Multi-Abstraction Concern Localization

Tien-Duy B. Le, Shaowei Wang, and David Lo
School of Information Systems,
Singapore Management University, Singapore
{btdle.2012,shaoweiwang.2010,davidlo}@smu.edu.sg

***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 need to be changed to address 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 *multi-abstraction* concern localization. A code unit and a textual description is represented at multiple abstraction levels. Similarity of a textual description and a code unit, is now made by considering all these abstraction levels. We have evaluated our solution on AspectJ bug reports and feature requests from the iBugs benchmark dataset. The experiment shows that our proposed approach outperforms a baseline approach, in terms of Mean Average Precision, by up to 19.36%.**

I. 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 [1]. 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.

A number of approaches have been proposed to link bug reports and feature requests to the corresponding code units, e.g., [5], [12]. The bug reports and feature requests could be viewed as *concerns*[1] and the linking process 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* [5], [12]. A textual document (i.e., feature request, bug report, or code unit) could be represented as a set of words that appear in it. Alternatively, a natural language processing technique, referred to as topic modeling, e.g., [2], 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 their set of topics). The code units that are most similar to the input concerns are output to the end user.

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, Eindhoven, can be abstracted to North Brabant, Netherlands, Western Europe, European Continent, Earth, and so on. Two documents might not share the same word "Eindhoven" but they might be about the same province (i.e., North Brabant), the same country (i.e., Netherlands), the same region (i.e., Western Europe), 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 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.

In the literature, vector space modeling (VSM) has been shown to outperform many other information retrieval (IR)-based techniques for concern localization [9], [12]. In this paper, we extend VSM to consider multi abstraction levels. We refer to the resultant model as multi-abstraction VSM (VSM^{MA}). We evaluate our approach on the iBugs dataset [3] which contains a few hundred of AspectJ concerns (i.e., bug reports and feature requests) and their corresponding code units. To demonstrate that the proposed multi-abstraction concept works, we compare our approach with the original VSM. The experiment results show that, in terms of mean average precision (MAP) [4], our multi-abstraction approach can outperform the original VSM by 19.36%.

Our contributions are as follows:

1) We propose multi-abstraction concern localization. We represent a document (i.e., a code unit, bug report, or

[1] A concern is a concept, requirement, feature, or property related to a software system [10]. 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.

1063-6773/13 $26.00 © 2013 IEEE
DOI 10.1109/ICSM.2013.48

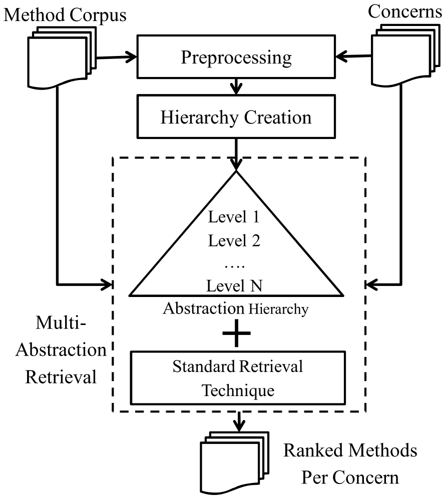

Fig. 1. Overall Framework of Multi-Abstraction Concern Localization

feature request) at multiple abstraction levels.

2) We propose a technique that leverages multiple topic models to capture representations of documents at different abstraction levels. Our technique then uses these representations to compute the similarity between a concern and a code unit.
3) We have evaluated our approach on hundreds of AspectJ concerns from the iBugs dataset. Our experiment shows that our proposed multi-abstraction concept works. In terms of MAP, our proposed approach can outperform the original VSM by 19.36%.

The structure of the paper is as follows. In Section II, we present the overall framework of multi-abstraction concern localization. In Section III, we discuss our multi-abstraction approach namely Multi-Abstraction VSM (VSM^{MA}). We present our experimental results in Section IV. We review related work in Section V. We finally conclude and mention future work in Section VI.

II. Overall Framework

A. Overview

Figure 1 presents the overall framework of multi-abstraction concern localization. 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 that appear in the source code of the method and comments (Java or Javadoc comments) that appear in or written for the method. *Concerns* is 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 three processing steps: *Preprocessing*, *Hierarchy Creation*, and *Multi-Abstraction Retrieval*. The purpose of the *Preprocessing* step is to convert methods and bug reports into a standard representation, i.e., a bag of words. The resultant bags of words are then input to the *Hierarchy Creation* step. The *Hierarchy Creation* step applies a topic modeling technique a number of times to construct the *abstraction hierarchy*. The *abstraction hierarchy* is a collection of topic models with various number of topics. Each topic model is a level in the hierarchy. The *abstraction hierarchy* is a part of the *Multi-Abstraction Retrieval* step. In this step, we enhance *standard retrieval techniques* by leveraging the *abstraction hierarchy*. 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 first two processing steps in the following subsections. Section III discusses the multi-abstraction retrieval step in more detail.

B. Preprocessing Step

In this step, we first remove common Java keywords such as *public*, *private*, *class*, *extends*, etc., as well as punctuation marks and special symbols. These words are deemed useless for linking concerns and code units (i.e., methods) as either they appear in most documents or they carry little meaning. Thus, we only retain some word tokens and number literals.

We then break identifiers into word tokens by assuming that identifiers follow Camel casing convention which is the naming convention adopted by most Java programs. Following the Camel casing convention, for every class name, each word starts with a capital letter; for other identifiers, the second and subsequent words start with a capital letter. We use this convention to break an identifier into word tokens. We perform this step to standardize word tokens that are used 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 create a bag (i.e., a multi-set) of words for each concern and method.

C. Hierarchy Creation Step

In the hierarchy creation step, we apply a topic modeling algorithm a number of times. We use Latent Dirichlet Allocation (LDA) which is a popular topic modeling algorithm [2]. LDA accepts as input the number of topics K and a set of documents [3] (in bag of words representation). It produces the following:

1) K topics, where each topic is a distribution of words.
2) For each document d and each topic t, LDA assigns a probability of topic t to appear in document d.
3) For each word w in document d, LDA assigns a topic to w. This topic is an abstraction of the word.

Each application of LDA creates a topic model with K topics. This topic model forms an abstraction level. We repeat

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

[3] We set the other LDA parameters to their default values.

this step L times to create L abstraction levels. 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*.

III. Multi-Abstraction Retrieval

In this section, 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) [4] is often used to assign weights to words. 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)):

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

In the above equation, $f(w, d)$ 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 [4].

In Multi-Abstraction VSM (VSM^{MA}), 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 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: *"suppress"*(3), *"warning"*(2), *"pointcut"*(2), *"aj"*(2), *"advice"*(1), *"lint"*(1), *"require"*(1). We also have two methods m_1 and m_2. Each of them contains 5 words: m_1 =\{*"suppress"(7), "warning"(4), "pointcut"(3), "lint"(7), "require"(1)*\} and m_2 =\{*"suppress"(10), "warning"(10), "aj"(5), "advice"(4), "lint"(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:

$$V_{br} = [1.62, 1.291.43, 1.43, 0.90, 0.81, 0.90, \mathit{0.26}, \mathit{0.72}, \mathit{0.02}]$$
$$V_{m_1} = [2.44, 1.89, 1.81, 0.00, 0.00, 2.44, 0.90, \mathit{0.00}, \mathit{0.99}, \mathit{0.00}]$$
$$V_{m_2} = [2.81, 2.81, 0.00, 2.33, 2.10, 2.28, 0.00, \mathit{0.57}, \mathit{0.43}, \mathit{0.00}]$$

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 rounded probabilities $\theta_{d,t_i}^{H_1}$ of topics 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.

IV. Experiments & Analysis

We use AspectJ concerns (i.e., bug reports and feature requests) from the iBugs dataset [3]. In iBugs, there are 350 AspectJ faulty versions, but some relevant methods in 65 versions cannot be found in the AspectJ codes that are included in the iBugs dataset. Thus, we exclude 65 concerns corresponding to these 65 versions. For each concern, we have its description, along with methods that are responsible for it, i.e., the method is changed to address the concern.

We measure effectiveness in terms of *mean average precision* (MAP) [4]. MAP has been used in past studies, e.g., [9]. A retrieval technique returns a sorted list of documents (i.e., methods) given a query (i.e., concern). The MAP of retrieval results corresponding to a set of queries (i.e., concerns) Q to retrieve relevant documents (i.e., methods) from a document corpus D is:

$$MAP(Q, D) = \frac{\sum_{q \in Q} AvgP(q, D)}{|Q|} \qquad (1)$$

$$AvgP(q, D) = \frac{\sum_{k=1}^{|D|} P@k \times rel(k)}{\text{Total number of relevant methods}}$$

TABLE I
ABSTRACTION HIERARCHIES USED IN THE EXPERIMENTS

Hierarchies	Number of Topics
H_1	50
H_2	50, 100
H_3	50, 100, 150
H_4	50, 100, 150, 200

TABLE II
EFFECTIVENESS OF MULTI-ABSTRACTION VSM OVER STANDARD VSM

	MAP	Improvement
Baseline (i.e., VSM)	0.0669	0%
H_1	0.0715	6.82%
H_2	0.0777	16.11%
H_3	0.0787	17.65%
H_4	0.0799	19.36%

In the above equation, $AvgP(q, D)$ is the *average precision* for query q. $P@k$ is the precision at k defined as the proportion of relevant methods among the top-k methods in the retrieval results. Also, $rel(k)$ is a function that returns 1 if the method returned at position k is relevant to the concern, and 0 otherwise.

We experiment with the 4 hierarchies H_1, H_2, H_3, and H_4 of heights 1, 2, 3, and 4 respectively (listed in Table I). The number of topics in the topic model(s) of H_1 is 50, H_2 are 50 and 100, H_3 are 50, 100, and 150, and H_4 are 50, 100, 150, and 200. In this preliminary study, we arbitrarily decide the hierarchy heights and the number of topics.

Our experiment results are shown in Table II. The table shows that for Multi-Abstraction VSM, for all hierarchy settings (H_1, H_2, H_3, and H_4), the performance is better than that of the baseline (standard VSM). Moreover, the MAP improvement for H_4 is 19.36%. Furthermore, we note that the MAP is improved when the height of the abstraction hierarchy is increased from 1 (H_1) to 4 (H_4). Table III shows the number of concerns where the improvements in the *average precision* ($AveP$) are within a particular range for H_1 to H_4. We note that for the majority of the concerns the improvements are positive. For H_4, the improvements are positive for 222 out of the 285 concerns (77.89%).

V. RELATED WORK

A number of past studies have employed various text retrieval techniques for concern localization [5], [12], [14]. Wang et al. [12] evaluate 10 information retrieval techniques and discover that VSM has the best performance. Rao and Kak also investigate the use of LDA with VSM [9]. 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 VSM. Aside from text, other sources of information, e.g., execution traces [8], have been used to aid concern localization. Our technique does not consider execution traces since most bug reports do not come with execution traces [11].

TABLE III
NUMBER OF CONCERNS WITH VARIOUS $AveP$ IMPROVEMENTS

Improvement (p)	Number of Concerns			
	H_1	H_2	H_3	H_4
$p < -10\%$	21	27	30	30
$-10\% \leq p < 0\%$	25	22	25	22
$p = 0\%$	18	14	12	11
$0\% < p \leq 10\%$	113	64	42	41
$p > 10\%$	108	158	176	181

Petrenko and Rajlich proposes an impact analysis approach which leverages program dependencies at *multiple* granularities (i.e., classes, class members, and code fragments) [7]. In this work, we address a different problem.

VI. CONCLUSION AND FUTURE WORK

In this study, we propose multi-abstraction concern localization which combines a hierarchy of topic models with VSM. Our experiments on 285 AspectJ concerns shows that we can improve MAP of VSM by up to 19.36%.

In the future, we plan to perform a deeper analysis on cases where our multi-abstraction approach does not work well. Also, we want to extend our study by experimenting with different numbers of topics in each level of the hierarchy, different hierarchy heights (> 4), and different topic models. We also want to analyze the effect of document lengths on the effectiveness of the proposed approach for different number of topics and hierarchy heights. Furthermore, we plan to experiment with Panichella et al.'s method [6] to infer good LDA configurations, and leverage other advanced text mining solutions, e.g., paraphrase detection [13], to further improve concern localization results.

REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy, "Coping with an open bug repository," in *ETX*, 2005, pp. 35–39.

[2] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.

[3] V. Dallmeier and T. Zimmermann, "Extraction of bug localization benchmarks from history," in *ASE*, 2007, pp. 433–436.

[4] C. Manning, P. Raghavan, and H. Schutze, *Introduction to Information Retrieval*. Cambridge, 2008.

[5] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *ICSE 2003*.

[6] A. Panichella, B. Dit, R. Oliveto, M. D. Penta, D. Poshyvanyk, and A. D. Lucia, "How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms," in *ICSE*, 2013.

[7] M. Petrenko and V. Rajlich, "Variable granularity for improving precision of impact analysis," in *ICPC*, 2009, pp. 10–19.

[8] D. Poshyvanyk, Y.-G. Gueheneuc, A. Marcus, G. Antoniol, and V. Rajlich, "Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval," *TSE*, 2007.

[9] S. Rao and A. C. Kak, "Retrieval from software libraries for bug localization: a comparative study of generic and composite text models," in *MSR*, 2011.

[10] M. P. Robillard and G. C. Murphy, "Representing concerns in source code," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 1, 2007.

[11] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in *ASE*, 2011, pp. 253–262.

[12] S. Wang, D. Lo, Z. Xing, and L. Jiang, "Concern localization using information retrieval: An empirical study on linux kernel," in *WCRE 2011*.

[13] X. Wang, D. Lo, J. Jiang, L. Zhang, and H. Mei, "Extracting paraphrases of technical terms from noisy parallel software corpora," in *ACL/IJCNLP*, 2009.

[14] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports," in *ICSE*, 2012, pp. 14–24.