Concern Localization using Information Retrieval: An Empirical Study on Linux Kernel

Shaowei Wang[1], David Lo[1], Zhenchang Xing[2], and Lingxiao Jiang[1]
[1]School of Information Systems, Singapore Management University
[2]School of Computing, National University of Singapore
Email: {shaoweiwang.2010,davidlo,lxjiang}@smu.edu.sg, xingzc@comp.nus.edu.sg

***Abstract*—Many software maintenance activities need to find *code units* (functions, files, etc.) that implement a certain *concern* (features, bugs, etc.). To facilitate such activities, many approaches have been proposed to automatically link code units with concerns described in natural languages, which are termed as *concern localization* and often employ Information Retrieval (IR) techniques. There has not been a study that evaluates and compares the effectiveness of *latest* IR techniques on a *large* dataset. This study fills this gap by investigating ten IR techniques, some of which are new and have not been used for concern localization, on a Linux kernel dataset. The Linux kernel dataset contains more than 1,500 concerns that are linked to over 85,000 C functions. We have evaluated the effectiveness of the ten techniques on recovering the links between the concerns and the implementing functions and ranked the IR techniques based on their precisions on concern localization.**

***Keywords*-concern localization; information retrieval; Linux kernel; mean average precision;**

I. Introduction

To help with program comprehension and software maintenance, various techniques have been proposed to link units of program code (e.g. functions, files) with a particular concern, which are termed as *concern localization* (e.g. bug localization [22], [29], feature location [1], [14], [15], [34]). Many studies on concern localization use Information Retrieval (IR) techniques to recover the links between concerns and code units [16], [24], [26]–[29], [32], [33].

However, most previous studies are evaluated with a relatively small numbers of concerns and small to medium sized programs. Would IR-based concern localization approaches remain effective for a large scale software system with a large number of concerns?

On another related angle, the information retrieval community has proposed many alternative IR techniques. For example, topic modeling [5], [6] has been shown to perform well on textual corpora in both natural languages and programming languages [30]. Would these techniques, some of which have not been used before for concern localization, achieve better results than older, simpler IR techniques?

In this paper, we attempt to answer the above questions by performing an empirical study of the effectiveness of various IR techniques on a large software system with a large number of concerns. Specifically, we study the Linux kernel that consists of more than 1,500 features (i.e. concerns to be localized) and over 85,000 C functions (i.e. code units of interest in our study) in about 10 million lines of code.

We investigate ten IR techniques, including Vector Space Model (VSM) [23], Smoothed Unigram Model (SUM) [23], [29], Non-negative Matrix Factorization (NMF) [20], Latent Semantic Indexing (LSI) [3], [13], probabilistic LSI (pLSI) [17], Latent Dirichlent Allocation(LDA) [7], hierarchical LDA (hLDA) [5], Laplacian pLSI (LapPLSI) [25], Locally-consistent Topic Modeling (LTM) [9], and Discriminative Topic Modeling (DTM) [18]. The last four are new and have not been used before for concern localization.

We measure the effectiveness of the IR techniques with Mean Average Precision (MAP). We also perform statistical significance tests to investigate whether the differences in effectiveness between each pair of techniques are statistically significant, and arrange the IR techniques into a partial order based on their effectiveness measures.

II. Linux Kernel

The Linux kernel (http://www.kernel.org) was originally developed for 32-bit x86-based PCs, but has evolved into a software product line [2], which consists of thousands of features that can be configured to generate specific kernel products for vast combinations of architectures, subsystems, device drivers, etc.

In this study, we use a stable version 2.6.38 of the Linux kernel released on March 15th, 2011. We filter out many features having no implementing functions (e.g., some features are only linked to certain variable declarations), and obtain a final dataset containing 1,561 features linked to 85,466 non-empty implementing functions.

We extract the followings from the Linux kernel: 1) features with textual descriptions that can be used as queries for concern localization using IR techniques, 2) C functions that form corpus for concern localization, and 3) ground truth links between features and their implementing functions for evaluating the effectiveness of various IR techniques.

Extracting Features as Queries. The Linux kernel manages features using a feature modeling language (Kconfig) and its accompanied configuration tool [2]. Each defined feature has a configuration symbol, a short prompt, a free-form textual description, together with other information (such as datatype and default value of the feature, dependencies between features), and can be extracted from the Kconfig model.

