On the Use of Positional Proximity in IR-Based Feature Location

Emily Hill
Department of Computer Science
Montclair State University
Montclair, NJ, USA
hillem@mail.montclair.edu

Bunyamin Sisman, Avinash Kak
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN, USA
{bsisman, kak}@purdue.edu

***Abstract*—As software systems continue to grow and evolve, locating code for software maintenance tasks becomes increasingly difficult. Recently proposed approaches to bug localization and feature location have suggested using the *positional proximity* of words in the source code files and the bug reports to determine the relevance of a file to a query. Two different types of approaches have emerged for incorporating word proximity and order in retrieval: those based on ad-hoc considerations and those based on Markov Random Field (MRF) modeling. In this paper, we explore using both these types of approaches to identify over 200 features in five open source Java systems. In addition, we use positional proximity of query words within natural language (NL) phrases in order to capture the NL semantics of positional proximity. As expected, our results indicate that the power of these approaches varies from one dataset to another. However, the variations are larger for the ad-hoc positional-proximity based approaches than with the approach based on MRF. In other words, the feature location results are more consistent across the datasets with MRF based modeling of the features.**

***Index Terms*—feature location, source code search, software maintenance**

I. Introduction

As software systems continue to grow and evolve, locating code for software maintenance tasks becomes increasingly difficult. Feature or concern location techniques can be used to identify locations in source code that may be relevant to maintenance tasks [1], and numerous IR-based feature location approaches have been proposed [2], [3], [4], [5], [6], [7], [8], [9], [10]. Recently proposed approaches to bug localization and feature location have suggested using the *positional proximity* of words in the source code files and the bug reports to determine the relevance of a file to a query [11], [4]. In this paper, we take a systematic look at the impact different varieties of positional proximity information have on feature location effectiveness.

In this work, we interpret positional proximity within a window of terms as well as within natural language (NL) phrases. Specifically, we apply the formal Markov Random Field (MRF) framework to calculate how frequently query words occur within a window of terms in a method body. As a simple ad-hoc approximation to this technique, we also define a "window" to be a program structure statement delimited by semicolons or curly braces. Finally, we also account for positional proximity within NL phrases extracted from method calls. For instance, the method call cart.add(item) encodes the NL phrase "add item to cart". The final approach looks for occurrences of the query terms within a NL phrase extracted from a method call or signature.

Our work explores using both these types of approaches to identify over 200 features in five open source Java systems. As expected, the results indicate that the power of these approaches varies from one dataset to another. However, the variations are larger for the ad-hoc positional-proximity based approaches than with the approach based on MRF. In other words, the feature location results are more consistent across the datasets with MRF based modeling of the features.

II. Positional Proximity Approaches

A. Markov Random Fields (MRF)

In prior work, we have applied positional proximity using Markov Random Fields (MRF) to improve the effectiveness of bug localization [12]. MRF modeling was used previously by Metzler and Croft [13] as a means to improving the performance of IR algorithms for retrieval from text corpora. As presented in [12], [13], with MRF, you model the inter-term dependencies for retrievals by constructing a dependency graph G that contains one node for the method being evaluated for its relevance to the query, with the other nodes representing the query terms. Typically, we denote the node that stands for the method by m and the other nodes by the query terms $Q = \{q_1, q_2, ..., q_{|Q|}\}$. Assuming that such a graph has the cliques $C_1, C_2, .., C_K$, the joint distribution $P(Q, m)$ over the method m and the query terms is uniquely defined as

$$P(m, