of NBFL and VSM. 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 – that the effectiveness of NBFL over VSM is statistically significant for all but one subject system. For the one exception, Rhino, the p-value obtained from the test is a bit more than $\alpha$ (0.053). In other
words, with 94.7% confidence, it can be said that the difference between the two approaches is significant. The effect size values for the effectiveness metric indicated a large practical significance for AspectJ and Netbeans and a small practical significance for JDT. For the Rhino project that the p-value was more than z, the effect size showed a small improvement. In general, the statistical analysis of the obtained results shows that both the implementations in effectiveness and in the accuracy of NBFL over VSM are statistically significant and the effect sizes mostly suggest a large practical significance.

NBFL 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. Note that all of the statistical results for the multiple group analysis are less than z (p-value < 0.05). This shows the overall superiority of NBFL with respect to accuracy and effectiveness for feature location. In other words, NBFL significantly improves the process of feature location by using time-aware term-weighting and using only noun terms. The results of the two-way ANOVA test reveal that there is no interaction between the time consideration and the noun usage. The results of this test for all the projects show an R-Squared of more than 0.98 with low error. This result also points to the low probability of the existence of any important co-variables.

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: NBFL

TATW with NBFL

TF-IDF, and NBFL using only noun terms with NBFL using all the terms. As shown in Table 8, the accuracy and effectiveness results of NBFL

TATW and NBFL

TF-IDF are significant for the JDT, AspectJ and Netbeans projects. For the effectiveness results of NBFL

TATW on AspectJ, with 92.3% confidence, it can be said that the results are significantly better than NBFL

TATW. However, the statistical analysis for the Rhino project (marked with an asterisk) indicates a significant difference between the results of NBFL

TATW and NBFL

TF-IDF, with average accuracy and effectiveness of NBFL

TF-IDF being better than NBFL

TATW. The reasons of this exception are discussed in Section 4.5.

Furthermore, to complement inferential statistics, the effect size values (Hedges' g and Cliff's delta) are reported in Table 8. The effect size values for the accuracy of NBFL

TATW against NBFL

TF-IDF suggest a medium practical significance for JDT, AspectJ and Netbeans. Although, the p-value shows the superiority of NBFL

TF-IDF for the Rhino project, the effect size value indicates the effect size to be of small practical significance. For the effectiveness measurement, the effect size value reveals a negligible effect size for JDT and AspectJ and a small practical significance on Netbeans. On the Rhino project that the p-value indicates the superiority of the NBFL

TF-IDF, the effect size suggests a negligible difference. The reason for the small effect size for the effectiveness measurement is discussed in Section 4.5.

When using noun terms only, the results of NBFL 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 all the subject systems. For the effectiveness metric, the effect size values suggest a medium practical significance for Rhino, a small practical significance for JDT and Netbeans, and a negligible improvement for AspectJ. The analysis of the obtained descriptive and statistical results are discussed in the next section.

4.5. Discussion

The analysis of the descriptive and statistical results for the NBFL approach compared to the feature location baseline, VSM, shows the superiority of NBFL for all identified metrics. As shown in Table 3, and Figs. 8 and 9, for both the accuracy and effectiveness metrics, NBFL outperforms VSM on all subject systems. Moreover, the statistical analysis of the results emphasizes the significance of the improvement of NBFL for feature location. Furthermore, the values of the effect size mostly suggest a large practical significance for NBFL when compared with VSM. These results lead to answering RQ1 and concluding that NBFL outperforms the feature

<table>
<thead>
<tr>
<th>Table 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>The statistical test results.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Project</th>
<th>Multiple test</th>
<th>Two sample hypotheses</th>
<th>Effect size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>JDT</td>
<td>Two-way ANOVA</td>
<td>NBFL VS VSM</td>
<td>Hedges' g 1.35</td>
</tr>
<tr>
<td></td>
<td>AspectJ</td>
<td>Two-way ANOVA</td>
<td>NBFL(Noun-Only, All-Terms) VS VSM</td>
<td>Hedges' g 0.85</td>
</tr>
<tr>
<td></td>
<td>Netbeans</td>
<td>Two-way ANOVA</td>
<td>NBFL(Noun-Only, All-Terms) VS VSM</td>
<td>Hedges' g 0.66</td>
</tr>
<tr>
<td></td>
<td>Rhino</td>
<td>Two-way ANOVA</td>
<td>NBFL(Noun-Only, All-Terms) VS VSM</td>
<td>Hedges' g 0.85</td>
</tr>
<tr>
<td>Effectiveness</td>
<td>JDT</td>
<td>Permutation Friedman</td>
<td>NBFL VS Wilcoxon</td>
<td>Cliff's delta 0.23</td>
</tr>
<tr>
<td></td>
<td>AspectJ</td>
<td>Permutation Friedman</td>
<td>NBFL(Noun-Only, All-Terms) VS Wilcoxon</td>
<td>Cliff's delta 0.19</td>
</tr>
<tr>
<td></td>
<td>Netbeans</td>
<td>Permutation Friedman</td>
<td>NBFL(Noun-Only, All-Terms) VS Wilcoxon</td>
<td>Cliff's delta 0.19</td>
</tr>
<tr>
<td></td>
<td>Rhino</td>
<td>Permutation Friedman</td>
<td>NBFL(Noun-Only, All-Terms) VS Wilcoxon</td>
<td>Cliff's delta 0.19</td>
</tr>
</tbody>
</table>
location baseline. In other words, considering the time of use of the noun terms improves the feature location process.

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 (Table 4). The statistical testing of these results confirms the significance of the differences and the effect size magnitudes for the accuracy and effectiveness metrics suggest a medium to negligible improvement for these projects. 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 approaches, the effect size cannot reflect the real level of improvement.

Based on these descriptive and statistical results, it can be concluded that the TATW technique outperforms the TF-IDF technique, specifically for those projects with medium and high evolution speeds. These results lead to answering RQ2, 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 approach for projects with a medium to high evolution speed, such as JDT. This result is encouraging, as these are the types of projects that most need assistance with feature location. This improvement is likely due to the fact that the IR methods originated from a natural language context where time-metadata was not available, as was discussed in Section 1. The novelty of NBFL-TATW is the consideration of the evolution of the text resources over time. This will naturally provide a greater effect for a project with a high evolution speed, as the weights of the terms are affected by the volume of modifications to the text resources over time.

As it was shown in Table 4, 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 NBFL-TATW is no better than NBFL-TF-IDF for projects with a low evolution speed. This result shows the effect of the volume of modified data over time. As indicated in Table 2, 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 will not provide considerable improvement over other term-weighting techniques, and in the exceptional case, might have a negative effect. 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 terms of using only nouns, the descriptive results show a significant improvement for all metrics (Table 5). The results of the statistical test on the descriptive results indicates the significant improvement when using only the noun terms. Furthermore, the effect size values suggest a large practical significance for the accuracy metric and a negligible 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 volume of the datasets (Table 6) thereby improving the execution time for the feature location approach. The data collected from the software repositories is very noisy. Using only the nouns, instead of all the terms, not only provides enough information with which to identifying feature locations, but also reduces the amount of noisy data. As with any feature location approach, 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.

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 [30]. Also, using only the noun terms can summarize the source code data and enhance MSR tasks [50]. In summary, the use of only noun terms has a high positive impact in the feature location process and strongly supports the third alternative hypothesis (H3). Collectively, these reasons answer RQ3, the impact of using only noun terms in feature location, that noun usage not only reduces the volume of the dataset but also improves the accuracy, performance and effectiveness of a feature location process.

5. Threats to validity

Presented in this section are some of the threats to the validity of this study, specifically threats to the internal validity and the external validity.

5.1. 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 named, and that the developers adhered to good programming practices when naming variables, methods, and classes. In this context, if project developers were to use meaningless names, the effectiveness of NBFL 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 both the content of the fixed change requests and the content of the new change request for which source code locations were sought. If either of the content is of low quality, meaning that the Summary and Description are not well-defined, then the NBFL 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 NBFL would be adversely affected. To reduce the effect of this problem, source code identifiers were used as another data-source.
Third, recall that the POS tagger component of ANNIE plug-in was used for term categorization. However, this component was trained using data from the Wall Street Journal, a domain not 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 the POS tagger component in term categorization, especially noun determination, could potentially influence the results of the NBFL 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.

5.2. External validity

External validity is concerned with whether or not the results of the evaluation can be generalized to other datasets besides those datasets used in the study. First, all of 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 NBFL 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 of the factors determined for selecting the most suitable subject systems. All of the subject systems were written in the Java programming language, and they were all software applications that support software development. This means that all of 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 by using projects from a different domain of software projects, such as systems for healthcare, transportation, or e-commerce. The evaluation of systems from these 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 was minimized by the use of the GATE tool that treated the 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 is that “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. [29] nor is it as low as that used by Poshvyanyk [49,3]. It is believed that this data-set size provides a good compromise between those used by other works.

6. Related work

Feature location, as one of the MSR tasks, is an active research area that has attracted a lot of research effort and has resulted in the proposal of many automatic and semi-automatic approaches. The most recent survey in this area [9] reviewed 89 articles from 25 venues and classified feature location approaches into three categories: dynamic, static, and 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 [51]. Unlike dynamic approaches, static analysis requires neither a working software system, nor a test case that exercises the feature. Instead, static analysis deals with structural information such as control or data flow dependencies [52,42]. Finally, text analysis investigates the text data stored in the historical project repositories and analyzes them to extract useful information [53,10]. The proposed feature location approach that uses a new term-weighting technique falls into this latter category. Consequently, this section focuses on literature that analyzes text data and uses term-weighting techniques. Feature location literature that uses text data is divided into three main areas [9] as follows:

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

Like the proposed approach, all of these methods are query based. Pattern matching usually involves a text-based technique to search the source code using a utility such as regular expression matching tools, grep [10,11] being an example. This technique could be considered the simplest text analysis feature location method. Information retrieval techniques are based on statistical methods. IR analyzes and retrieves documents of a corpus similar to a query [54,53,55,42,12]. This is similar to the process used in the proposed approach. Feature location approaches that use IR methods typically use all of the terms in the document after common preprocessing steps, whereas the proposed approach uses only the noun terms, thus reducing the volume of data and summarizing it. IR considers the text resources as a collection of terms that co-occur frequently in the documents of the corpus [27]. As mentioned in Section 1, text data in the software repositories has additional data not found in simple text documents that can be considered, such as the evolution of the text. Unlike IR, the proposed approach and term-weighting technique leverage this additional information. Natural Language Processing, the last type of text analysis feature location approach, considers the grammatical category of a term in the text to analyze and extract the required information from the documents [14,15]. In the proposed approach, similar to NLP, the category of terms is considered in the analysis of the content of the text resources. Investigation of the feature location literature reveals that IR is the most commonly used text analysis method for feature location [9].

As mentioned in Section 3.1, the IR method involves a set of models, such as SUM [28], LDA [42], LSI [12], and VSM [28], to retrieve and index the documents in the corpus. The algebraic IR models, such as VSM and LSI, weight and index the terms in the documents based on similar terms for a given query. Common term-weighting techniques include Binary, Term Count, Term Frequency, and Term Frequency-Inverse Document Frequency (TF-IDF) [1]. The Binary term-weighting technique assigns a weight of one to the terms that appear in the document and are common with the query, and zero for terms that are not in common. In the Term Count weighting technique, the weight of each term is based on the number of times it appears in the document, whereas in the Term Frequency weighting, this count is normalized to prevent bias toward longer documents. Finally, in the TF-IDF weighting technique, the weight of each term increases proportionally by the number of times it appears in the document, but is offset by its frequency across the corpus. TF-IDF is the most commonly used term-weighting technique for IR models [13]. For all of these

S. Zamani et al. / Information and Software Technology 56 (2014) 991–1011
term-weighting techniques, the main factor is the number of appearances or the number of repetitions of the term in the document. In this work, in addition to the term frequency, the time-metadata associated with the terms is considered in the weighting process.

In addition to feature location, time decay has been considered in the area of fault prediction. Sisman and Kak [20] applied the findings from the fault prediction area to bug localization. Unlike this work, they utilized time decay in weighting the files in a probabilistic IR model, not an algebraic model. However, the consideration of time-metadata at the level of files misses the value of the terms that exist in the file as they evolve over time. In the context of term-weighting for feature location, Bassett and Kraft [17] considered the structural information in weighting the terms of the document without taking into account the time decay. 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. In contrast, our work focuses on weighting the terms, based on their value over time, whereas Bassett and his colleagues focused on the probabilistic distribution of a term and the term’s importance within its associated document.

As mentioned in Section 1, the use of only noun terms has been done by Capobianco et al. [24] in the area of traceability recovery. Traceability recovery research area is closely related to feature location [9]. Traceability is the ability to describe and follow the life of software repositories [37]. It aims to connect different types of software repositories, while feature location is more concerned with identifying the source code file associated with a software feature, not specific parts of a document. Many traceability link recovery methods have been proposed to automate the link recovering process [56–59], with IR being one of the most used to facilitate the tracing process [60,16].

7. Conclusion and future work

Feature location is a significant software maintenance activity. In this paper, a text categorization approach to feature location was presented that used only noun terms and employed a novel time-aware term-weighting technique. The empirical evaluation of the proposed approach on four open-source projects of varying sizes and evolution speeds, showed that the approach is more accurate than the commonly used VSM approach in the range of 13–50% when presenting a ranked list of ten source code location recommendations. This comparison reveals up to a 17% effective- ness improvement and a 13% performance improvement. Additionally, the use of the time-aware term-weighting technique was compared to the use of the standard TF-IDF technique and it was found that the proposed technique improved the accuracy up to 15%, the effectiveness up to 10%, and performance up to 40%. It was also found that the use of only noun terms in the proposed approach resulted in improvements to the accuracy, effectiveness, and performance by as much as 28%, 21%, and 58%, respectively. Finally, it was shown that by using only the noun terms and time-aware term-weighting, not only is the accuracy, performance and effectiveness of a feature location approach improved, but it also avoids the need for some general text analysis activities such as dimensionality reduction and threshold determination.

There are a number of additional directions that can be explored in the future: First, the application of time-metadata in term-weighting techniques used by feature location approaches such as TF-IDF. Second, the use of other metadata, such as developer information, found in software repositories for feature location. Also, this work evaluated NBFL on software projects that are predominantly in the domain of software development tools. A further evaluation of NBFL using projects from other domains is needed to demonstrate the general applicability of the approach. Finally, the NBFL approach needs to be further validated by a field study of software developers using NBFL in a real-world setting.

Acknowledgments

This work was carried out within the framework of a research project supported by the High Impact Research Grant with reference UM.C/625/1/HIR/MOHE/FSCT17/13, funded by the Ministry of Education, Malaysia. We would like to thank Reza M. Parizi for his helpful comments on improving the earlier versions of this paper; and the TraceLab Team for their support in using their framework.

References