Building C Function Corpus. We collect all C functions in the source code as the candidates linking to each feature. For each C function in the source code, we transform its function body, including the C code and comments, excluding macros, to a bag of words so that IR techniques can be applied. This step is done with a customized C parser.

Establishing Ground Truth of Feature-Function Links. Note that the features in the Linux kernel are configured via a set of feature symbol-value pairs. These feature symbol-value pairs are used as macros in the makefiles and source code to control which directories, files, and code blocks would be compiled. Such correspondence between the feature symbol and the macros controlling which code gets compiled allows us to automatically establish the ground truth links between features and their implementing functions.

For example, by searching for the uses of the symbol `CONFIG_SWAP` in the makefiles and source code, we can infer that the swap feature of the Linux kernel has 120 implementing functions, including the two functions $inactive\_anon\_is\_low()$ and $inactive\_anon\_is\_low\_global()$ defined in $vmscan.c$, 77 functions defined in $page\_io.c$, $swap\_state.c$, $swapfile.c$, and $thrash.c$, and 41 functions defined in other source files.

Note that, in fact, we do not need IR techniques to establish the link between features and code for the Linux kernel. However, the availability of this ground-truth allows us to systematically evaluate the effectiveness of IR techniques for concern localization at a large scale.

III. IR Techniques

In this section, we describe in brief the ten IR techniques that we investigate. The ten IR techniques could be categorized into four groups: vector space model, language modeling, matrix factorization, and topic modeling.

Vector Space Model. Such models represent each document d as a vector. Each value in the vector corresponds to the weight of a term t in d. The weight is often calculated based on *term frequency* (i.e., the frequency of the term in the document) and *inverse document frequency* (i.e., the reciprocal of the number of documents containing the term). We use a typical formula in information retrieval studies for the weight of t in d:

$$weight(t, d) = tf(t, d) \times idf(t, D)$$

where t, d, D, $tf(t)$, and $idf(t)$ correspond to a term, a document, a set of documents, the term frequency of t in d, and the inverse document frequency of t in D, respectively.

Given a query containing the short prompt and the description of a Linux kernel feature, and a set of documents which is comprised of the C code and comments in functions in the kernel source code, we convert each of them to a vector representing the weights of the terms in it. We then compute the similarities between the query vector and all document vectors; the similarity between two vectors is calculated as the cosine distance between them [23]. The documents with high similarities with the query vector are returned as result.

Language Modeling. In this study we consider a simple model, Unigram Model, that is frequently used and has recently be shown to be effective in bug localization [29].

Matrix Factorization. A large matrix can be factorized into two smaller matrices in various ways for easier manipulation. In this study, we only evaluate NMF [20].

We use NMF for information retrieval as follows. We characterize a set of documents as a term-document matrix. This document matrix is then decomposed into a smaller term-feature matrix (not the same kind of features as the Linux kernel features, but the technical noun used in matrix factorization) and a feature-document matrix. The feature-document matrix describes a document as a distribution of features—for each document, each feature is given a weight. To compute the similarity of a document in a corpus to a given query, we compute the cosine distance of their representative weight vectors. Documents with sufficiently high similarity scores are returned as the query results.

Topic Modeling. Techniques in this category relate words to topic and can help to identify synonyms and different words related to a same topic.

Well-known topic modeling techniques include Latent Semantic Indexing (LSI) [3], [13], Latent Dirichlet Allocation (LDA) [7], and their extensions, such as probabilistic LSI (pLSI) [17], Laplacian Latent Semantic Indexing (LapPLSI) [8], [25], hierarchical LDA (hLDA) [5], Locally-consistent Topic Modeling (LTM) [9], and Discriminative Topic Modeling (DTM) [18]. These seven topic modeling techniques are used in our study. We simply consider a query and every document as a vector of weights. Then, cosine similarity scores are used again to measure the similarities between the query and documents, and the documents with high scores are returned as the query results.

IV. Concern Localization using IR

In this section, we describe the process of using IR techniques discussed in Section III for concern localization. A concern localization approach that leverages IR techniques could be broken down into two phases: extraction of bag-of-words representation, and retrieval of relevant code units implementing a concern. Fig. 1 shows the overall process.

Bag-of-words Extraction. We split the Linux kernel into a set of functions (i.e. code units of interest) and convert each function into a bag of words in three steps: tokenization, stop word (e.g. keywords) removal, and stemming. Each linux feature is also converted into a bag of words using the three steps, except that the stop words are the stop words in English instead of the keywords in C.

