**Noname manuscript No.**
(will be inserted by the editor)

# Fusing Multi-Abstraction Vector Space Models for Concern Localization

**Yun Zhang · David Lo · Xin Xia · Giuseppe Scanniello · Tien-Duy B. Le · Jianling Sun**

Received: date / Accepted: date

**Abstract** Concern localization refers to the process of locating code units that match a particular textual description. It takes as input textual documents such as bug reports and feature requests and outputs a list of candidate code units that are relevant to the bug reports or feature requests. Many information retrieval (IR) based concern localization techniques have been proposed in the literature. These techniques typically represent code units and textual descriptions as a bag of tokens at *one level of abstraction*, e.g., each token is a word, or each token is a topic. In this work, we propose a *multi-abstraction* concern localization technique named MULAB. MULAB represents a code unit and a textual description at multiple abstraction levels. Similarity of a textual description and a code unit is now made by considering all these abstraction levels. We combine a vector space model (VSM) and multiple topic models to compute the similarity and apply a genetic algorithm to infer semi-optimal topic model configurations. We also propose 12 variants of MULAB by using different data fusion methods. We have evaluated our solution on 175 concerns from 9 open source Java software systems. The experimental results show that variant COMBMNZ-DEF performs better than other variants, and also outperforms the state-of-art baseline called PR (PageRank based algorithm), which is proposed by Scanniello et al (2015) in terms of effectiveness and rank.

Yun Zhang, Jianling Sun
College of Computer Science and Technology, Zhejiang University, China
E-mail: yunzhang28@zju.edu.cn, sunjl@zju.edu.cn

David Lo, Tien-Duy B. Le
School of Information Systems, Singapore Management University, Singapore
E-mail: davidlo@smu.edu.sg, btdle.2012@smu.edu.sg

Xin Xia
Faculty of Information Technology, Monash University, Australia
College of Computer Science and Technology, Zhejiang University, China
E-mail: xxia@zju.edu.cn, xin.xia@monash.edu

Giuseppe Scanniello
University of Basilicata, Potenza, Italy
E-mail: giuseppe.scanniello@unibas.it

## 1 Introduction

Developers receive bug reports and feature requests through issue management systems such as Bugzilla and JIRA daily. The amount of these reports are often too many for developers to handle (Anvik et al, 2005). For each of these reports and requests, developers need to locate the code units that need to be modified to fix bugs or be extended to implement a particular feature. Considering a large code base with thousands or even millions of files, this task is a daunting one. Much manual effort needs to be spent to locate relevant code units. Thus, an automated solution is needed.

Concern localization is a software maintenance process of locating code units that need to be changed in response to a modification request, such as bug fixing or a new feature request. Change requests are usually formulated in natural language, describing the problems or the solutions of the software system, while the source code also includes large amounts of text such as comments and identifiers.

Recently, a number of approaches have been proposed to link bug reports and feature requests to the corresponding code units, (e. g., Le et al (2015); Marcus and Maletic (2003); Wang and Lo (2014); Wang et al (2011b, 2014); Xia et al (2014); Zhou et al (2012)). The bug reports and feature requests could be viewed as *concerns*,[1] and the linking process of code units to concerns is referred to as *concern localization*. Many past studies on bug localization, feature location, etc. could be viewed as specific instances of concern localization.

Many existing studies characterize both concerns (e.g., feature requests or bug reports) and code units as a bag (i.e., multi-set) of tokens at *one abstraction level*, e.g., (Marcus and Maletic, 2003; Wang et al, 2011b). A textual document (i.e., feature request, bug report, or code unit) is represented as a set of words that appear in it. Alternatively, a natural language processing technique referred to as topic modeling (e.g., Blei et al (2003)) can be applied to infer a set of topics that appear in the document. A topic is a distribution of words and is a higher level abstraction of the original words. A set of topics can be inferred from documents and these topics would represent these documents. Similarities of documents can then be measured as the similarities of their representations (i.e., their set of words or topics). The code units that are most similar to the input concerns are output to the end user.

Recently, Scanniello et al. propose a static concern localization approach named PR which combines textual and structural information together (Scanniello et al, 2015). PR extracts dependency among methods in a code base (based on direct references between methods) and uses the PageRank algorithm (Kleinberg et al, 1999) to rank methods based on their importance. The similarities between a concern and a code unit (i.e., a method) is then measured by multiplying the textual similarity computed by comparing the concern and the code unit using vector space modeling (VSM) (Salton and Harman, 2003) and the importance of the code unit estimated using PageRank. The experiment results show that their approach leads to better retrieval performance than several baseline approaches: one that uses textual information only and one that combines textual and structural information on clustering (Scanniello

[1] A concern is a concept, requirement, feature, or property related to a software system (Robillard and Murphy, 2007). In this work, we focus on bug reports and feature requests which are subsets of concerns, but the proposed approach could be used for generic concerns.

and Marcus, 2011). In this paper, PR is a state-of-the-art baseline approach that we compare our proposed approach with.

While many past studies only compare two documents at one abstraction level, in this work, we compare documents at multiple abstraction levels. A word can be abstracted at multiple levels of abstraction. For example, "AVL tree" in a document can be abstracted to "binary tree", "tree", "graph", "data structure", and so on. Two documents might not share "AVL tree" but they might both related to "binary tree", "tree", "graph", or "data structure", and so on. By viewing a document at multiple levels of abstractions the similarity or difference of two documents can be better assessed.

To represent documents in multiple abstraction levels, we leverage topic modeling. Topic modeling (Blei and Lafferty, 2007) maps words that appear in a document to topics. Each word is assigned to one topic. The fewer the number of topics, the higher the abstraction level. This is the case as a topic now represents more words. On the other hand, the larger the number of topics, the lower the abstraction level. Thus, we can iteratively apply topic modeling using different numbers of topics to create multiple abstraction levels. We can then aggregate these abstractions to measure the similarity between a concern (e.g., a bug report or a feature request) and a code unit. We apply an adaptive Latent Dirichlet Allocation (LDA) with Genetic Algorithm (GA) (Panichella et al, 2013) to determine a near-optimal configuration for LDA to tune the topic number of each abstraction level. Topic models have recently been used in software textual retrieval and analysis to support software engineering tasks. However, previous research show that applying topic models on software artifacts using the same settings as for natural language text did not always produce the expected results, as text extracted from source code is much more repetitive and predictable as compared to natural language text (Arcuri and Fraser, 2011; Hindle et al, 2012; Oliveto et al, 2010; Panichella et al, 2013). So we need GA to determine the near-optimal topic number for LDA when used in software engineering tasks. By using multiple abstraction levels, the impact of the parameters of LDA and GA will be reduced.

In the literature, VSM has been shown to outperform many other information retrieval (IR)-based techniques for concern localization (Rao and Kak, 2011; Wang et al, 2011b). In this paper, we propose a new approach which combines VSM and multiple abstraction levels; the multiple abstraction levels are built by LDA (tuned using genetic algorithms), which is described in detail in Section 3.5. We refer to the resultant model as MULti-ABstraction VSM (MULAB). We evaluate MULAB on 9 open-source software systems using information from 175 past change requests which map to a total of 501 changed methods. To demonstrate that the proposed multi-abstraction concept works, we propose 12 variants of MULAB by using different data fusion methods (Lucia et al, 2014; Wu, 2012) and choose the best one to compared with PR, the state-of-the-art proposed by Scanniello et al (2015). We address the following research questions.

1. **RQ 1:** What is the best variant and parameter setting of MULAB?
   We find that when evaluated by both effectiveness and rank, COMBMNZ-DEF (CombMNZ fusion method with normalization) performs the best among the 12

variants of MULAB. Then we investigate the best parameter (the height of the abstraction hierarchy). Height L=6 performs better for CombMNZ-Def than other settings of L. i.e., L=1, L=2, L=3, L=4, and L=5. But considering both performance and time efficiency, height L=4 is the best choice for our experiment.

2. **RQ 2:** How much improvement could the best performing variant achieve over its components (i.e., VSM and topic models)?
   We find that when evaluated by effectiveness and rank, CombMNZ-Def performs better than its 5 components (VSM model and 4 abstraction levels).
3. **RQ 3:** How effective is CombMNZ-Def compared with state-of-the-art approach?
   We find that CombMNZ-Def outperforms PR on 7 among the 9 Java systems when evaluated in terms of effectiveness and rank. Statistical tests show that the differences are statistically significant and substantial.
4. **RQ 4:** What is the effect of varying the text used to represent a concern on CombMNZ-Def's effectiveness?
   We find that CombMNZ-Def with default configuration (which uses text from both summary and description fields to represent a concern) performs better than when only text from summary and text from description are used independently, in terms of both effectiveness and rank scores.

This paper extends our preliminary study which appears as an research paper of ICSME 2016 (Zhang et al, 2016). In particular, we extend our preliminary work in several directions: *(i)* In addition to following our previous method which combines VSM model and multi abstraction levels by putting them into one vector, we propose 11 variants that perform the combination using 6 data fusion methods; *(ii)* We strengthen the experimental part by adding a data set eclipse and investigating two additional research questions; these questions investigate: (1) the performance of the 12 methods (our original method and the 11 variants) to choose the best one, (2) if the best variant performs better than its components; *(iii)* After getting the best performing variant, we have repeated experiments to answer the three research questions appearing in our ICSME 2016 paper using the best performing variant. We have compared the effectiveness of the best performing variant against a recently proposed state-of-the-art approach (Scanniello et al, 2015).

Our contributions, which form a super-set of those of our preliminary study, are as follows:

1. We propose multi-abstraction concern localization. We represent a document (i.e., a code unit, bug report, or feature request) at multiple abstraction levels.
2. We propose a technique MULAB that leverages multiple topic models to capture representations of documents at different abstraction levels. MULAB employs an adaptive LDA with genetic algorithm (LDA-GA) to tune the topic numbers of each abstraction level. MULAB then uses these representations to compute the similarity between a concern and a code unit.
3. We propose 12 variants of MULAB by using different data fusion methods.
4. We have evaluated the 12 variants of MULAB on 175 concerns from 9 Java software systems, and choose a best performing variant. Results show that our best variant of multi-abstraction approach outperforms PR, which is a state-of-the-art

approach proposed by Scanniello et al. (Scanniello et al, 2015), by a substantial margin.

**Paper structure**. In Section 2, we briefly introduce LDA and GA. In Section 3, we present the details of MULAB. We present our experimental results in Section 4. We review related work in Section 6. We conclude and mention future work in Section 7.

## 2 Preliminaries

### 2.1 Latent Dirichlet Allocation

A topic model (Blei and Lafferty, 2007) views a document to be a probability distribution of topics, while a topic is a probability distribution of words. In our setting, a document is a program method in the code base or a concern, and a topic is a higher-level concept corresponding to a distribution of words. For example, we can have a topic "Java Programming" which is a distribution of words such as "variable", "inheritance", "class", "method", and so on.

Latent Dirichlet Allocation (LDA) is a well-known topic modeling technique proposed by Blei et al. (Blei et al, 2003), which has been widely used in software engineering (Asuncion et al, 2010; Panichella et al, 2013; Thomas, 2011; Xia et al, 2017). LDA takes a document-by-term matrix $D$ as input, and outputs two matrices $DT$ and $TT$, i.e., a document-by-topic matrix and a topic-by-term matrix. The document-by-term matrix $D$ is a term frequency matrix, in which $D_{ij}$ represents the number of times that the $j$-th term (i.e., word) appears in the $i$-th document. In the document-by-topic matrix $DT$, $DT_{ij}$ represents the probability of the $i$-th document to belong to the $j$-th topic. Generally, a document is considered to belong to the topic with the highest probability. In the topic-by-term matrix $TT$, $TT_{ij}$ represents the probability that the $j$-th term belongs to the $i$-th topic. Likewise, we assign a term to the topic with the highest probability and then we can conclude what a topic is about by looking up the terms it contains. After training, LDA can be used to infer the topic distribution of a new document (in our case: a new concern). LDA takes several parameters: the number of topics ($K$), and two hyper-parameters $\alpha$ and $\beta$. While the hyper-parameters are typically set to be $50/K$ and $0.01$ respectively following the suggestions by Blei et al. (Blei et al, 2003), the values of $K$ needs to be carefully tuned.

There are several implementations for LDA in the literature. In our work, we use an implementation based on collapsed Gibbs sampling. This approach typically achieves the same accuracy as the standard LDA implementation while being faster in its execution (Griffiths and Steyvers, 2004; Wallach et al, 2009). Besides the three parameters, $K$, $\alpha$, and $\beta$ introduced above, our Gibbs sampling implementation takes an additional parameter $m$ which specifies the number of Gibbs sampling iterations. By default, we set $m$ to be 1,000.

## 2.2 Genetic Algorithms

A genetic algorithm (GA) is a stochastic search technique that mimics the process of natural selection. Since its first introduction by Holland (Holland, 1975) in the 1970s, genetic algorithms have been widely used to generate solutions to optimization problems using techniques such as mutation, selection, and crossover. The advantage of GA with respect to other search algorithms is its intrinsic parallelism, i.e., having multiple solutions evolving in parallel to explore different parts of the search space.

The GA search starts with a population of randomly generated individuals, where each individual (i.e., a chromosome) represents a random parameter configuration of the optimization problem. Generally, the evolution of the whole population is an iterative process, in which each iteration is called a generation. In particular, the population evolves through subsequent generations and, during each generation, the individuals are evaluated based on a fitness function that has to be optimized. The fitness function is used to evaluate the different parameter configurations by generating different fitness values. For creating the next generation, new individuals (i.e., offsprings) are generated by: (1) applying a selection operator, which randomly picks individuals based on the fitness function (individuals with higher fitness values are more likely to be selected), (2) recombining, with a given probability, two individuals from the current generation using the crossover operator, and (3) modifying, with a given probability, individuals using the mutation operator. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations have been produced, or a satisfactory fitness level has been reached for the population. More details about GA can be found in a book by Goldberg (Goldberg, 1989), and we will show how we apply it in our algorithm in Section 3.3.

## 2.3 Score Normalization

In this work, we apply Zero-One score normalization (Wu, 2012) before data fusion. This method transforms scores from different abstraction levels into the same range i.e., zero to one. The method works as follows. Let $m$ be the total number of methods for each project, and $s_i$ denotes the score of the $i^{th}$ method, where $1 \leq i \leq m$. Furthermore, let $max_s$ denotes the maximum score among all the methods, and $min_s$ denotes the minimum score among all the methods. The normalized score of the $i^{th}$ method is calculated as follows:

$$s\_norm_i = \frac{s_i - min_s}{max_s - min_s}$$

## 2.4 Data Fusion

Data fusion methods proposed in the information retrieval community are used to integrate normalized scores from different techniques. The goal of data fusion is to combine relevant information from two or more data sources into a single one that

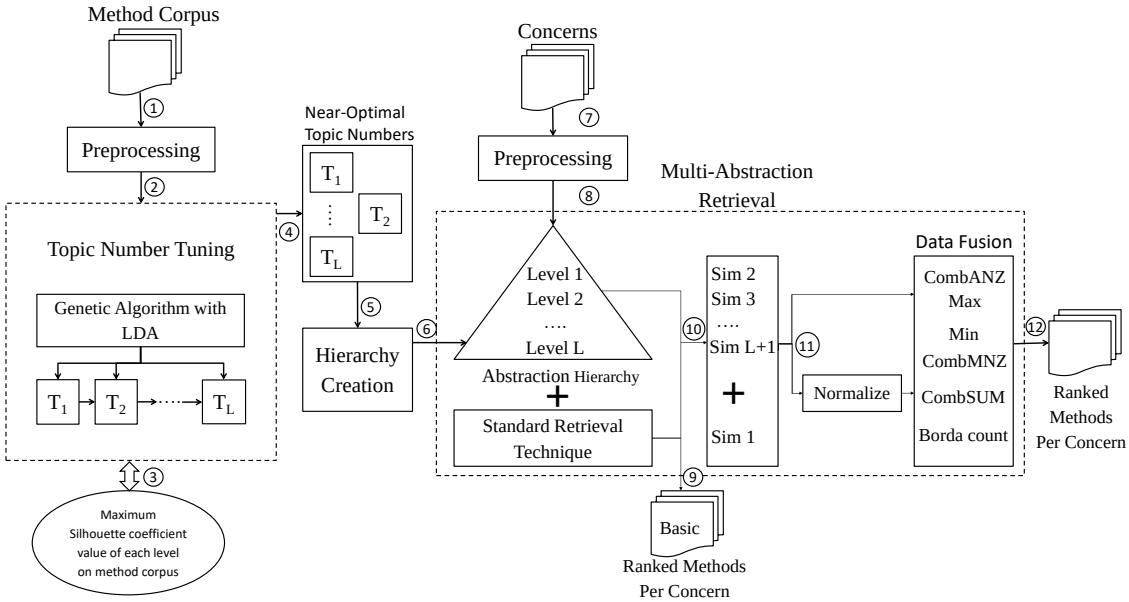



**Fig. 1** Overall Framework of MULAB

provides a more accurate performance than any of the individual data sources. In this work, we leverage six well-known unsupervised data fusion methods in the domain of information retrieval, namely CombANZ (Fox et al, 1992; Joseph, 1997; Shaw and Fox, 2014), Borda count (Aslam and Montague, 2001), Max, Min, CombMNZ (Fox et al, 1992; Joseph, 1997; Shaw and Fox, 2014), and CombSUM (Fox et al, 1992; Joseph, 1997; Shaw and Fox, 2014). We will elaborate how these fusion methods work in Section 3.

## 3 MULAB

### 3.1 Overview

In Figure 1, we present the overall framework of MULAB. Our framework takes as input *method corpus* and *concerns*. *Method corpus* is a collection of textual documents where each document corresponds to a method in the code base. Each document contains identifiers and words that appear in the source code, documentation (e.g., Javadoc), and implementation comments of the corresponding methods. *Concerns* are a collection of textual documents where each document is either a bug report or a feature request. For each bug report and feature request, we extract the text that appears in its title and description. The output of our framework is a set of ranked methods for each concern.

Our framework contains four processing steps: *preprocessing*, *topic number tuning*, *hierarchy creation*, and *multi-abstraction retrieval*. The purpose of the *prepro-*

*cessing* step is to convert methods and concern documents into a standard representation, i.e., a bag of words. The preprocessed documents (i.e., methods and concerns) are then input to the *topic number tuning* step. The *topic number tuning* step uses a genetic algorithm to determine a near-optimal topic number of LDA for each abstraction level and these are input to the *hierarchy creation* step. The *hierarchy creation* step applies a topic modeling technique a number of times to construct an *abstraction hierarchy*. The *abstraction hierarchy* is a collection of topic models with various settings, where each topic model is a level in the hierarchy. This *abstraction hierarchy* is used by the *multi-abstraction retrieval* step. In this step, we enhance a *standard text retrieval technique* based on VSM by leveraging the *abstraction hierarchy*, then we can choose to use data fusion methods or not. The goal of the final processing step is to compare a concern (a query) and a method (a document in the *method corpus*) by considering multiple abstraction levels. We elaborate the four processing steps in the following subsections.

### 3.2 Preprocessing Step

We first perform *text normalization* by removing common Java keywords (e.g., *public*, *private*, *class*, *extends*, etc.), and English stopwords. These words are deemed useless for retrieving relevant code units (i.e., methods) for concerns as either they appear in most documents or they carry little meaning. We also normalize the text by excluding punctuation marks and special symbols. Thus, we only retain some word tokens and number literals. Furthermore, we break identifiers into smaller tokens following Camel casing convention that is the naming convention adopted by most Java programs. By performing text normalization, we standardize word tokens in *Method Corpus* with those that are used in *Concerns*.

Next, we apply the Porter Stemming Algorithm[2] to reduce English words into their root forms. For example, "models", "modeled", "modeling" are all reduced to the same root word "model". We perform this step to standardize words of the same meaning but are in different forms. At the end of this step, we forward preprocessed method and concern documents to the *topic number tuning* step to determine best settings to infer topic models.

### 3.3 Topic Number Tuning Step

The parameter $K$ of LDA, which is the number of topics, is an important parameter that significantly determines LDA output. An improper value of $K$ for each abstraction level may affect the performance of our approach. Therefore, we use an adaptive LDA technique, leveraging genetic algorithm (GA), to optimize the value of $K$ in each abstraction level. This approach proposed by Panichella et al. is referred to as LDA-GA (Panichella et al, 2013).

[2] http://tartarus.org/martin/PorterStemmer/

At the beginning, a population of $p$ randomly-generated chromosomes is initialized by LDA-GA, where each of chromosome contains a random integer value corresponding to the number of topics. Then, the population will evolve in $n$ generations to search for an optimal value of the number of topics. The population is evolved relying on a fitness function which corresponds to the Silhouette coefficient. The Silhouette coefficient is a common evaluation metric for measuring the goodness of a clustering result (Hotho et al, 2002; Panichella et al, 2013; Rousseeuw and Kaufman, 1990; Sander et al, 1998). In LDA-GA, documents are clustered according to the topics inferred by LDA, where documents assigned to the same topic are grouped in the same cluster. From these clusters of documents, three steps are performed to compute the Silhouette coefficient:

1. **Step 1:** For a document $d_i$, we calculate the maximum distance from $d_i$ to the other documents in the same cluster, which is denoted as a($d_i$). And we calculate the minimum distance from $d_i$ to the centroids of the other clusters not containing $d_i$, which is denoted as b($d_i$).

2. **Step 2:** Given $a(d_i)$ and $b(d_i)$, we can calculate the Silhouette coefficient $s(d_i)$ for the document $d_i$ according to the following formula:

$$s(d_i) = \frac{b(d_i) - a(d_i)}{max\{a(d_i), b(d_i)\}}$$

3. **Step 3:** We compute the mean value of all s($d_i$) as the overall Silhouette coefficient.

The range of the Silhouette coefficient is $[-1, 1]$. A larger value of the Silhouette coefficient indicates a better clustering. When a high Silhouette coefficient is achieved for a particular value of the number-of-topic parameter of LDA, it means that the particular parameter value leads to a good result. The higher Silhouette coefficient is achieved using a particular parameter value, the more likely the parameter value is kept in the genetic algorithm (GA)'s evolution process. For each abstraction level, we perform LDA-GA once to find a suitable number of topics.

The original implementation of LDA-GA is written in R and it runs rather slowly. Thus, we reimplement LDA-GA approach on the top of *Pyevolve*,[3] an evolutionary computation framework. By default, we set $p$ as 100 and $n$ as 50 and *Pyevolve*'s crossover and mutation rate to be 0.09 and 0.02, respectively, as we empirically find that the values of these parameters do not make a big difference to the performance of our approach. For the mutation operator, we use random mutation. For each gene, with a certain probability, it randomly swaps the gene with another double value in the range of zero to one. For the crossover operator, we use the single point crossover operator. This operator processes pairs of chromosomes, for each pair, with a certain probability, it randomly picks a gene from a parent chromosome and swaps that gene and the subsequent ones with corresponding genes from the other parent chromosome. For each abstraction level, we execute LDA-GA to generate an optimal value of number of topics. We set different search ranges for each abstraction level. For

[3] http://pyevolve.sourceforge.net/

example, let us assume that there are $L$ levels in an abstraction hierarchy. For the first level, we set the search range to be integers in the interval $[2, \frac{D}{L}]$ where $D$ refers to the total number of documents in the data set. We set the search range as such since we assume there should be at least 2 and at most $D$ topics (i.e., each document belongs to its own topic). Let us assume that we get an optimal result $t_1$ for this range. For the second level, we set the range to be integers in $[t_1, \frac{2D}{L}]$ and get the optimal number of topics $t_2$. The process repeats for the subsequent levels. Finally, for the $L^{th}$ level, we set the search range in $[t_{L-1}, D]$, and get the best number of topics $t_L$ for this last level. This set of $L$ topic numbers is then output to the hierarchy creation step.

### 3.4 Hierarchy Creation Step

In the hierarchy creation step, we apply LDA a number of times to create the abstraction hierarchies with the number of topics inferred by the *topic number tuning* step. These $L$ abstraction levels form an abstraction hierarchy $H$. Topic models with fewer topics are higher in the hierarchy while those with more topics are lower in the hierarchy. We refer to the number of topic models contained in a hierarchy as the *height* of the hierarchy. At the end of this step, we create an abstraction hierarchy which is used in the next step: *multi-abstraction retrieval*.

### 3.5 Multi-Abstraction Retrieval

In this subsection, we discuss how to combine an abstraction hierarchy with a text retrieval model (i.e., VSM). A retrieval method takes a query (i.e., a bug report) and returns a sorted list of most similar documents in a corpus (i.e., methods).

In standard VSM, a document is represented as a vector of weights. Each element in a vector corresponds to a word, and its value is the weight of the word. Term frequency-inverse document frequency (*tf−idf*) is often used to assign weights to words (Manning et al, 2008). The following is the *tf−idf* weight of word $w$ in document $d$ given a corpus (i.e., a set of documents) $D$, denoted as *tf−idf*$(w, d, D)$:

$$\textit{tf−idf}\,(w, d, D) = \log(f(w, d) + 1) \times \log \frac{|D|}{|\{d_i \in D | w \in d_i\}|}$$

where $f(w, d)$ is the number of times word $w$ appears in document $d$, and $w \in d_i$ denotes that word $w$ appears in document $d_i$. Given a query document $q$, standard VSM retrieval model would return the most similar documents in the corpus $D$. Similarity between two documents is measured by computing the cosine similarity between the two documents' vector representations (Manning et al, 2008).

To combine the abstraction levels and VSM, the basic method we used is combine the topic distributions of the abstraction levels and standard VSM model into one vector (MULAB$_{basic}$). We also investigate several data fusion methods to identify a better performing strategy. We introduce MULAB$_{basic}$ and data fusion methods in the following paragraphs.

### *3.5.1* MULAB$_{basic}$

In MULAB$_{basic}$, we integrate abstraction hierarchy into standard VSM by extending the vector that represents a document. We added more elements to the vector. Each added element corresponds to a topic of a topic model in the abstraction hierarchy, and its value is the probability of the topic to appear in the document. The size of an extended document vector is $V + \sum_{i=1}^{L} K(H_i)$, where $V$ is the size of the original document vector, $L$ is the number of abstraction levels in the hierarchy, and $K(H_i)$ is the number of topics of the $i^{th}$ topic model in the abstraction hierarchy $H$. Based on this representation, the similarity between a query $q$ and document $d$, considering a corpus $D$, calculated using cosine similarity, is as follows:

$$sim(q, d, D) = \frac{\sum_{i=1}^{V} \textit{tf–idf}\,(w_i, q, D) \times \textit{tf–idf}\,(w_i, d, D) + \sum_{k=1}^{L} \sum_{i=1}^{K(H_k)} \theta_{q,t_i}^{H_k} \times \theta_{d,t_i}^{H_k}}{\|q\| \times \|d\|}$$

where

$$\|q\| = \sqrt{\sum_{i=1}^{V} \textit{tf–idf}\,(w_i, q, D)^2 + \sum_{k=1}^{L} \sum_{i=1}^{K(H_k)} (\theta_{q,t_i}^{H_k})^2}$$

and

$$\|d\| = \sqrt{\sum_{i=1}^{V} \textit{tf–idf}\,(w_i, d, D)^2 + \sum_{k=1}^{L} \sum_{i=1}^{K(H_k)} (\theta_{d,t_i}^{H_k})^2}$$

In the above equations, $\theta_{d,t_i}^{H_k}$ is the probability of topic $t_i$ to appear in the document $d$ as assigned by the $k^{th}$ topic model in the abstraction hierarchy $H$.

For example, assuming that a bug report $br$ after text preprocessing has the following 7 words: *"source"*(3), *"control"*(2), *"activity"*(2), *"reduce"*(2), *"tool"*(1), *"root"*(1), *"list"*(1). We also have two methods $m_1$ and $m_2$. Each of them contains 5 words: $m_1 =\{$*"source"(7), "control"(4), "activity"(3), "root"(7), "list"(1)*$\}$ and $m_2 =\{$*"source"(10), "control"(10), "reduce"(5), "tool"(4), "root"(6)*$\}$. The number in parentheses is the number of times a word appears in a document. Let us assume that an abstraction hierarchy of height 1 is used, and the topic model has 3 topics. Let us also assume that there are 1000 methods, and terms in $m_1$ and $m_2$ do not appear in other methods. Considering only the 7 words, the representative vectors of $br$, $m_1$, and $m_2$ are:

$$V_{br} = [1.62, 1.29, 1.43, 1.43, 0.90, 0.81, 0.90, \textit{0.26}, \textit{0.72}, \textit{0.02}]$$
$$V_{m_1} = [2.44, 1.89, 1.81, 0.00, 0.00, 2.44, 0.90, \textit{0.00}, \textit{0.99}, \textit{0.00}]$$
$$V_{m_2} = [2.81, 2.81, 0.00, 2.33, 2.10, 2.28, 0.00, \textit{0.57}, \textit{0.43}, \textit{0.00}]$$

The first 7 entries in each vector are the weights of the 7 words computed using the *tf–idf* formula, and the last 3 entries are the rounded probabilities $\theta_{d,t_i}^{H_1}$ of topics

**Table 1** An Example of Using Data Fusion Methods.

| Method ID | Sim1(VSM) | Sim2 (Level1) | Sim3 (Level2) |
|---|---|---|---|
| 1 | 0.4 | 0.8 | 0 |
| 2 | 0.6 | 0.1 | 0.7 |
| 3 | 0 | 0.5 | 0.3 |

1, 2 and 3 respectively in the documents. Finally, we calculate cosine similarities between bug report $br$ and methods $m_1$ and $m_2$. The results are $sim(br, m_1) = 0.82$ and $sim(br, m_2) = 0.84$. Thus, $m_2$ is more relevant to bug report $br$ than $m_1$.

### *3.5.2 Data Fusion Methods*

MULAB$_{basic}$ described in the last subsection integrates abstraction levels and standard VSM into one vector to compute the similarity; in this subsection, we try to combine several abstraction levels and standard VSM by using a number of data fusion methods. We investigate six well-known unsupervised data fusion methods: CombANZ (Fox et al, 1992; Joseph, 1997; Shaw and Fox, 2014), Max, Min, CombMNZ (Fox et al, 1992; Joseph, 1997; Shaw and Fox, 2014), CombSUM (Fox et al, 1992; Joseph, 1997; Shaw and Fox, 2014), and Borda count (Aslam and Montague, 2001). These six are classical data fusion methods mentioned in the book Data Fusion in Information Retrieval (Wu, 2012), which is an authoritative reference material on the topic. Among the six, Borda count is a ranking-based method, and while the others are score-based methods.

We use the example shown in Table 1 to illustrate how each method works. Let us assume that there are 3 methods and an abstraction hierarchy of height 2 is used. We calculate the cosine similarity scores between concerns and methods by VSM model vector, level1 topic distribution vector, and level2 topic distribution vector respectively. For each pair of concern and method, we can calculate three scores using VSM and the topic models: Sim1 (similarity calculated using VSM), Sim2 (similarity calculated using level 1 topic model), Sim3 (similarity calculated using level 2 topic model). Next, the following data fusion methods are used to combine the similarity scores. As the ranking lists produced by each of the components are typically different, data fusion methods are used to integrate the scores from different components. Data fusion methods can reduce the weakness of each single component by leveraging the strengths of other components (Binkley and Lawrie, 2014; Xia and Lo, 2017; Xia et al, 2015; Xuan and Monperrus, 2014; Ye et al, 2014). By combining the scores or ranks assigned to methods by different components, the most relevant ones are likely to be ranked higher in the final ranking list.

1. **CombANZ:** This method combines the similarity scores by computing the average of the non-zero scores. Let $m_j$ denotes the $j^{th}$ method and $S_i$ denotes the $i^{th}$ similarity score. The $i^{th}$ similarity score assigned to methods $m_j$ is denoted as $S_i(m_j)$. Suppose there are $n$ set of similarity scores and $n_{e_j}$ denotes the number

**Table 2** Example of Ranks and Ranking Points Given by Sim1 (VSM), Sim2 (Level1), and Sim3 (Level2).

| Method ID | Ranks | Ranking Points |
|---|---|---|
| 1 | 2, 1, 3 | 1, 2, 0 |
| 2 | 1, 3, 1 | 2, 0, 2 |
| 3 | 3, 2, 2 | 0, 1, 1 |

of non-zero scores assigned to $m_j$, CombANZ calculates the new score for $m_j$ as follows:

$$Score(m_j) = 1/n_{e_j} \times \sum_{i=1}^{n} S_i(m_j)$$

**Example.** Based on Table 1, the set of new similarity scores of Method 1 to 3 would be $\{\frac{1.2}{2}, \frac{1.4}{3}, \frac{0.8}{2}\} = \{0.6, 0.47, 0.4\}$.
Before using CombANZ, we can choose to do Zero-One score normalization or not, then we have two variants based on CombANZ: COMBANZ-DEF (normalization), COMBANZ-NO (nonnormalized).

2. **Max:** This method combines the similarity score sets by selecting the maximum one as the final score of each method in project.
**Example.** Based on Table 1, the set of new similarity scores of Method 1 to 3 would be $\{0.8, 0.7, 0.5\}$.
Before using Max, we can choose to do Zero-One score normalization or not, then we have two variants based on Max: MAX-DEF (normalization), MAX-NO (nonnormalized).

3. **Min:** Min combines the similarity score sets by selecting the minimum one as the final score of each method in project.
**Example.** Based on Table 1, the set of new similarity scores of Method 1 to 3 would be $\{0, 0.1, 0\}$.
Before using Min, we can choose to do Zero-One score normalization or not, then we have two variants based on Min: MIN-DEF (normalization), MIN-NO (nonnormalized).

4. **CombMNZ:** CombMNZ combines the similarity scores by multiplying the summation of all scores for a given method with the number of non-zero scores assigned to the method. Let $m_j$ denotes the $j^{th}$ method and $S_i$ denotes the $i^{th}$ similarity score. The $i^{th}$ similarity score assigned to methods $m_j$ is denoted as $S_i(m_j)$. Suppose there are $n$ set of similarity scores and $n_{e_j}$ denotes the number of non-zero scores assigned to $m_j$, CombMNZ calculates the new score for $m_j$ as follows:

$$Score(m_j) = n_{e_j} \times \sum_{i=1}^{n} S_i(m_j)$$

**Example.** Based on Table 1, the set of new similarity scores of Method 1 to 3 would be $\{1.2 \times 2, 1.4 \times 3, 0.8 \times 2\} = \{2.4, 4.2, 1.6\}$.

Before using CombMNZ, we can choose to do Zero-One score normalization or not, then we have two variants based on CombMNZ: COMBMNZ-DEF (normalization), COMBMNZ-NO (nonnormalized).

5. **CombSUM:** This method combines different similarity score sets by simply summing up their scores. This method assumes that the similarity scores produced by VSM and abstraction hierarchy are equally important.
   **Example.** Based on Table 1, the set of new similarity scores of Method 1 to 3 would be $\{1.2, 1.4, 0.8\}$.
   Before using CombSUM, we can choose to do Zero-One score normalization or not, then we have two variants based on CombSUM: COMBSUM-DEF (normalization), COMBSUM-NO (nonnormalized).

6. **Borda count:** Borda count converts the similarity scores that are assigned to each method by VSM and abstraction hierarchy into ranks – methods with higher scores would obtain smaller ranks. For each method, Borda count sums up the *ranking points* of a method computed using VSM and topic models. The ranking point of a method is defined as the substraction of the method's rank in the list from the total number of methods in the project.
   Let $m_j$ denotes the $j^{th}$ method and $r_i(m_j)$ denotes the rank of method $m_j$ produced by $i^{th}$ similarity score set. Also, let $M$ denotes the number of methods and $n$ denotes the number similarity score sets. Borda count calculates the new score for program element $m_j$ as follows:

$$Score(m_j) = \sum_{i=1}^{n}(M - r_i(m_j))$$

   **Example.** Table 2 shows the ranking points for each method in Table 1 given by VSM and abstraction hierarchy. Based on the summation of their ranking points, the set of new scores for Method 1 to 3 would be $\{3, 4, 2\}$.
   When using Borda count, whether we do Zero-One score normalization or not, the calculated *ranking points* are the same, so we only have one variant: BORDA.

## 4 Experiment and Analyses

In this section, we evaluate the effectiveness of the 12 variants of MULAB and compare it with other approaches.

### 4.1 Dataset

We use datasets from 9 open source Java software systems (including two versions of one of these systems, namely jEdit) for our experimentation. In the datasets, there are totally 175 concerns which map to 501 methods. The Java systems are the same as

**Table 3** Dataset

| System | Version | Classes | Methods | Concerns | Changed Methods | Description |
|---|---|---|---|---|---|---|
| Art of Illusion (www.artofillusion.org) | 2.4.1 | 453 | 6,229 | 8 | 12 | A free, open source 3D modeling and rendering studio |
| aTunes (www.atunes.org) | 1.10 | 419 | 3,712 | 16 | 30 | A full featured audio player and manager. |
| jEdit (www.jedit.org) | 4.2 | 411 | 5,384 | 16 | 33 | A text editor for programming with an extensible plug-in architecture. |
| jEdit (www.jedit.org) | 4.3 | 492 | 7,095 | 4 | 9 | A text editor for programming with an extensible plug-in architecture. |
| Cocoon (cocoon.apache.org) | 2.2 | 833 | 5,612 | 14 | 38 | A spring-based framework built on separation of concerns and component-based development. |
| Derby (db.apache.org/) | 10.7.1.1 | 3,418 | 40,278 | 29 | 80 | A pure Java relational database engine of using standard SQL and JDBC as its APIs. |
| Lucene (lucene.apache.org) | 4.0 | 5,199 | 24,682 | 24 | 112 | A full-featured text search engine library. |
| OpenJPA (openjpa.apache.org) | 2.0.1 | 4,765 | 41,474 | 25 | 74 | An open source implementation of the Java Persistence API specification. |
| Eclipse (www.eclipse.org) | 3.5 | 18,956 | 218,295 | 39 | 113 | A development platform for building, deploying, and managing software. |

those used by Scanniello et al. (Scanniello et al, 2015). In our experiment, the content of a concern is the textual description retrieved from the title and description of a bug report or a change request.

Each Java method is treated as a document, and all of the Java methods form a corpus. Table 3 shows the statistics of the data sets used in the experiment after preprocessing. The first column shows the names of the software systems and the URLs of their official web pages. The analyzed version of each system and the number of classes are reported in the second and third columns, respectively. The total number of methods in each system is shown in the fourth column, while the fifth column presents the number of concerns used in the study. The number of relevant methods is shown in the sixth column. A short description of each system is presented in the last column.

## 4.2 Evaluation Metrics

Concern localization takes a bug report and a collection of methods as input, and returns a ranked list of these methods. We use two performance metrics to evaluate a concern localization solution: *effectiveness* and *rank*, which are commonly used for concern localization studies (Gay et al, 2009; Poshyvanyk et al, 2007; Scanniello and Marcus, 2011; Scanniello et al, 2015) and used to evaluate our baseline approach *PR*.

*Effectiveness* refers to the position of the first relevant method in the returned ranked list. Once such a method is reached, developers can determine what other

methods need to be changed by analyzing the relationships between the methods. *Rank* refers to the positions of all the relevant methods in the returned ranked list. For each data set, we report the positions of all the relevant methods of all the concerns when rank is used as a yardstick. *Effectiveness* and *rank* nicely complement each other; in fact, effectiveness gives us a best case scenario when an ideal user is performing a concern localization task. Conversely, rank indicates the total effort needed to identify all relevant methods for a given concern by following the ranked list (i.e., a worst case scenario). The lower the effectiveness and rank values, the better a concern localization technique is.

### 4.3 Research Questions

**Research Question 1: What is the best variant and parameter setting of** MULAB**?**

**Motivation.** In this research question, we want to investigate the effectiveness of the 12 variants of MULAB: MULAB$_{basic}$, COMBANZ-NO, COMBANZ-DEF, BORDA, MAX-NO, MAX-DEF, MIN-NO, MIN-DEF, COMBMNZ-NO, COMBMNZ-DEF, COMBSUM-NO, COMBSUM-DEF, and choose the best performing variant. Then we also investigate the best parameter considering a range of model height for the best performing variant.

**Approach.** To answer this research question, we report the results obtained by applying the 12 variants of MULAB to our dataset mentioned in Section 4.1. The 12 variants of MULAB takes in one parameter $L$ which is the height of the abstraction hierarchy. First, we set $L$ to be 4, and compute the effectiveness and rank scores of the 12 variants for each concern and calculate the number of concerns for which each of the approach outperforms (or achieves the same scores as) the others. Second, we conduct an experiment with six different hierarchy heights (i.e., $L$ = 1, 2, 3, 4, 5 and 6). We then compare the results achieved by the best performing variant using these different hierarchy heights in terms of effectiveness and rank scores.

To check if the differences in the performance of two approach are statistically significant, we apply the Wilcoxon signed-rank test (Wilcoxon, 1945) at 95% significance level on two paired data of all the 175 concerns. We also use Cliffs delta ($\delta$) (Cliff, 2014), which is a non-parametric effect size measure that quantifies the amount of difference between two approaches. The delta values range from -1 to 1, where $\delta = -1$ or 1 indicates the absence of overlap between two approaches (i.e., all values of one group are higher than the values of the other group, and vice versa), while $\delta = 0$ indicates the two approaches are completely overlapping. Table 4 describes the meaning of different Cliffs delta values and their corresponding interpretation (Cliff, 2014).

**Results.** Tables 5 presents the effectiveness and rank results of the 12 variants of MULAB, we report the overall results of the 9 datasets here, and the detailed results will show in the Appendix A. For each variant, we report the number of wins, loses, and draws for all the 9 Java systems. Wins, loses, and draws represent the number

**Table 4** Cliffs Delta and the Effectiveness Level (Cliff, 2014)

| **Cliffs Delta($\|\delta\|$)** | **Effectiveness Level** |
|---|---|
| $\|\delta\| < 0.147$ | Negligible |
| $0.147 \leq \|\delta\| < 0.33$ | Small |
| $0.33 \leq \|\delta\| < 0.474$ | Medium |
| $\|\delta\| \geq 0.474$ | Large |

of concerns[4] or methods[5] for which a variant performs the best, performs worse than another, and performs as well as the others, respectively. "Draws" means the 12 variants achieve the same effectiveness/rank score for the concern/method, and there is no best variant on this concern/method.

From the 12 tables, we can see that among the 175 concerns, MULAB$_{basic}$ performs the best on 37 concerns, and 4 variants perform better than MULAB$_{basic}$ in terms of effectiveness: MAX-DEF, MIN-NO, COMBMNZ-NO, and COMBMNZ-DEF, 1 variant performs as well as MULAB$_{basic}$: COMBSUM-DEF. MIN-NO performs the best among the 12 variants, which wins on 45 concerns. Among the 501 methods, when evaluated by rank, MULAB$_{basic}$ wins on 104 methods, and 2 variants perform better than MULAB$_{basic}$: MAX-DEF and COMBMNZ-DEF. COMBMNZ-DEF performs the best among the 12 variants, which wins on 118 methods. When considering both effectiveness and rank, MAX-DEF and COMBMNZ-DEF perform better than MULAB$_{basic}$, and COMBMNZ-DEF performs better than MAX-DEF. Wilcoxon sign-rank test shows that the difference in the effectiveness scores of COMBMNZ-DEF compared with other variants are statistically significant at p-value of $< 0.05$, and the difference in the rank scores of COMBMNZ-DEF compared with other variants are statistically significant at p-value of $< 0.05$ except MULAB$_{basic}$ and MAX-DEF.

Then, we conduct an experiment on COMBMNZ-DEF with six different hierarchy heights (i.e., $L$ = 1, 2, 3, 4, 5 and 6). The experiment results are shown in Table 6(a), 6(b) and Table 7(a), 7(b). For each hierarchy height, we report the number of wins, loses, and draws for each Java system. Wins, loses, and draws represent the number of concerns[6] or methods[7] for which a variant of COMBMNZ-DEF (with a given hierarchy height) outperforms the other variants, loses to another variant, and perform equally well as the other variants, respectively. We also report the overall results in the last row.

Table 6(a) and 6(b) shows the data analysis results on effectiveness scores of COMBMNZ-DEF with different hierarchy heights (L=1,2,3,4,5,6). From the table, we can see that the variant of COMBMNZ-DEF with L set to 6 outperforms the others. Among the 175 concerns, COMBMNZ-DEF with L=6 performs the best on 53 concerns, COMBMNZ-DEF with L=5 performs the best on 45 concerns, COMBMNZ-DEF with L=4 performs the best on 48 concerns, COMBMNZ-DEF with L=3 per-

[4] When effectiveness is used as a yardstick

[5] When rank is used as a yardstick

[6] When effectiveness is used as a yardstick

[7] When rank is used as a yardstick

**Table 5** Overall data analysis results of 9 datasets on effectiveness and rank scores of 12 variants of MULAB. #Wins = Number of concerns and methods for which a variant outperforms the others, #Loses = Number of concerns and methods for which a variant loses from another variant, #Draws = Number of concerns and methods for which all variants perform equally well. We also report the p-value of CombMNZ-Def compared with other variants, thus there's no values for CombMNZ-Def when compared with itself.

| | Effectiveness | | | Rank | | | P-value | |
|---|---|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** | **Effectiveness** | **Rank** |
| $\text{MULAB}_{basic}$ | 37 | 137 | 1 | 104 | 393 | 4 | 0.020 | 0.319 |
| CombANZ-No | 27 | 147 | 1 | 68 | 429 | 4 | 6.814e-07 | 2.351e-07 |
| CombANZ-Def | 29 | 145 | 1 | 89 | 408 | 4 | 4.773e-05 | 8.997e-05 |
| Borda | 34 | 140 | 1 | 78 | 419 | 4 | 0.001 | 1.4e-10 |
| Max-No | 25 | 149 | 1 | 61 | 436 | 4 | 6.276e-09 | 4.891e-07 |
| Max-Def | **41** | 133 | 1 | **110** | 387 | 4 | 0.008 | 0.642 |
| Min-No | **45** | 129 | 1 | 100 | 397 | 4 | 0.0004 | 9.099e-12 |
| Min-Def | 35 | 139 | 1 | 70 | 427 | 4 | 0.0002 | 3.508e-13 |
| CombMNZ-No | **41** | 133 | 1 | 94 | 403 | 4 | 1.488e-05 | 1.104e-07 |
| CombMNZ-Def | **44** | 130 | 1 | **118** | 379 | 4 | NA | NA |
| CombSUM-No | 33 | 141 | 1 | 76 | 421 | 4 | 7.281e-06 | 2.861e-07 |
| CombSUM-Def | **37** | 137 | 1 | 84 | 413 | 4 | 0.0005 | 0.004 |

forms the best on 28 concerns, CombMNZ-Def with L=2 performs the best on 22 concerns, and CombMNZ-Def with L=1 performs the best on 20 concerns. The effectiveness scores are the same for 13 concerns.

Table 7(a) and 7(b) shows the data analysis results on rank scores of CombMNZ-Def with different hierarchy heights (L=1,2,3,4,5,6). From the table, we can see that the variant of CombMNZ-Def with L set to 6 outperforms the others. Among the 501 methods, CombMNZ-Def with L=6 performs the best on 117 methods, CombMNZ-Def with L=5 performs the best on 107 methods, CombMNZ-Def with L=4 performs the best on 109 methods, CombMNZ-Def with L=3 performs the best on 84 methods, CombMNZ-Def with L=2 performs the best on 84 methods, and CombMNZ-Def with L=1 performs the best on 73 methods. The ranks scores are the same for 41 methods.

Though CombMNZ-Def with L=6 outperforms the other five in terms of effectiveness and rank, it needs longer time to train the model. Across the 9 data sets, we need about 254 minutes to train the model CombMNZ-Def with L=4, and 402 minutes to train the model CombMNZ-Def with L=6. We have performed a Wilcoxon signed-rank test at 95% significance level and found that the difference in the training time is significant with a p-value of $< 0.001$, which means the time efficiency of CombMNZ-Def with L=4 is significantly better than CombMNZ-Def with L=6. Then we compare the effectiveness and rank scores of CombMNZ-Def with L=4 and CombMNZ-Def with L=6. Wilcoxon sign-rank test shows that the difference in the effectiveness scores is not statistically significant at p-value of $0.076$, the Cliff's delta is 0.025, which corresponds to a negligible effect size. And Wilcoxon sign-rank test also shows that the difference in the rank scores is significant at p-value of $0.001$,

but the Cliff's delta is 0.018, which corresponds to a negligible effect size. So we can draw the conclusion that the height L=4 is the best choice when considering both performance and time efficiency.

*When evaluated by both effectiveness and rank,* CombMNZ-Def *performs the best among the 12 variants of* MULAB. *Height L=6 performs better than* CombMNZ-Def *with L=1, L=2, L=3, L=4, and L=5. But considering both performance and time efficiency, height L=4 is the best choice for our experiment.*

**Table 6** Data analysis results on effectiveness scores of CombMNZ-Def with different hierarchy heights (L=1,2,3,4,5,6). #Wins = Number of concerns for which a variant of CombMNZ-Def outperforms the others, #Loses = Number of concerns for which a variant CombMNZ-Def loses from another variant, #Draws = Number of concerns for which all variants perform equally well.

(a) L=1,2,3

| | **L=1** | | | **L=2** | | | **L=3** | | |
|---|---|---|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 1 | 7 | 0 | 1 | 7 | 0 | 2 | 6 | 0 |
| aTunes | 2 | 12 | 2 | 1 | 13 | 2 | 2 | 12 | 2 |
| jEdit4.2 | 2 | 13 | 1 | 3 | 12 | 1 | 3 | 12 | 1 |
| jEdit4.3 | 0 | 2 | 2 | 1 | 1 | 2 | 0 | 2 | 2 |
| Cocoon | 2 | 8 | 4 | 2 | 8 | 4 | 2 | 8 | 4 |
| Derby | 4 | 25 | 0 | 4 | 25 | 0 | 6 | 23 | 0 |
| Lucene | 2 | 19 | 3 | 2 | 19 | 3 | 3 | 18 | 3 |
| OpenJPA | 2 | 23 | 0 | 4 | 21 | 0 | 4 | 21 | 0 |
| Eclipse | 5 | 33 | 1 | 4 | 34 | 1 | 6 | 32 | 1 |
| **Overall** | **20** | **142** | **13** | **22** | **140** | **13** | **28** | **134** | **13** |

(b) L=4,5,6

| | **L=4** | | | **L=5** | | | **L=6** | | |
|---|---|---|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 2 | 6 | 0 | 1 | 7 | 0 | 2 | 6 | 0 |
| aTunes | 4 | 10 | 2 | 4 | 10 | 2 | 5 | 9 | 2 |
| jEdit4.2 | 5 | 10 | 1 | 6 | 9 | 1 | 4 | 11 | 1 |
| jEdit4.3 | 2 | 0 | 2 | 1 | 1 | 2 | 2 | 0 | 2 |
| Cocoon | 3 | 7 | 4 | 3 | 7 | 4 | 4 | 6 | 4 |
| Derby | 9 | 20 | 0 | 7 | 22 | 0 | 10 | 19 | 0 |
| Lucene | 7 | 14 | 3 | 8 | 13 | 2 | 6 | 15 | 3 |
| OpenJPA | 8 | 17 | 0 | 7 | 18 | 0 | 10 | 15 | 0 |
| Eclipse | 8 | 30 | 1 | 8 | 30 | 1 | 10 | 28 | 1 |
| **Overall** | **48** | **114** | **13** | **45** | **117** | **13** | **53** | **109** | **13** |

**Research Question 2: How much improvement could the best performing variant achieve over its components (i.e., VSM and topic models)?**

**Table 7** Data analysis results on rank scores of CombMNZ-Def with different hierarchy heights (L=1,2,3,4,5,6). #Wins = Number of methods for which a variant of CombMNZ-Def outperforms the others, #Loses = Number of methods for which a variant CombMNZ-Def loses from another variant, #Draws = Number of methods for which all variants perform equally well.

(a) L=1,2,3

| | L=1 | | | L=2 | | | L=3 | | |
|---|---|---|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 1 | 11 | 0 | 2 | 10 | 0 | 2 | 10 | 0 |
| aTunes | 3 | 22 | 5 | 4 | 21 | 5 | 5 | 20 | 5 |
| jEdit4.2 | 3 | 28 | 2 | 3 | 28 | 2 | 6 | 25 | 2 |
| jEdit4.3 | 0 | 7 | 2 | 2 | 5 | 2 | 1 | 6 | 2 |
| Cocoon | 4 | 22 | 12 | 8 | 18 | 12 | 5 | 21 | 12 |
| Derby | 14 | 64 | 2 | 13 | 65 | 2 | 17 | 61 | 2 |
| Lucene | 20 | 79 | 13 | 19 | 80 | 13 | 15 | 84 | 13 |
| OpenJPA | 11 | 61 | 2 | 13 | 59 | 2 | 10 | 62 | 2 |
| Eclipse | 17 | 93 | 3 | 20 | 90 | 3 | 23 | 87 | 3 |
| **Overall** | **73** | **387** | **41** | **84** | **376** | **41** | **84** | **376** | **41** |

(b) L=4,5,6

| | L=4 | | | L=5 | | | L=6 | | |
|---|---|---|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 3 | 9 | 0 | 2 | 10 | 0 | 3 | 9 | 0 |
| aTunes | 7 | 18 | 5 | 8 | 17 | 5 | 9 | 16 | 5 |
| jEdit4.2 | 9 | 22 | 2 | 8 | 23 | 2 | 10 | 21 | 2 |
| jEdit4.3 | 4 | 3 | 2 | 3 | 4 | 2 | 3 | 4 | 2 |
| Cocoon | 7 | 19 | 12 | 7 | 19 | 12 | 8 | 18 | 12 |
| Derby | 19 | 59 | 2 | 19 | 59 | 2 | 20 | 58 | 2 |
| Lucene | 22 | 77 | 13 | 24 | 75 | 13 | 25 | 74 | 13 |
| OpenJPA | 16 | 56 | 2 | 15 | 57 | 2 | 16 | 56 | 2 |
| Eclipse | 22 | 88 | 3 | 21 | 89 | 3 | 23 | 87 | 3 |
| **Overall** | **109** | **351** | **41** | **107** | **353** | **41** | **117** | **343** | **41** |

**Motivation.** We need to compare the best performing variant CombMNZ-Def with its components (i.e., VSM and topic models) to show whether MULAB with a data fusion method performs better than its constituent components. Answer to this research question shows the benefit of CombMNZ-Def over its components.

**Approach.** To answer this research question, we report the results obtained by applying CombMNZ-Def to our dataset mentioned in Section 4.1, and we also report the results of VSM model and each abstraction levels before using data fusion method CombMNZ, to investigate the performance of each component. The method CombMNZ-Def takes in one parameter $L$ which is the height of the abstraction hierarchy. For this RQ, we set $L$ to be 4. We compute the effectiveness and rank scores of CombMNZ-Def, VSM model and 4 abstraction levels (i.e., topic models) for

each concern and calculate the number of concerns for which each of the approach outperforms (or achieves the same scores as) the others.

**Results.** Tables 8 presents the effectiveness and rank results of COMBMNZ-DEF, VSM model, and 4 abstraction levels, we report the overall results of the 9 datasets here, and the detailed results will show in the Appendix B. For each method,we report the number of wins, loses, and draws for all the 9 Java systems. Wins, loses, and draws represent the number of concerns[8] or methods[9] for which a method performs the best, performs worse than another, and performs as well as the others, respectively. One approach wins on a concern when it performs better than all other 5 approaches, and it loses on a concern as long as one or more approaches perform better than it. So it normal that the number of #loses>#wins.

From the 6 tables, we can see that among the 175 concerns, COMBMNZ-DEF performs the best on 66 concerns, which is better than its five components. Among the 501 methods, when evaluated by rank, COMBMNZ-DEF wins on 138 methods, which is also better than its 5 components. So we can draw the conclusion that COMBMNZ-DEF outperforms its 5 components (VSM model and 4 abstraction levels) in terms of effectiveness and rank. Wilcoxon sign-rank test shows that the difference in the effectiveness scores of COMBMNZ-DEF compared with other components are statistically significant at p-value of $< 0.05$ except VSM, and the difference in the rank scores of COMBMNZ-DEF compared with other components are statistically significant at p-value of $< 0.05$.

**Table 8** Overall data analysis results of 9 datasets on effectiveness and rank scores of COMBMNZ-DEF and its components. #Wins = Number of concerns and mothods for which COMBMNZ-DEF or a component outperforms the others, #Loses = Number of concerns and methods for which COMBMNZ-DEF or a component loses from another component, #Draws = Number of concerns for which all methods perform equally well. We also report the p-value of COMBMNZ-DEF compared with its components.

| | **Effectiveness** | | | **Rank** | | | **P-value** | |
|---|---|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** | **Effectiveness** | **Rank** |
| COMBMNZ-DEF | 66 | 100 | 9 | 138 | 336 | 27 | NA | NA |
| VSM (Sim1) | 23 | 143 | 9 | 52 | 422 | 27 | 0.075 | 0.022 |
| level 1 (Sim2) | 32 | 134 | 9 | 79 | 395 | 27 | 6.299e-06 | 0.0006 |
| level 2 (Sim3) | 23 | 143 | 9 | 61 | 413 | 27 | 0.0003 | 0.014 |
| level 3 (Sim4) | 37 | 129 | 9 | 101 | 373 | 27 | 0.002 | 0.001 |
| level 4 (Sim5) | 38 | 128 | 9 | 101 | 373 | 27 | 2.753e-05 | 7.092e-07 |

*When evaluated by effectiveness and rank,* COMBMNZ-DEF *performs better than its 5 components (VSM model and 4 abstraction levels).*

**Research Question 3: How effective is** COMBMNZ-DEF **compared with state-of-the-art approach?**

[8] When effectiveness is used as a yardstick

[9] When rank is used as a yardstick

**Table 9** Data analysis results on effectiveness scores of CombMNZ-Def and PR. #Wins = Number of concerns for which CombMNZ-Def outperforms PR, #Loses = Number of concerns for which CombMNZ-Def loses from PR, #Draws = Number of concerns for which both approaches achieve the same effectiveness scores.

| **Systems** | #Wins | #Loses | #Draws |
|---|---|---|---|
| Art of Illusion | 4 | 4 | 0 |
| aTunes | 9 | 6 | 1 |
| jEdit4.2 | 8 | 8 | 0 |
| jEdit4.3 | 3 | 1 | 0 |
| Cocoon | 8 | 2 | 4 |
| Derby | 24 | 5 | 0 |
| Lucene | 20 | 4 | 0 |
| OpenJPA | 17 | 8 | 0 |
| Eclipse | 21 | 18 | 0 |
| **Overall** | **114** | **56** | **5** |

**Motivation.** We investigate the effectiveness of CombMNZ-Def and compare its results with those by Scanniello et al. (Scanniello et al, 2015) (*PR*, from here on). Answer to this research question would shed light to whether and to what extent CombMNZ-Def improves over the state-of-the-art approach.

**Approach.** To answer this research question, we report the results obtained by applying CombMNZ-Def and PR to our dataset mentioned in Section 4.1. CombMNZ-Def takes in one parameter $L$ which is the height of the abstraction hierarchy. For this RQ, we set $L$ to be 4. We compute the effectiveness and rank scores of CombMNZ-Def and PR for each concern and calculate the number of concerns for which each of the approach outperforms (or achieves the same scores as) the other.

To check if the differences in the performance of CombMNZ-Def and PR are statistically significant, we apply the Wilcoxon signed-rank test (Wilcoxon, 1945) at 95% significance level on two paired data of all the 175 concerns which corresponds to the effectiveness and rank scores of two competing approaches respectively. We do not apply the test to each system as the numbers of concerns in some systems are small (e.g., 4 for jEdit4.3, 8 for Art of Illusion), it makes no sense to do the statistical test. We also use Cliffs delta ($\delta$) (Cliff, 2014) to compare CombMNZ-Def with PR. The delta values range from -1 to 1, where $\delta = -1$ or 1 indicates the absence of overlap between two approaches (i.e., all values of one group are higher than the values of the other group, and vice versa), while $\delta = 0$ indicates the two approaches are completely overlapping.

**Results.** Table 9 presents the analysis results of effectiveness scores of CombMNZ-Def and PR. The second column represents the number of concerns on which CombMNZ-Def achieves better effectiveness scores than PR, the third column indicates the number of concerns on which PR performs better than CombMNZ-Def, and the last column shows the number of concerns on which CombMNZ-Def and PR achieve the same scores. We also report the overall results of the 9 systems in the last row. The results demonstrate that CombMNZ-Def is more effective than PR on all but two of the Java systems. For two of the Java systems (i.e., Art of Illusion, jEdit4.2), the

**Table 10** Data analysis results on Rank scores of CombMNZ-Def and PR. #Wins = Number of methods for which CombMNZ-Def outperforms PR, #Loses = Number of methods for which CombMNZ-Def loses from PR, #Draws = Number of methods for which both approaches achieve the same rank scores.

| **Systems** | #Wins | #Loses | #Draws |
|---|---|---|---|
| Art of Illusion | 6 | 6 | 0 |
| aTunes | 18 | 11 | 1 |
| jEdit4.2 | 19 | 14 | 0 |
| jEdit4.3 | 8 | 1 | 0 |
| Cocoon | 27 | 6 | 5 |
| Derby | 60 | 20 | 0 |
| Lucene | 82 | 30 | 0 |
| OpenJPA | 45 | 29 | 0 |
| Eclipse | 65 | 48 | 0 |
| **Overall** | **330** | **165** | **6** |

results in Table 9 show that CombMNZ-Def and PR perform equally well as they win on the same number of concerns. For Art of Illusion, CombMNZ-Def performs better on 4 concerns and PR performs better on another 4 concerns. For jEdit4.2, CombMNZ-Def performs better on 8 concerns and PR performs better on another 8 concerns. We look into the concerns in these two Java systems, find that PR performs better than CombMNZ-Def on the concerns which are short, and contain a mix of code, URL, etc. CombMNZ-Def achieves a better performance when the concerns are relatively long and there is no large semantic gap between concerns and source code. Among the 175 concerns, CombMNZ-Def performs better on 114 concerns, PR performs better on 56 concerns, and the two approaches achieve the same effectiveness scores on 5 concerns. We have also performed a Wilcoxon signed-rank test and found that the difference in the effectiveness scores is significant with a p-value of $< 0.001$. The Cliff's delta is 0.377, which corresponds to a medium effect size.

Table 10 presents the analysis results of rank scores of CombMNZ-Def as compared with those of PR. The second column represents the number of methods on which CombMNZ-Def achieves better rank scores than PR, the third column indicates the number of methods on which PR performs better than CombMNZ-Def, and the last column shows the number of methods on which CombMNZ-Def and PR achieve the same scores. We also report the overall results of the 9 systems in the last row. From the table, we can see that for the 9 systems, CombMNZ-Def performs better than that of the PR. The results demonstrate that CombMNZ-Def outperforms PR on all but one of the Java systems. For one of the Java systems (i.e., Art of Illusion), the results in Table 10 show that CombMNZ-Def and PR perform equally well as they win on the same number of methods, CombMNZ-Def performs better on 6 methods and PR performs better on another 6 methods. We look into the methods in Art of Illusion, find that CombMNZ-Def performs better when the methods are long, complicated, and contain more different words. But some methods in Art of Illusion are very short and contain many repeated words, on which PR achieves a better performance. Among the overall 501 methods, CombMNZ-Def performs better on 330 methods, PR performs better on 165 methods, and the two

approaches achieve the same rank scores on 6 methods. A Wilcoxon signed-rank test shows that the difference in the rank scores is significant with a p-value of $< 0.001$. The Cliff's delta is 0.399, which corresponds to a medium effect size.

COMBMNZ-DEF *outperforms* PR *on 7 among the 9 Java systems when evaluated in terms of effectiveness and rank. Statistical tests show that the differences are statistically significant and substantial.*

**Research Question 4: What is the effect of varying the text used to represent a concern on COMBMNZ-DEF's effectiveness?**

**Motivation.** By default, we use the text in the summary and description fields of bug reports and change requests to represent a concern – which is the setting used for RQ1-3 and RQ5. In this research question, we investigate the performance of COMBMNZ-DEF when we only use text in the summary field and text in the description field independently. We want to investigate if our default setting is a better option.

**Approach.** To answer this research question, we conduct an experiment with three kinds of text to represent a concern: default (summary and description), summary only, description only. COMBMNZ-DEF takes in one parameter $L$ which is the height of the abstraction hierarchy. For this RQ, we set $L$ to be 4. We compare the effectiveness and rank scores achieved by COMBMNZ-DEF using each of the three kinds of text.

**Table 11** Data analysis results on effectiveness scores of COMBMNZ-DEF with different kinds of text to represent a concern. #Wins = Number of concerns for which a kind of text performs the best, #Loses = Number of concerns for which a kind of text performs worse than another, #Draws = Number of concerns for which all kinds of text lead to the same score.

| | **Default** | | | **Summary** | | | **Description** | | |
|---|---|---|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 3 | 5 | 0 | 4 | 4 | 0 | 3 | 5 | 0 |
| aTunes | 7 | 8 | 1 | 4 | 11 | 1 | 6 | 9 | 1 |
| jEdit4.2 | 8 | 6 | 2 | 4 | 10 | 2 | 5 | 9 | 2 |
| jEdit4.3 | 2 | 2 | 0 | 2 | 2 | 0 | 2 | 2 | 0 |
| Cocoon | 10 | 2 | 2 | 4 | 8 | 2 | 6 | 6 | 2 |
| Derby | 10 | 19 | 0 | 15 | 14 | 0 | 9 | 20 | 0 |
| Lucene | 12 | 9 | 3 | 9 | 12 | 3 | 9 | 12 | 3 |
| OpenJPA | 10 | 15 | 0 | 10 | 15 | 0 | 9 | 16 | 0 |
| Eclipse | 16 | 22 | 1 | 14 | 24 | 1 | 11 | 27 | 1 |
| **Overall** | **78** | **88** | **9** | **66** | **100** | **9** | **60** | **106** | **9** |

**Results.** The experiment results are shown in Tables 11 and 12. For each kind of text (default, summary, or description), we report the number of wins, loses, and draws for each Java system. Wins, loses, and draws represent the number of concerns[10] or

[10] When effectiveness is used as a yardstick

**Table 12** Data analysis results on rank scores of COMBMNZ-DEF with different kinds of text to represent a concern. #Wins = Number of concerns for which a kind of text performs the best, #Loses = Number of concerns for which a kind of text performs worse than another, #Draws = Number of concerns for which all kinds of text lead to the same score.

| | **Default** | | | **Summary** | | | **Description** | | |
|---|---|---|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 4 | 8 | 0 | 6 | 6 | 0 | 3 | 9 | 0 |
| aTunes | 13 | 15 | 2 | 9 | 19 | 2 | 11 | 17 | 2 |
| jEdit4.2 | 16 | 15 | 2 | 8 | 23 | 2 | 10 | 21 | 2 |
| jEdit4.3 | 4 | 5 | 0 | 3 | 6 | 0 | 4 | 5 | 0 |
| Cocoon | 20 | 13 | 5 | 9 | 24 | 5 | 15 | 18 | 5 |
| Derby | 29 | 51 | 0 | 36 | 44 | 0 | 22 | 58 | 0 |
| Lucene | 46 | 57 | 9 | 36 | 67 | 9 | 34 | 69 | 9 |
| OpenJPA | 30 | 44 | 0 | 25 | 49 | 0 | 25 | 49 | 0 |
| Eclipse | 38 | 72 | 3 | 40 | 70 | 3 | 34 | 76 | 3 |
| **Overall** | **200** | **280** | **21** | **172** | **308** | **21** | **158** | **322** | **21** |

methods[11] for which a particular kind of text performs the best, performs worse than another, and performs as well as the others, respectively. We also report the overall results in the last row.

Table 11 shows the data analysis results on effectiveness scores of COMBMNZ-DEF using the three kinds of text to represent a concern. From the table, we can see that the default setting outperforms the others. Among the 175 concerns, *default* performs the best on 78 concerns, *summary* performs the best on 66 concerns, and *description* performs the best on 60 concerns. The effectiveness scores are the same for 9 concerns. So we can draw the conclusion that our default configuration (i.e., use both summary and description) outperforms the other two in terms of effectiveness.

Table 12 shows the data analysis results on rank scores of COMBMNZ-DEF using the three kinds of text to represent a concern. From the table, we can see that the default setting outperforms the others. Among the 501 methods, *default* performs the best on 200 methods, *summary* performs the best on 172 methods, and *description* performs the best on 158 methods. The rank scores are the same for 21 methods. So we can draw the conclusion that our default configuration (i.e., use both summary and description) outperforms the other two in terms of rank.

COMBMNZ-DEF *with default configuration (which uses text from both summary and description fields to represent a concern) performs better than when only text from summary and text from description are used independently, in terms of both effectiveness and rank scores.*

[11] When rank is used as a yardstick

### 4.4 Threats to Validity

Threats to internal validity relate to errors in our experiments. We have double checked our implementations and all the experiment results. Hence, we believe there are minimal threats to internal validity. Still, there could be errors that we did not notice.

Threats to external validity relate to the generalizability of our results. We have tried to mitigate this threat by evaluating our approach on concerns from 9 open source software systems, but there may still exist some other data sets on which our algorithm performs not so well. The software systems we used in our empirical study were chosen primarily because of the availability of data and previous studies. Data of these systems were manually vetted and a part of these systems were also used in previous work (Haiduc et al, 2013; Moreno et al, 2013; Scanniello and Marcus, 2011; Scanniello et al, 2015). Admittedly, the concerns that we investigate may not sufficiently represent all concerns from all systems. Finally, our choice of baseline clearly impacts the results. As future work, we plan to study more baselines.

Threats to construct validity refer to the suitability of our evaluation metrics. We use effectiveness and rank which are also used by past software engineering studies to evaluate the effectiveness of concern localization techniques (Gay et al, 2009; Poshyvanyk et al, 2007; Scanniello and Marcus, 2011; Scanniello et al, 2015). Thus, we believe there is little threat to construct validity.

## 5 Discussion

### 5.1 Qualitative Analysis

In the last section, the experiment results show that CombMNZ-Def performs better than other variants in terms of effectiveness and rank. In this subsection, we discuss the results further by answering the following questions. We randomly selected three issue reports as illustrative examples to answer some of the questions. Among the three issue reports, two are selected from jEdit4.2 (issue 993789 and 1275607) to show different results within a project and another one is selected from OpenJPA (issue 2010) to represent a different project. Table 13 shows for each issue report the positions of relevant files in the ranked list and the similarity scores computed by the 5 components and the 12 variants of MULAB.

*Why* CombMNZ-Def *outperforms the other variants?*

CombMNZ-Def applies score normalization to the 5 components and uses CombMNZ as data fusion method. The following paragraphs provide reasons why each of these two are good choices.

Benefit of score normalization: Some components may produce too big or too small scores, which may impact the performance of data fusion. After normalization, the scores of the components are all in range 0-1 which improves the effectiveness of variants that use this step. From Table 13, we can see that the score normalization method can improve the data fusion performance in most cases.

Benefit of CombMNZ: CombMNZ first sums up all scores for a given method to balance the difference of the scores, second it multiplies the summation with the

number of non-zero scores. There are two fusion methods that have similar computational process with COMBMNZ: COMBSUM and COMBANZ. COMBMNZ can enhance the impact of zero scores when compared with COMBSUM and COMBANZ. For COMBSUM, it just sums up all the scores and ignore the existence of zero scores; but zero score may indicate that this method is not similar with the concern in some aspects. For COMBANZ, it sums up all scores for a given method and divide the summation with the number of non-zero scores. The division process will weaken the impact of zero scores.

To compare COMBMNZ and COMBANZ, we show an example in Table 14. We list the similarity scores of two methods compared with issue report 1275607 when using the 5 components, COMBANZ, and COMBMNZ. We can see that method 1658 contains 4 non-zero scores, and method 2573 contains 5 non-zero scores, but the average of non-zero scores of method 1658 is a little bit bigger than method 2573. When using COMBANZ, method 1658 gets a higher position in the ranked list even though it contains a zero score. However, when we use COMBMNZ, the multiplication step strengthens the effect of the zero score for method 1658, and this causes method 2573 to get a higher position than method 1658 in the ranked list.

Aside from COMBMNZ, COMBSUM, and COMBANZ, we also have BORDA, MIN and MAX. BORDA, MIN, and MAX are easily impacted by a bad performing component. For example, MIN chooses the smallest score and ignores others. Thus, if two methods receive the same minimum score (outputted by a component), MIN cannot differentiate which method is more similar to the concern. Consider issue report 2010, MIN chooses the smallest scores 0.008 from the worst component (i.e., level 1) resulting in bad performance. A similar argument explains the bad performance of MAX. For BORDA, a bad performing component may return a very large rank C consider issue report 1275607 and 2010 in the table.

*Are* MULAB *performing better for certain subject systems more than others?*

For each of the 9 datasets, the performance of MULAB is similar. MULAB just performs a little better for dataset Cocoon.

*Are* MULAB *components complementary in their results?*

MULAB has several components: VSM and topic models learned using LDA. VSM only consider the word frequency in the documents and ignore the semantic of words. On the other hand, LDA can analyze the semantic of words, and abstract documents to different sets of topics. After combining VSM and LDA, the model is optimized by considering both word frequency and semantic. Consider the issue report 2010 shown in Table 13, if we only use VSM, we can get the relevant method at the 54th position in the ranked list, and if we only use a LDA(level 2), the relevant method is returned at the 224th position in the ranked list. However, if we fuse the 5 components (VSM and 4 LDA models), $\text{MULAB}_{basic}$ can return the relevant method at the 8th position in the ranked list, while COMBANZ, COMBMNZ, and COMBSUM can return the relevant method at the 2nd position in the ranked list. This demonstrates that MULAB components are complementary in their results.

*What kinds of concerns are hard to localize by* MULAB*?*

There exist some concerns for which MULAB cannot produce good results. Consider issue report 2289 from OpenJPA. Figure 2 shows the title and description of

**Table 13** The positions(similarity scores) in the ranked lists of the relevant methods of 3 issue reports when using the 5 components and the 12 variants of MULAB.

| Algorithms | issue report 993789 | issue report 1275607 | issue report 2010 |
|---|---|---|---|
| VSM (Sim1) | 1(0.358) | 144(0.077) | 54(0.240) |
| level 1 (Sim2) | 1(0.734) | 286(0.073) | 32606(0.008) |
| level 2 (Sim3) | 157(0.033) | 3992(0.007) | 224(0.192) |
| level 3 (Sim4) | 2(0.456) | 203(0.027) | 28(0.636) |
| level 4 (Sim5) | 1775(0.003) | 3547(0.003) | 10(0.412) |
| MULAB$_{basic}$ | 1(0.247) | 266(0.029) | 8(0.269) |
| COMBANZ-NO | 1(0.317) | 254(0.037) | 2(0.298) |
| COMBANZ-DEF | 1(0.618) | 201(0.107) | 2(0.534) |
| BORDA | 39(24984) | 1180(18748) | 1186(174448) |
| MAX-NO | 1(0.734) | 540(0.077) | 33(0.636) |
| MAX-DEF | 1(1.000) | 286(0.305) | 59(0.882) |
| MIN-NO | 1054(0.003) | 2089(0.003) | 2402(0.008) |
| MIN-DEF | 1001(0.006) | 2069(0.003) | 4068(0.010) |
| COMBMNZ-NO | 1(7.917) | 237(0.933) | 2(7.452) |
| COMBMNZ-DEF | 1(15.456) | 194(2.677) | 2(13.340) |
| COMBSUM-NO | 1(1.583) | 242(0.187) | 2(1.490) |
| COMBSUM-DEF | 1(3.091) | 197(0.535) | 2(2.668) |

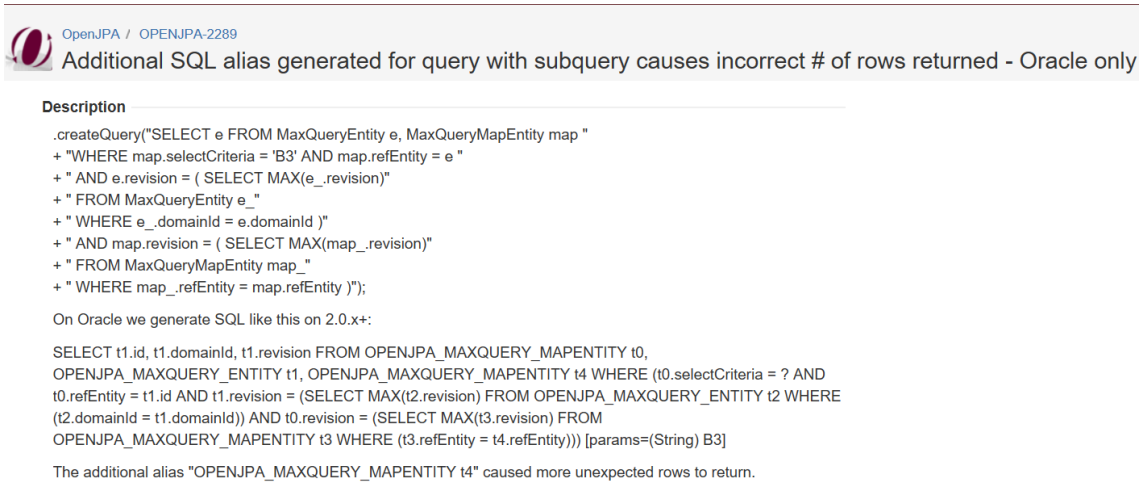
**Table 14** The similarity scores of two methods compared with issue report 1275607 when using the 5 components, COMBANZ, and COMBMNZ.

| Method ID | VSM | level 1 | level 2 | level 3 | level 4 | COMBANZ | COMBMNZ |
|---|---|---|---|---|---|---|---|
| 1658 | 0 | 0.488 | 0.294 | 0.072 | 0.096 | 0.237(57th position) | 3.797(106th position) |
| 2573 | 0.363 | 0.290 | 0.105 | 0.009 | 0.413 | 0.236(58th position) | 5.896(57th position) |

**Table 15** The positions in the ranked list of the relevant method of issue report 2289 of 12 variants of MULAB.

| | MULAB$_{basic}$ | COMBANZ-NO | COMBANZ-DEF | BORDA | MAX-NO | MAX-DEF |
|---|---|---|---|---|---|---|
| Positions | 3926 | 13948 | 6683 | 23683 | 7241 | 3418 |
| | MIN-NO | MIN-DEF | COMBMNZ-NO | COMBMNZ-DEF | COMBSUM-NO | COMBSUM-DEF |
| Positions | 9058 | 14777 | 12197 | 5800 | 13058 | 6157 |

issue report 2289. Figure 3 shows the relevant method that need to be modified to resolve the bug described in issue report 2289. Table 15 shows the positions of this method in the ranked list produced by the 12 variants of MULAB. From the Figure 2 and 3, we can see that the text in the description of issue report 2289 are mostly SQL statements, but the corresponding method that needs to be modified is written in Java with some English comments. MULAB cannot perform well for this (as shown in Table 15) and several other concerns for which semantic gap between text in issue reports and their corresponding methods is large



OpenJPA / OPENJPA-2289

Additional SQL alias generated for query with subquery causes incorrect # of rows returned - Oracle only

**Description**

.createQuery("SELECT e FROM MaxQueryEntity e, MaxQueryMapEntity map "
+ "WHERE map.selectCriteria = 'B3' AND map.refEntity = e "
+ " AND e.revision = ( SELECT MAX(e_.revision)"
+ " FROM MaxQueryEntity e_"
+ " WHERE e_.domainId = e.domainId )"
+ " AND map.revision = ( SELECT MAX(map_.revision)"
+ " FROM MaxQueryMapEntity map_"
+ " WHERE map_.refEntity = map.refEntity )");

On Oracle we generate SQL like this on 2.0.x+:

SELECT t1.id, t1.domainId, t1.revision FROM OPENJPA_MAXQUERY_MAPENTITY t0,
OPENJPA_MAXQUERY_ENTITY t1, OPENJPA_MAXQUERY_MAPENTITY t4 WHERE (t0.selectCriteria = ? AND
t0.refEntity = t1.id AND t1.revision = (SELECT MAX(t2.revision) FROM OPENJPA_MAXQUERY_ENTITY t2 WHERE
(t2.domainId = t1.domainId)) AND t0.revision = (SELECT MAX(t3.revision) FROM
OPENJPA_MAXQUERY_MAPENTITY t3 WHERE (t3.refEntity = t4.refEntity))) [params=(String) B3]

The additional alias "OPENJPA_MAXQUERY_MAPENTITY t4" caused more unexpected rows to return.

**Fig. 2** Issue report 2289 from OpenJPA.

```
1  package org.apache.openjpa.jdbc.sql;
2  /*Standard {@link Select} implementation. Usage note:...*/
3  public class SelectImpl
4          public Joins crossJoin(Table localTable, Table foreignTable) {
5              // cross joins are for unbound variables; unfortunately we have
6              // to always go DISTINCT for unbound vars because there are certain
7              // cases that require it, and we can't differentiate them from the
8              // cases that don't
9              _sel._flags |= IMPLICIT_DISTINCT;
10
11             if (_sel.getJoinSyntax() != JoinSyntaxes.SYNTAX_SQL92
12                 || _sel._from != null) {
13                 // don't make any joins, but update the path if a variable
14                 // has been set
15                 this.append(this.var);
16                 this.var = null;
17                 _outer = false;
18                 return this;
19             }
20             ......
21             return this;
22         }
```

**Fig. 3** Relevant method of issue report 2289.

## 5.2 Time Efficiency

The efficiency of the algorithm will affect its practical usage. Thus, in this subsection, we investigate the time efficiency of MULAB with height 4. We use an Intel(R) Core(TM) i7-6850K 3.60 GHz CPU, 64GB RAM server to run the experiments. We run MULAB and report the average model training and test time. Model training

**Table 16** Model testing time of 12 variants of MULAB.

| | MULAB$_{basic}$ | COMBANZ-NO | COMBANZ-DEF | BORDA | MAX-NO | MAX-DEF |
|---|---|---|---|---|---|---|
| Testing time(s) | 2.52 | 2.34 | 2.40 | 2.61 | 2.30 | 2.35 |
| | MIN-NO | MIN-DEF | COMBMNZ-NO | COMBMNZ-DEF | COMBSUM-NO | COMBSUM-DEF |
| Testing time(s) | 2.29 | 2.34 | 2.34 | 2.40 | 2.33 | 2.39 |

time refers to the time taken to tune the topic numbers and create hierarchy. Test time refers to the time taken for MULAB to retrieve similar documents for each concern. The training time of the 12 variants are the same, as we only need to train the model once. We notice that the training time of MULAB are reasonable, e.g., on average, we need about 198 minutes to tune the topic numbers, 56 minutes to create hierarchy. The testing time of the 12 variants are shown in Table 16, for each variant, we report the average testing time of a concern in the second row, and for all the variants, the average time to retrieve similar documents for a concern is 2.38 seconds. Notice that the training phase can be done offline (e.g., overnight) and the model does not need to be updated all the time. The model only needs to be retrained when large numbers of code changes are made which leads to the original model being no longer accurate. A trained model can be used to retrieve many concerns.

## 6 Related Work

### 6.1 Concern Localization

Concern localization is an important and recurring step in maintenance of a software system. We describe some past studies in the following paragraphs. Due to space limitations, our survey is by no means complete.

**Text analysis.** Wang et al. (Wang et al, 2011b) evaluate 10 information retrieval techniques and discover that VSM has the best performance. Rao and Kak also investigate the use of LDA with VSM (Rao and Kak, 2011). However, in their approach, VSM is considered separately from LDA. The results of the two are combined together using a *weighted sum*. The performance of the resulting composite model is *worse* than that of VSM. In this work, we integrate LDA and VSM by constructing a single unified vector and we use a hierarchy of topic models; the resulting approach performs better than Scanniello et al.'s approach, which has been shown to be better than VSM on the same dataset (Scanniello et al, 2015).

**Text and static analysis.** To improve the accuracy of concern localization, a few hybrid approaches have been proposed, which combine IR techniques with static program analysis. Zhao et al. (Zhao et al, 2006) present a two-phase approach to concern localization, which first applies an IR technique to identify an initial set of feature-code-unit links based on the textual description of the concerns and code units, and then enrich the initial links by exploring program call graph. Similarly, Eaddy et

al. (Eaddy et al, 2008) employ pruned dependency analysis to boost the recall of IR or dynamic-analysis-based approaches. Most recently, Scanniello et al. (Scanniello et al, 2015) propose a text retrieval-based concern localization technique which considers the structural relationships between source code documents. They use a link analysis algorithm PageRank to rank the document space and to improve concern localization. The algorithm uses links (i.e., dependencies) among documents to organize them into a hierarchical structure. With their technique, source code documents are automatically ranked with respect to a textual query written by the developer, based on the dependencies and the lexical similarities between the documents. We have shown that our approach which relies only on textual contents of concerns and methods are able to outperform the latest approach by Scanniello et al. on a benchmark dataset used by many prior studies.

**Text, static and/or dynamic analysis.** Aside from text and information gleaned using static analysis, execution traces have been used to aid concern localization. Liu et al. (Liu and Xu, 2007) apply IR-based filtering to rank the methods being executed in a single test scenario. Dit et al. (Dit et al, 2013) define a data fusion model for feature location that integrates different types of information to locate features using IR, dynamic analysis, and web mining algorithms. Our technique does not consider execution traces since most bug reports and change requests do not come with execution traces (Sun et al, 2011).

### 6.2 Search-Based Algorithms in Software Engineering

Search-based algorithms have been used to improve various software engineering activities. Harman and Jones propose the concept of search-based software engineering and they demonstrate how to reformulate a SE problem as a search-based problem (Harman and Jones, 2001). Later, Harman et al. provide a review and classification of search-based software engineering techniques (Harman et al, 2012). Many search-based algorithms have been proposed in the literature; we highlight a number of them in the following paragraphs.

Li et al. use various search algorithms including greedy search, hill climbing, and genetic algorithms for test case prioritization (Li et al, 2007). Canfora et al. construct a classification model by using multi-objective genetic algorithm for cross-project defect prediction (Canfora et al, 2013). Wang et al. propose a search-based approach for clone detection (Wang et al, 2013). A number of search-based algorithms have been proposed to generate test cases that satisfy various criteria for various programs (Tonella, 2004). Antoniol et al. apply a genetic algorithm to allocate staff to project teams and to allocate teams to work package (Antoniol et al, 2004). Gold et al. reformulate concept binding problem (i.e., assigning the most plausible concept for a source code segment) as a search problem to allow overlapping concept boundaries, and genetic and hill climbing algorithms are used to search for solutions to this problem (Gold et al, 2006).

Mancoridis *et al.* use a search-based algorithm to group software modules into clusters by minimizing cohesion and maximizing coupling (Mancoridis et al, 1999).

Wang et al. use a genetic algorithm to improve fault localization; their approach analyzes a set of failing and correct execution traces to locate faulty basic blocks that are root causes of bugs (Wang et al, 2011a). Goues et al. propose *GenProg*, which uses genetic algorithm to automatically repair defects in software projects (Le Goues et al, 2012). Le et al. propose HDRepair that mines bug fix patterns from version history and subsequently uses genetic programming to evolve patches for new bugs based on mined fix patterns (Le et al, 2016b). Le et al. propose to use program logic specifications to evolve a buggy implementation until a correct patch is found via genetic programming and deductive verification (Le et al, 2016a). More recently, Panichella et al. use genetic algorithm to identify near optimal solutions to customize various stages of an IR process (Panichella et al, 2016). The proposed approach explores what kinds of character pruning, identifier splitting, stop word removal, stemming, term weighting, and IR techniques are best to be used. Lohar et al. present a novel approach to trace retrieval, which utilizes a machine-learning engine to search for the best configuration given an initial training set of validated trace links (Lohar et al, 2013). Wang et al. introduce desktop and parallelised cloud-deployed versions of a search-based solution that finds suitable configurations for empirical studies (Wang et al, 2013). Xia et al. propose an accurate change classification technique named collective personalized change classification (CPCC), which leverages a multi-objective genetic algorithm (Xia et al, 2016b). They also utilize genetic algorithm to do cross-project defect prediction; in particular, they propose a hybrid model reconstruction approach, named HYDRA, which contains two phases: genetic algorithm (GA) phase and ensemble learning (EL) phase (Xia et al, 2016a).

In this work, similar to the above approaches, we also utilize a search-based algorithm. However, we address a new problem, namely multi-abstraction concern localization. The approach by Panichella et al. (Panichella et al, 2016) only considers one level of abstraction.

## 7 Conclusion and Future Work

Existing concern localization studies characterize both concerns and code units as a bag of tokens at *one abstraction level*. In this study, we propose a multi-abstraction concern localization technique named MULAB which combines a hierarchy of topic models with VSM. We use genetic algorithm to estimate a near-optimal configuration of the topic models. Our experiments on 175 concerns from 9 open-source software systems show that our approach performs better than PR, the state-of-art approach recently proposed by Scanniello et al. (Scanniello et al, 2015), when evaluated in terms of effectiveness and rank. In the future, we plan to perform a deeper analysis on cases where our multi-abstraction approach does not work well, and improve the effectiveness of our proposed approach further. We also plan to merge our approach with other advanced text mining solutions, e.g., paraphrase detection, deep learning, etc., for more optimal performance. What's more, we plan to investigate some intermediate data sources like API documents and knowledge from online forums to help to bridge the semantic gap between issue reports and their corresponding source code methods.

**Acknowledgment.** This work was supported by NSFC Program (No. 61602403 and 61572426).

## A Appendix A

Following tables show the detailed experiment results of Research Question 1, each table presents the effectiveness and rank scores of 1 among the 12 variants of MULAB. For each variant, we report the number of wins, loses, and draws for each Java system. Wins, loses, and draws represent the number of concerns[12] or methods[13] for which a variant outperforms the other variants, loses from another variant, and performs as well as all the other variants, respectively. We also report the overall results in the last row.

**Table 17** Data analysis results on effectiveness and rank scores of $\text{MULAB}_{basic}$.

| | **Effectiveness** | | | **Rank** | | |
|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 1 | 7 | 0 | 1 | 11 | 0 |
| aTunes | 5 | 11 | 0 | 11 | 19 | 0 |
| jEdit4.2 | 5 | 11 | 0 | 6 | 27 | 0 |
| jEdit4.3 | 2 | 2 | 0 | 4 | 5 | 0 |
| Cocoon | 8 | 6 | 0 | 20 | 18 | 0 |
| Derby | 1 | 28 | 0 | 9 | 71 | 0 |
| Lucene | 5 | 18 | 1 | 24 | 84 | 4 |
| OpenJPA | 1 | 24 | 0 | 11 | 63 | 0 |
| Eclipse | 9 | 30 | 0 | 18 | 95 | 0 |
| **Overall** | **37** | **137** | **1** | **104** | **393** | **4** |

## B Appendix B

Following tables show the detailed experiment results of Research Question 2, each table presents the effectiveness and rank scores of CombMNZ-Def, VSM model, and 4 abstraction levels. For each method, we report the number of wins, loses, and draws for each Java system. Wins, loses, and draws represent the number of concerns[14] or methods[15] for which a method performs the best, loses from another method, and performs as well as all the other methods, respectively. We also report the overall results in the last row.

[12] When effectiveness is used as a yardstick
[13] When rank is used as a yardstick
[14] When effectiveness is used as a yardstick
[15] When rank is used as a yardstick

**Table 18** Data analysis results on effectiveness and rank scores of COMBANZ-NO.

| | Effectiveness | | | Rank | | |
|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 3 | 5 | 0 | 3 | 9 | 0 |
| aTunes | 3 | 13 | 0 | 4 | 26 | 0 |
| jEdit4.2 | 2 | 14 | 0 | 5 | 28 | 0 |
| jEdit4.3 | 1 | 3 | 0 | 1 | 8 | 0 |
| Cocoon | 7 | 7 | 0 | 15 | 23 | 0 |
| Derby | 1 | 28 | 0 | 4 | 76 | 0 |
| Lucene | 5 | 18 | 1 | 15 | 93 | 4 |
| OpenJPA | 4 | 21 | 0 | 9 | 65 | 0 |
| Eclipse | 1 | 38 | 0 | 12 | 101 | 0 |
| **Overall** | **27** | **147** | **1** | **68** | **429** | **4** |

**Table 19** Data analysis results on effectiveness and rank scores of COMBANZ-DEF.

| | Effectiveness | | | Rank | | |
|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 1 | 7 | 0 | 2 | 10 | 0 |
| aTunes | 1 | 15 | 0 | 3 | 27 | 0 |
| jEdit4.2 | 3 | 13 | 0 | 6 | 27 | 0 |
| jEdit4.3 | 1 | 3 | 0 | 1 | 8 | 0 |
| Cocoon | 7 | 7 | 0 | 16 | 22 | 0 |
| Derby | 4 | 25 | 0 | 10 | 70 | 0 |
| Lucene | 7 | 16 | 1 | 24 | 84 | 4 |
| OpenJPA | 1 | 24 | 0 | 4 | 70 | 0 |
| Eclipse | 4 | 35 | 0 | 23 | 90 | 0 |
| **Overall** | **29** | **145** | **1** | **89** | **408** | **4** |

**Table 20** Data analysis results on effectiveness and rank scores of MULAB$_{Borda}$.

| | Effectiveness | | | Rank | | |
|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 1 | 7 | 0 | 1 | 11 | 0 |
| aTunes | 3 | 13 | 0 | 4 | 26 | 0 |
| jEdit4.2 | 1 | 15 | 0 | 2 | 31 | 0 |
| jEdit4.3 | 3 | 1 | 0 | 5 | 4 | 0 |
| Cocoon | 3 | 11 | 0 | 10 | 28 | 0 |
| Derby | 7 | 22 | 0 | 15 | 65 | 0 |
| Lucene | 8 | 15 | 1 | 16 | 92 | 4 |
| OpenJPA | 2 | 23 | 0 | 6 | 68 | 0 |
| Eclipse | 6 | 33 | 0 | 19 | 94 | 0 |
| **Overall** | **34** | **140** | **1** | **78** | **419** | **4** |

**Table 21** Data analysis results on effectiveness and rank scores of MAX-NO.

| | Effectiveness | | | Rank | | |
|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 1 | 7 | 0 | 1 | 11 | 0 |
| aTunes | 4 | 12 | 0 | 8 | 22 | 0 |
| jEdit4.2 | 3 | 13 | 0 | 5 | 28 | 0 |
| jEdit4.3 | 1 | 3 | 0 | 1 | 8 | 0 |
| Cocoon | 4 | 10 | 0 | 13 | 25 | 0 |
| Derby | 5 | 24 | 0 | 11 | 69 | 0 |
| Lucene | 1 | 22 | 1 | 8 | 100 | 4 |
| OpenJPA | 1 | 24 | 0 | 3 | 71 | 0 |
| Eclipse | 5 | 34 | 0 | 11 | 102 | 0 |
| **Overall** | **25** | **149** | **1** | **61** | **436** | **4** |

**Table 22** Data analysis results on effectiveness and rank scores of MAX-DEF.

| | Effectiveness | | | Rank | | |
|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 2 | 6 | 0 | 2 | 10 | 0 |
| aTunes | 1 | 15 | 0 | 2 | 28 | 0 |
| jEdit4.2 | 5 | 11 | 0 | 10 | 23 | 0 |
| jEdit4.3 | 1 | 3 | 0 | 2 | 7 | 0 |
| Cocoon | 6 | 8 | 0 | 14 | 24 | 0 |
| Derby | 5 | 24 | 0 | 17 | 63 | 0 |
| Lucene | 8 | 15 | 1 | 22 | 86 | 4 |
| OpenJPA | 6 | 19 | 0 | 18 | 56 | 0 |
| Eclipse | 7 | 32 | 0 | 23 | 90 | 0 |
| **Overall** | **41** | **133** | **1** | **110** | **387** | **4** |

**Table 23** Data analysis results on effectiveness and rank scores of MIN-NO.

| | Effectiveness | | | Rank | | |
|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 0 | 8 | 0 | 2 | 10 | 0 |
| aTunes | 2 | 14 | 0 | 3 | 27 | 0 |
| jEdit4.2 | 3 | 13 | 0 | 4 | 29 | 0 |
| jEdit4.3 | 2 | 2 | 0 | 3 | 6 | 0 |
| Cocoon | 4 | 10 | 0 | 15 | 23 | 0 |
| Derby | 8 | 21 | 0 | 13 | 67 | 0 |
| Lucene | 7 | 16 | 1 | 18 | 90 | 4 |
| OpenJPA | 11 | 14 | 0 | 15 | 59 | 0 |
| Eclipse | 8 | 31 | 0 | 27 | 86 | 0 |
| **Overall** | **45** | **129** | **1** | **100** | **397** | **4** |

**Table 24** Data analysis results on effectiveness and rank scores of MIN-DEF.

| | Effectiveness | | | Rank | | |
|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 0 | 8 | 0 | 0 | 12 | 0 |
| aTunes | 3 | 13 | 0 | 6 | 24 | 0 |
| jEdit4.2 | 1 | 15 | 0 | 2 | 31 | 0 |
| jEdit4.3 | 1 | 3 | 0 | 1 | 8 | 0 |
| Cocoon | 5 | 9 | 0 | 13 | 25 | 0 |
| Derby | 4 | 25 | 0 | 7 | 73 | 0 |
| Lucene | 9 | 14 | 1 | 17 | 91 | 4 |
| OpenJPA | 4 | 21 | 0 | 9 | 65 | 0 |
| Eclipse | 8 | 31 | 0 | 15 | 98 | 0 |
| **Overall** | **35** | **139** | **1** | **70** | **427** | **4** |

**Table 25** Data analysis results on effectiveness and rank scores of COMBMNZ-NO.

| | Effectiveness | | | Rank | | |
|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 3 | 5 | 0 | 3 | 9 | 0 |
| aTunes | 5 | 11 | 0 | 6 | 24 | 0 |
| jEdit4.2 | 2 | 14 | 0 | 5 | 28 | 0 |
| jEdit4.3 | 1 | 3 | 0 | 1 | 8 | 0 |
| Cocoon | 8 | 6 | 0 | 16 | 22 | 0 |
| Derby | 2 | 27 | 0 | 10 | 70 | 0 |
| Lucene | 5 | 18 | 1 | 21 | 87 | 4 |
| OpenJPA | 6 | 19 | 0 | 10 | 64 | 0 |
| Eclipse | 9 | 30 | 0 | 22 | 91 | 0 |
| **Overall** | **41** | **133** | **1** | **94** | **403** | **4** |

**Table 26** Data analysis results on effectiveness and rank scores of COMBMNZ-DEF.

| | Effectiveness | | | Rank | | |
|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 1 | 7 | 0 | 3 | 9 | 0 |
| aTunes | 2 | 14 | 0 | 4 | 26 | 0 |
| jEdit4.2 | 4 | 12 | 0 | 8 | 25 | 0 |
| jEdit4.3 | 1 | 3 | 0 | 1 | 8 | 0 |
| Cocoon | 7 | 7 | 0 | 16 | 22 | 0 |
| Derby | 10 | 19 | 0 | 19 | 61 | 0 |
| Lucene | 7 | 16 | 1 | 23 | 85 | 4 |
| OpenJPA | 3 | 22 | 0 | 19 | 55 | 0 |
| Eclipse | 9 | 30 | 0 | 25 | 88 | 0 |
| **Overall** | **44** | **130** | **1** | **118** | **379** | **4** |

**Table 27** Data analysis results on effectiveness and rank scores of CombSUM-No.

| | Effectiveness | | | Rank | | |
|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 2 | 6 | 0 | 2 | 10 | 0 |
| aTunes | 4 | 12 | 0 | 5 | 25 | 0 |
| jEdit4.2 | 1 | 15 | 0 | 5 | 28 | 0 |
| jEdit4.3 | 1 | 3 | 0 | 1 | 8 | 0 |
| Cocoon | 7 | 7 | 0 | 16 | 22 | 0 |
| Derby | 2 | 27 | 0 | 5 | 75 | 0 |
| Lucene | 5 | 18 | 1 | 18 | 90 | 4 |
| OpenJPA | 5 | 20 | 0 | 9 | 65 | 0 |
| Eclipse | 6 | 33 | 0 | 15 | 98 | 0 |
| **Overall** | **33** | **141** | **1** | **76** | **421** | **4** |

**Table 28** Data analysis results on effectiveness and rank scores of CombSUM-Def.

| | Effectiveness | | | Rank | | |
|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 1 | 7 | 0 | 2 | 10 | 0 |
| aTunes | 2 | 14 | 0 | 4 | 26 | 0 |
| jEdit4.2 | 3 | 13 | 0 | 5 | 28 | 0 |
| jEdit4.3 | 1 | 3 | 0 | 1 | 8 | 0 |
| Cocoon | 7 | 7 | 0 | 16 | 22 | 0 |
| Derby | 6 | 23 | 0 | 11 | 69 | 0 |
| Lucene | 7 | 16 | 1 | 24 | 84 | 4 |
| OpenJPA | 2 | 23 | 0 | 8 | 66 | 0 |
| Eclipse | 8 | 31 | 0 | 13 | 100 | 0 |
| **Overall** | **37** | **137** | **1** | **84** | **413** | **4** |

**Table 29** Data analysis results on effectiveness and rank scores of CombMNZ-Def.

| | Effectiveness | | | Rank | | |
|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 3 | 5 | 0 | 5 | 7 | 0 |
| aTunes | 5 | 10 | 1 | 13 | 15 | 2 |
| jEdit4.2 | 6 | 10 | 0 | 11 | 20 | 2 |
| jEdit4.3 | 1 | 3 | 0 | 2 | 7 | 0 |
| Cocoon | 3 | 8 | 3 | 6 | 21 | 11 |
| Derby | 15 | 14 | 0 | 29 | 50 | 1 |
| Lucene | 10 | 10 | 4 | 28 | 76 | 8 |
| OpenJPA | 9 | 16 | 0 | 22 | 51 | 1 |
| Eclipse | 14 | 24 | 1 | 22 | 89 | 2 |
| **Overall** | **66** | **100** | **9** | **138** | **336** | **27** |

**Table 30** Data analysis results on effectiveness and rank scores of VSM model(Sim1).

| | Effectiveness | | | Rank | | |
|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 1 | 7 | 0 | 2 | 10 | 0 |
| aTunes | 3 | 12 | 1 | 3 | 25 | 2 |
| jEdit4.2 | 1 | 15 | 0 | 2 | 29 | 2 |
| jEdit4.3 | 0 | 4 | 0 | 0 | 9 | 0 |
| Cocoon | 0 | 11 | 3 | 2 | 25 | 11 |
| Derby | 5 | 24 | 0 | 8 | 71 | 1 |
| Lucene | 3 | 17 | 4 | 10 | 94 | 8 |
| OpenJPA | 4 | 21 | 0 | 10 | 63 | 1 |
| Eclipse | 6 | 32 | 1 | 15 | 96 | 2 |
| **Overall** | **23** | **143** | **9** | **52** | **422** | **27** |

**Table 31** Data analysis results on effectiveness and rank scores of abstraction level 1 (Sim2).

| | Effectiveness | | | Rank | | |
|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 2 | 6 | 0 | 3 | 9 | 0 |
| aTunes | 2 | 13 | 1 | 4 | 24 | 2 |
| jEdit4.2 | 1 | 15 | 0 | 4 | 27 | 2 |
| jEdit4.3 | 2 | 2 | 0 | 3 | 6 | 0 |
| Cocoon | 4 | 7 | 3 | 8 | 19 | 11 |
| Derby | 6 | 23 | 0 | 15 | 64 | 1 |
| Lucene | 5 | 15 | 4 | 18 | 86 | 8 |
| OpenJPA | 5 | 20 | 0 | 13 | 60 | 1 |
| Eclipse | 5 | 33 | 1 | 11 | 100 | 2 |
| **Overall** | **32** | **134** | **9** | **79** | **395** | **27** |

**Table 32** Data analysis results on effectiveness and rank scores of abstraction level 2 (Sim3).

| | Effectiveness | | | Rank | | |
|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 0 | 8 | 0 | 1 | 11 | 0 |
| aTunes | 1 | 14 | 1 | 2 | 26 | 2 |
| jEdit4.2 | 1 | 15 | 0 | 2 | 29 | 2 |
| jEdit4.3 | 2 | 2 | 0 | 4 | 5 | 0 |
| Cocoon | 1 | 10 | 3 | 2 | 25 | 11 |
| Derby | 5 | 24 | 0 | 10 | 69 | 1 |
| Lucene | 5 | 15 | 4 | 12 | 92 | 8 |
| OpenJPA | 3 | 22 | 0 | 9 | 64 | 1 |
| Eclipse | 5 | 33 | 1 | 19 | 92 | 2 |
| **Overall** | **23** | **143** | **9** | **61** | **413** | **27** |

**Table 33** Data analysis results on effectiveness and rank scores of abstraction level 3 (Sim4).

| | Effectiveness | | | Rank | | |
|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 1 | 7 | 0 | 3 | 9 | 0 |
| aTunes | 3 | 12 | 1 | 5 | 23 | 2 |
| jEdit4.2 | 4 | 12 | 0 | 7 | 24 | 2 |
| jEdit4.3 | 1 | 3 | 0 | 2 | 7 | 0 |
| Cocoon | 3 | 8 | 3 | 6 | 21 | 11 |
| Derby | 6 | 23 | 0 | 18 | 61 | 1 |
| Lucene | 7 | 13 | 4 | 24 | 80 | 8 |
| OpenJPA | 5 | 20 | 0 | 13 | 60 | 1 |
| Eclipse | 7 | 31 | 1 | 23 | 88 | 2 |
| **Overall** | **37** | **129** | **9** | **101** | **373** | **27** |

**Table 34** Data analysis results on effectiveness and rank scores of abstraction level 4 (Sim5).

| | Effectiveness | | | Rank | | |
|---|---|---|---|---|---|---|
| **System** | **#Wins** | **#Loses** | **#Draws** | **#Wins** | **#Loses** | **#Draws** |
| Art of Illusion | 2 | 6 | 0 | 2 | 10 | 0 |
| aTunes | 2 | 13 | 1 | 4 | 24 | 2 |
| jEdit4.2 | 4 | 12 | 0 | 6 | 25 | 2 |
| jEdit4.3 | 0 | 4 | 0 | 1 | 8 | 0 |
| Cocoon | 2 | 9 | 3 | 4 | 23 | 11 |
| Derby | 7 | 22 | 0 | 20 | 59 | 1 |
| Lucene | 6 | 14 | 4 | 20 | 84 | 8 |
| OpenJPA | 7 | 18 | 0 | 18 | 55 | 1 |
| Eclipse | 8 | 30 | 1 | 26 | 85 | 2 |
| **Overall** | **38** | **128** | **9** | **101** | **373** | **27** |

## References


Antoniol G, Penta MD, Harman M (2004) A robust search-based approach to project management in the presence of abandonment, rework, error and uncertainty. In: Proceedings of the Software Metrics, 10th International Symposium, IEEE Computer Society, Washington, DC, USA, METRICS '04, pp 172–183, DOI 10.1109/METRICS.2004.4, URL http://dx.doi.org/10.1109/METRICS.2004.4

Anvik J, Hiew L, Murphy GC (2005) Coping with an open bug repository. In: ETX, pp 35–39

Arcuri A, Fraser G (2011) On parameter tuning in search based software engineering. Search based software engineering pp 33–47

Aslam JA, Montague M (2001) Models for metasearch. In: SIGIR 2001: Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, September 9-13, 2001, New Orleans, Louisiana, Usa, pp

275–284
Asuncion HU, Asuncion AU, Taylor RN (2010) Software traceability with topic modeling. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1, ACM, pp 95–104
Binkley D, Lawrie D (2014) Learning to rank improves ir in se. In: IEEE International Conference on Software Maintenance and Evolution, pp 441–445
Blei DM, Lafferty JD (2007) Correction: A correlated topic model of science. Statistics 1(1):17–35
Blei DM, Ng AY, Jordan MI (2003) Latent dirichlet allocation. Journal of machine Learning research 3(Jan):993–1022
Canfora G, De Lucia A, Di Penta M, Oliveto R, Panichella A, Panichella S (2013) Multi-objective cross-project defect prediction. In: Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on, IEEE, pp 252–261
Cliff N (2014) Ordinal methods for behavioral data analysis. Psychology Press
Dit B, Revelle M, Poshyvanyk D (2013) Integrating information retrieval, execution and link analysis algorithms to improve feature location in software. Empirical Software Engineering 18(2):277–309
Eaddy M, Aho AV, Antoniol G, Guéhéneuc YG (2008) Cerberus: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis. In: Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on, IEEE, pp 53–62
Fox EA, Koushik MP, Shaw JA, Modlin R, Rao D (1992) Combining evidence from multiple searches. In: Text Retrieval Conference, pp 319–328
Gay G, Haiduc S, Marcus A, Menzies T (2009) On the use of relevance feedback in ir-based concept location. In: Software Maintenance, 2009. ICSM 2009. IEEE International Conference on, IEEE, pp 351–360
Gold N, Harman M, Li Z, Mahdavi K (2006) Allowing overlapping boundaries in source code using a search based approach to concept binding. In: Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on, IEEE, pp 310–319
Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning, 1989. Reading: Addison-Wesley
Griffiths TL, Steyvers M (2004) Finding scientific topics. Proceedings of the National Academy of Sciences 101(suppl 1):5228–5235
Haiduc S, Bavota G, Marcus A, Oliveto R, De Lucia A, Menzies T (2013) Automatic query reformulations for text retrieval in software engineering. In: Proceedings of the International Conference on Software Engineering, IEEE Press, ICSE, pp 842–851
Harman M, Jones BF (2001) Search-based software engineering. Information and Software Technology 43(14):833–839
Harman M, Mansouri SA, Zhang Y (2012) Search-based software engineering: Trends, techniques and applications. ACM Computing Surveys (CSUR) 45(1):11
Hindle A, Barr ET, Su Z, Gabel M, Devanbu P (2012) On the naturalness of software. In: Software Engineering (ICSE), 2012 34th International Conference on, IEEE, pp 837–847

Holland JH (1975) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. U Michigan Press

Hotho A, Maedche A, Staab S (2002) Ontology-based text document clustering. KI 16(4):48–54

Joseph EA (1997) Combination of multiple searches. Int J Uncertain Fuzziness Knowl- Based Syst

Kleinberg, Jon, Tomkins, Andrew (1999) Applications of linear algebra in information retrieval and hypertext analysis

Le TDB, Oentaryo RJ, Lo D (2015) Information retrieval and spectrum based bug localization: Better together. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 579–590

Le XBD, Le QL, Lo D, Goues CL (2016a) Enhancing automated program repair with deductive verification. In: ICSME

Le XD, Lo D, Le Goues C (2016b) History driven program repair. In: IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016, pp 213–224

Le Goues C, Nguyen T, Forrest S, Weimer W (2012) Genprog: A generic method for automatic software repair. Software Engineering, IEEE Transactions on 38(1):54–72

Li Z, Harman M, Hierons RM (2007) Search algorithms for regression test case prioritization. IEEE Trans Softw Eng 33(4):225–237, DOI 10.1109/TSE.2007.38, URL http://dx.doi.org/10.1109/TSE.2007.38

Liu D, Xu S (2007) A combined concept location method for java programs. In: Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International, IEEE, vol 2, pp 29–42

Lohar S, Amornborvornwong S, Zisman A, Cleland-Huang J (2013) Improving trace accuracy through data-driven configuration and composition of tracing features. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ACM, pp 378–388

Lucia L, Lo D, Xia X (2014) Fusion fault localizers. In: Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, ACM, pp 127–138

Mancoridis S, Mitchell BS, Chen Y, Gansner ER (1999) Bunch: A clustering tool for the recovery and maintenance of software system structures. In: Proceedings of the IEEE International Conference on Software Maintenance, IEEE Computer Society, Washington, DC, USA, ICSM '99, pp 50–, URL http://dl.acm.org/citation.cfm?id=519621.853406

Manning C, Raghavan P, Schutze H (2008) Introduction to Information Retrieval. Cambridge

Marcus A, Maletic JI (2003) Recovering documentation-to-source-code traceability links using latent semantic indexing. In: ICSE 2003

Moreno L, Bandara W, Haiduc S, Marcus A (2013) On the relationship between the vocabulary of bug reports and source code. In: Proceedings of International Conference on Software Maintenance, pp 452–455

Oliveto R, Gethers M, Poshyvanyk D, De Lucia A (2010) On the equivalence of information retrieval methods for automated traceability link recovery. In: Program Comprehension (ICPC), 2010 IEEE 18th International Conference on, IEEE, pp 68–71

Panichella A, Dit B, Oliveto R, Di Penta M, Poshyvanyk D, De Lucia A (2013) How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In: Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, pp 522–531

Panichella A, Dit B, Oliveto R, Penta MD, Poshyvanyk D, Lucia AD (2016) Parameterizing and assembling ir-based solutions for software engineering tasks using genetic algorithms. In: SANER

Poshyvanyk D, Gueheneuc YG, Marcus A, Antoniol G, Rajlich VC (2007) Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. Software Engineering, IEEE Transactions on 33(6):420–432

Rao S, Kak AC (2011) Retrieval from software libraries for bug localization: a comparative study of generic and composite text models. In: MSR

Robillard MP, Murphy GC (2007) Representing concerns in source code. ACM Trans Softw Eng Methodol 16(1)

Rousseeuw PJ, Kaufman L (1990) Finding Groups in Data. Wiley Online Library

Salton G, Harman D (2003) Information retrieval p 777

Sander J, Ester M, Kriegel HP, Xu X (1998) Density-based clustering in spatial databases: The algorithm gdbscan and its applications. Data mining and knowledge discovery 2(2):169–194

Scanniello G, Marcus A (2011) Clustering support for static concept location in source code. In: Program Comprehension (ICPC), 2011 IEEE 19th International Conference on, IEEE, pp 1–10

Scanniello G, Marcus A, Pascale D (2015) Link analysis algorithms for static concept location: an empirical assessment. Empirical Software Engineering 20(6):1666–1720

Shaw JA, Fox EA (2014) Combination of multiple searches. IEEE Transactions on Multimedia 16(1):277–282

Sun C, Lo D, Khoo SC, Jiang J (2011) Towards more accurate retrieval of duplicate bug reports. In: ASE, pp 253–262

Thomas SW (2011) Mining software repositories using topic models. In: Proceedings of the 33rd International Conference on Software Engineering, ACM, pp 1138–1139

Tonella P (2004) Evolutionary testing of classes. In: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis, ACM, New York, NY, USA, ISSTA '04, pp 119–128, DOI 10.1145/1007512.1007528, URL http://doi.acm.org/10.1145/1007512.1007528

Wallach HM, Mimno DM, McCallum A (2009) Rethinking lda: Why priors matter. In: Advances in neural information processing systems, pp 1973–1981

Wang S, Lo D (2014) Version history, similar report, and structure: Putting them together for improved bug localization. In: Proceedings of the 22nd International Conference on Program Comprehension, ACM, pp 53–63

Wang S, Lo D, Jiang L, Lucia, Lau HC (2011a) Search-based fault localization. In: Alexander P, Pasareanu CS, Hosking JG (eds) ASE, IEEE, pp 556–559

Wang S, Lo D, Xing Z, Jiang L (2011b) Concern localization using information retrieval: An empirical study on linux kernel. In: WCRE 2011

Wang S, Lo D, Lawall J (2014) Compositional vector space models for improved bug localization

Wang T, Harman M, Jia Y, Krinke J (2013) Searching for better configurations: a rigorous approach to clone evaluation. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ACM, pp 455–465

Wilcoxon F (1945) Individual comparisons by ranking methods. Biometrics bulletin 1(6):80–83

Wu S (2012) Data fusion in information retrieval. Adaptation Learning & Optimization 36(2):2997C3006

Xia X, Lo D (2017) An effective change recommendation approach for supplementary bug fixes. Automated Software Engineering 24(2):455–498

Xia X, Lo D, Wang X, Zhang C, Wang X (2014) Cross-language bug localization. In: Proceedings of the 22nd International Conference on Program Comprehension, ACM, pp 275–278

Xia X, Lo D, Wang X, Zhou B (2015) Dual analysis for recommending developers to resolve bugs. Journal of Software Evolution & Process 27(3):195–220

Xia X, Lo D, Pan SJ, Nagappan N, Wang X (2016a) Hydra: Massively compositional model for cross-project defect prediction. IEEE Transactions on software Engineering

Xia X, Lo D, Wang X, Yang X (2016b) Collective personalized change classification with multiobjective search. IEEE Transactions on Reliability

Xia X, Lo D, Ding Y, Al-Kofahi JM, Nguyen TN, Wang X (2017) Improving automated bug triaging with specialized topic model. IEEE Transactions on Software Engineering 43(3):272–297

Xuan J, Monperrus M (2014) Learning to combine multiple ranking metrics for fault localization. In: IEEE International Conference on Software Maintenance and Evolution, pp 191–200

Ye X, Bunescu R, Liu C (2014) Learning to rank relevant files for bug reports using domain knowledge. In: ACM Sigsoft International Symposium on Foundations of Software Engineering, pp 689–699

Zhang Y, Lo D, Xia X, Duy TDB, Scanniello G, Sun J (2016) Inferring links between concerns and methods with multi-abstraction vector space model. In: IEEE International Conference on Software Maintenance and Evolution ICSME, pp 110–121

Zhao W, Zhang L, Liu Y, Sun J, Yang F (2006) Sniafl: Towards a static noninteractive approach to feature location. ACM Transactions on Software Engineering and Methodology (TOSEM) 15(2):195–226

Zhou J, Zhang H, Lo D (2012) Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In: 2012 34th International Conference on Software Engineering (ICSE), IEEE, pp 14–24