Code-Units Retrieval. We feed the bags of words to our information retrieval engine, which has mainly two steps:

- Model Construction. In this step, all bags of words are used to build either a vector space model, a language model, a factorized matrix, or a topic model. This step takes the most amount of time but is only performed once.

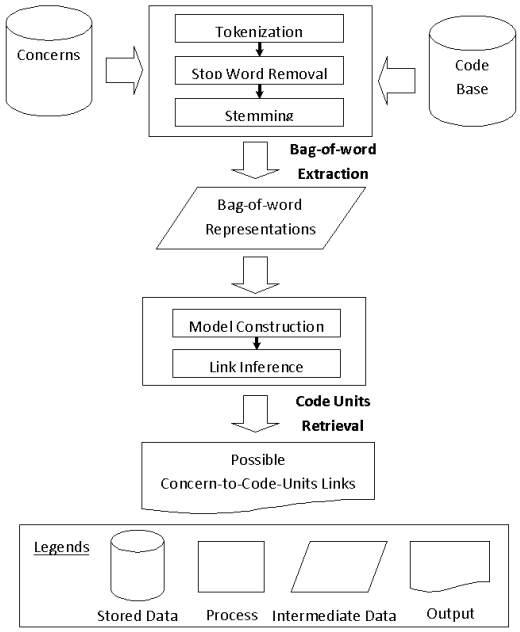

Fig. 1. Concern Localization using Information Retrieval.

- Link Inference. In this step, a concern is treated as a query to the IR engine. The engine would return a list of code units ordered based on their similarity scores against the query. This link inference step (a.k.a. the query step) could be repeated for any query.

When we finish the link inference step for every concern, we could compare all returned results with the ground truth in the Linux kernel (c.f. Section II) to evaluate the effectiveness of the IR techniques for concern localization.

V. Empirical Evaluation

In this section, we present our research questions, experimental settings, and results.

A. Research Questions

We are interested in understanding the effectiveness of IR techniques on concern localization at large. In particular, we aim to gain insights into the following research questions:

- Q1: Which IR technique performs better than others for concern localization?
- Q2: Do recently proposed, more sophisticated IR techniques outperform older, simpler ones?
- Q3: Is the effectiveness of IR-based concern localization on large datasets comparable to that on small to medium sized datasets?
- Q4: Compared to textual corpora in natural languages, do IR techniques perform well on software data?

B. Experimental Settings

Four of the ten IR techniques discussed in Section III were implemented in Java: VSM and SUM were implemented by ourselves, LDA was an implementation by Phan and Nguyen[1], and hLDA was from MALLET's homepage[2]. The other six IR techniques were implemented in Matlab: NMF was from IMM DTU[3] [19], LSI was implemented by ourselves, pLSI was from Peter Gehler's Code and Dataset Page[4], LapPLSI and LTM were from Deng Cai's webpage[5], and DTM was provided by its authors [18].

Our evaluation was performed on a PC with a 3.3GHz 2-core CPU and 4 GiB of memory running Window 7. And we use Mean Average Precision (MAP), a commonly used measure in IR community [23], to evaluate the effectiveness of IR techniques for concern localization.

Technique / Topic #	MAP			
	50	100	303	500
pLSI	0.0138	0.0213	0.0647	0.0853
LapPLSI	0.0147	0.0198	0.0617	0.0781
SUM	0.1520	0.1520	0.1520	0.1520
VSM	0.2626	0.2626	0.2626	0.2626
LDA	0.0077	0.0130	0.0223	0.0491
LSI	0.0179	0.0226	0.0378	0.0524
NMF	0.0123	0.0035	0.0364	O/M
hLDA	N/A	N/A	0.0203	N/A
LTM	O/M			
DTM	O/M			

TABLE I
Precision Comparison among all IR techniques with various numbers of topics.

C. Evaluation Results

A) Raw Results

There are many control parameters used in various IR techniques. A major parameter common to a number of techniques is the number of topics that should be used for model construction. In this study, we use 50, 100, 303, and 500 as the numbers of topics and perform the comparison accordingly. Note that 303 is used because it is the number automatically chosen by the 3-level hierarchical LDA, and thus we use 303 (instead of 300) for fair comparison with other IR techniques. We use default settings for other parameters.

Table I summarizes the effectiveness of all IR techniques used. Two of them, LTM and DTM, cannot complete because of out of memory ("O/M"). NMF ran out of memory for 500 number of topics. hLDA is not applicable ("N/A") to 50, 100, and 500 number of topics, because it automatically chooses a topic number for a corpus (in our case, 303). Each number indicates the MAP value for each technique; the higher the better. Fig. 2 shows the corresponding plots.

B) Research Questions Answered

Overall, we compare the effectiveness of the IR techniques based on MAP. Since the MAP of some IR techniques (e.g., NMF, LSI, pLSI, LDA, LapPLSI) depends on the number of topics used during link inference, we treat each instance of such techniques instantiated with a topic number as a different technique in our comparative evaluation. For example, we use pLSI(500) to denote the particular instance of pLSI that is instantiated with 500 topics.

[1]http://jgibblda.sourceforge.net/
[2]http://mallet.cs.umass.edu/
[3]http://cogsys.imm.dtu.dk/toolbox/
[4]http://people.kyb.tuebingen.mpg.de/pgehler/code/
[5]http://www.zjucadcg.cn/dengcai/LapPLSA/index.html

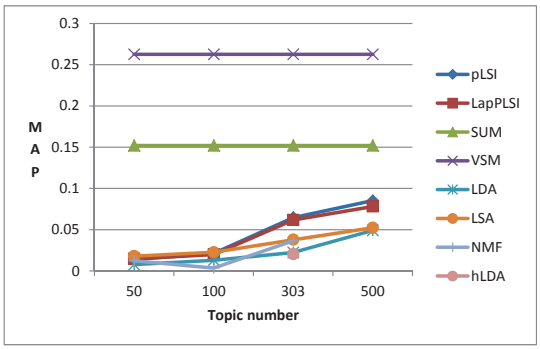

Fig. 2. MAP for eight IR techniques with topic number k = 50,100,303,500

1) Q1: Better Performing IR Techniques: We rank the IR techniques based on their MAP measures (c.f. Table I and Fig. 2), from high to low, as follows:

VSM > SUM > pLSI(500) > LapPLSI(500) >
pLSI(303) > LapPLSI(303) > LSI(500) > LDA(500) >
LSI(303) > NMF(303) > LSI(100) > LDA(303) >
pLSI(100) > hLDA(3-level) > LapPLSI(100) > LSI(50) >
LapPLSI(50) > pLSI(50) > LDA(100) > NMF(50) >
LDA(50) > NMF(100)

Since MAP is aggregated from the underlying precision measures for individual queries (which can be viewed as samples drawn from certain distributions), we employ the *Wilcoxon test* [10], [31] to compare each pair of the IR techniques using their underlying precision measures.

The results of the Wilcoxon tests on all pairs of the IR techniques can be represented as a partial order as follows, where some IR techniques have comparable effectiveness (i.e. their differences are not statistically significant) and are denoted by $\approx$:

VSM > SUM > pLSI(500) $\approx$ LapPLSI(500) >
pLSI(303) $\approx$ LapPLSI(303) > LSI(500) $\approx$ LDA(500) >
LSI(303) $\approx$ LapPLSI(100) $\approx$ pLSI(100) > NMF(303) >
LSI(100) $\approx$ LapPLSI(50) $\approx$ LDA(303) $\approx$ pLSI(50) >
LSI(50) $\approx$ LDA(100) > LDA(50) > hLDA(3-level) $\approx$
NMF(50) $\approx$ NMF(100)

Note that this partial order produced by Wilxocon tests is different from the total order based on the absolute MAP measures for the IR techniques. For example, MAP for LSI(100) and pLSI(100) are 0.0226 and 0.0213 respectively, but they are shifted as approximate ($\approx$) in the partial order. The biggest shifting is for hLDA, from a middle position in the total order to the bottom in the partial order, which implies that hLDA may perform badly for some queries.

2) Q2: Newer, More Sophisticated versus Older, Simpler IR Techniques: Based on the above partial order of the IR techniques, we have no reason to believe that newer or more sophisticated IR techniques would outperform older or simpler IR techniques. A similar observation was also noted by Rao and Kak in their study on bug localization using IR techniques [29]. We hypothesize that this phenomenon is mainly because our dataset (software systems) is quite different from datasets in natural languages for which the IR techniques are designed.

VSM is the best performer among all IR techniques, although it is older, and simpler than other techniques in a number of ways. Some of reasons may be: 1) VSM considers common words when calculating the similarity score between two documents; 2) VSM does not perform dimension reduction that is used in many other IR techniques to reduce probably irrelevant information which may be relevant for software corpora.

Compared with older IR techniques (e.g., LSI and pLSI), newer ones (e.g., LDA, hLDA, LapPLSI, LTM, and DTM) are more complex, but do not necessarily perform better either.

3) Q3: Large versus Small-Medium Datasets: To answer this question, we compare our results against Rao and Kak's results in [29]. Rao and Kak study the effectiveness of five IR techniques (VSM, SUM, CBDM, LSA, and LDA) on fault localization using iBUGS [11], [12]—a benchmark dataset containing several medium-size Java programs with about three hundreds bugs for fault localization.

There are differences between our results and theirs. In our study, VSM has the best retrieval precisions for concern localization, while in their study of fault localization, SUM has the best precisions. Furthermore, their study shows that LDA performs the worst among their five IR techniques, while our study shows that hLDA can perform even worse.

In spite of the differences, both our and their results suggest that IR techniques can be useful in general for either feature location or bug localization, and that simpler techniques (e.g., VSM and SUM) perform better than more sophisticated techniques (e.g., LSI, LDA, hLDA).

The best MAP score reported in [29] is between 0.14 to 0.15. In our study, the best MAP score is better, i.e., 0.26. Thus, the effectiveness of concern localization is not diminished when a large software system is analyzed.

4) Q4: Software Corpora versus Natural Language Corpora: Information retrieval techniques are typically applied on a corpus of articles in natural languages, while we apply the techniques on software corpora that have quite different textual appearance, syntactic structure, and semantics from articles in natural languages, which would affect the effectiveness of IR techniques for concern localization in software systems.

For example, using LDA, the MAP measure reported in [30] (a study of LDA on natural language articles) is between 25-30%, much higher than what we achieved in this study which is about 1-5%. Further work would be to investigate the underlying causes and design a customized IR technique that is suited for concern localization in large software systems.

VI. Related Work

Concern localization is often referred to as feature location, concept assignment, or bug localization in different work. A survey on feature location is available in [14]. We highlight some past studies in this section. Biggerstaff et al. [4] and Wilde et al. [32] pioneered the research on concept assignment and feature location. Software reconnaissance by Wilde and Scully [32] and its extensions (e.g., [33]) identify the parts of the program that implement a feature by comparing execution profiles or slices of two sets of carefully designed test cases.

Antoniol et al. [1] apply both a probabilistic and a vector space model to recover links between source code units and

free text documents (e.g., manual pages or requirements). Marcus and Maletic [24] use LSI for a similar purpose. The underlying assumption is that programmers use meaningful words for code units, and these words capture application-specific knowledge. Our study suggests that although this assumption is valid in general, it may be necessary to customize IR techniques so that they can perform similarly well on software data, compared to natural language corpora.

A recent work by Rao and Kak [29] describes a similar study that evaluates the effectiveness of various IR techniques on fault localization. Our earlier work [21] investigate the effectiveness of various association measures on fault localization. The subjects examined in this paper are different: our study considers a much larger software system with a larger set of concerns using IR techniques; we also investigate four latest IR techniques that have never been used for concern localization. Such comparative studies can help gain insights into the applicability and effectiveness of various techniques on various software engineering tasks.

VII. Conclusion and Future Work

In this study, we perform an empirical study on the effectiveness of many IR techniques for the purpose of concern localization in a large software system. We collect the Linux kernel dataset that contains more than 1,500 features linked to over 85,000 functions in about 10 million lines of code. We explore ten different IR techniques from four different groups: vector space model, language model, matrix factorization, and topic modeling, some of which are new and have not been used for concern localization.

Our evaluation results show that (1) older and simpler IR technique, such as VSM and SUM, can perform better than more recent or more complicated IR techniques, such as NMF, pLSI, LDA, hLDA, and LapPLSI; (2) the same IR techniques may perform differently on different software systems for different applications (e.g., feature location versus bug localization); (3) IR-based concern localization techniques work as well in a large software system as it is in small-to-medium sized software systems; and (4) existing IR techniques may perform worse on software corpora than on natural language textual corpora.

In the future, we plan to investigate the effects of the parameters used in the IR techniques on the precision. We also plan to investigate more large systems in different programming languages. It is also interesting to develop a customized IR technique that is specifically suited for concern localization.

Acknowledgements. We would like to thank all of the authors who make their implementations of NMF, pLSI, LDA, hLDA, LapPLSI, LTM, and DTM available for our use.

References

[1] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, "Recovering traceability links between code and documentation," in *TSE*, 2002.

[2] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki, "Variability modeling in the real: A perspective from the operating systems domain," in *ASE*, 2010, pp. 73–82.

[3] M. Berry, S. Dumais, and G. O'Brien, "Using linear algebra for intelligent information retrieval." in *SIAM Review*, 1995.

[4] T. Biggerstaff, B. Mitbander, and D. Webster, "The concept assignment problem in program understanding," in *ICSE*, 1993.

[5] D. Blei, T. Griffiths, and M. Jordan, "The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies." in *Journal of the ACM*, 2010.

[6] D. Blei and J. Lafferty., "Topic models," in *Text Mining: Classification, Clustering, and Applications*, 2009.

[7] D. Blei, A. Ng, and M. Jordan, "Latent dirichlet allocation," in *Journal of Machine Learning Research*, 2003.

[8] D. Cai, Q. Mei, J. Han, and C. Zhai, "Modeling hidden topics on document manifold," in *CIKM*, 2008, pp. 911–920.

[9] D. Cai, X. Wang, and X. He, "Probabilistic dyadic data analysis with local and global consistency," in *ICML*, 2009, pp. 105–112.

[10] G. W. Corder and D. I. Foreman, *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*. John Wiley and Sons Inc, 2009.

[11] V. Dallmeier and T. Zimmermann, *iBUGS: Bug Repositories*, http://www.st.cs.uni-saarland.de/ibugs/.

[12] ——, "Extraction of bug localization benchmarks from history," in *ASE*, 2007, pp. 433–436.

[13] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman, "Indexing by latent semantic analysis," in *Journal of the American Society for Information Science*, 1990.

[14] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: A taxonomy and survey," *JSME*, 2012.

[15] M. Eaddy, A. Aho, G. Antoniol, and Y.-G. Gueheneuc, "CERBERUS: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis," in *ICPC*, 2008.

[16] T. Eisenbarth, R. Koschke, and D. Simon, "Locating features in source code," in *TSE*, 2003.

[17] T. Hofmann, "Probabilistic latent semantic analysis," in *UAI*, 1999.

[18] S. Huh and S. E. Fienberg, "Discriminative topic modeling based on manifold learning," in *KDD*, 2010, pp. 653–662.

[19] T. Kolenda, S. Sigurdsson, O. Winther, L. K. Hansen, and J. Larsen, "DTU:Toolbox," http://isp.imm.dtu.dk/toolbox/.

[20] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *NIPS*, 2001.

[21] Lucia, D. Lo, L. Jiang, and A. Budi, "Comprehensive evaluation of association measures for fault localization," in *ICSM*, 2010, pp. 1–10.

[22] S. Lukins, N. Kraft, and E. Letha, "Source code retrieval for bug localization using latent dirichlet allocation." *WCRE*, 2008.

[23] C. Manning, P. Raghavan, and H. Schtze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[24] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *ICSE*, 2003, pp. 125–135.

[25] Q. Mei, D. Cai, D. Zhang, and C. Zhai, "Topic modeling with network regularization," in *WWW*, 2008, pp. 101–110.

[26] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. D. Lucia, "On the equivalence of information retrieval methods for automated traceability link recovery," in *ICPC*, 2010, pp. 68–71.

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

[28] D. Poshyvanyk and A. Marcus, "Combining formal concept analysis with information retrieval for concept location in source code," in *ICPC*, 2007, pp. 37–48.

[29] 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, pp. 43–52.

[30] X. Wei and W. Croft, "LDA-based document models for ad-hoc retrieval." in *SIGIR*, 2006.

[31] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[32] N. Wilde and M. C. Scully, "Software reconnaissance: Mapping program features to code," in *JSME*, 1995.

[33] W. Wong, J. Horgan, S. Gokhale, and K. Trivedi, "Locating program features using execution slices," in *ASSET*, 1999.

[34] W. Zhao, L. Zhang, Y. Liu, J. Sun, and F. Yang, "SNIAFL: Towards a static noninteractive approach to feature location," in *TOSEM*, 2006.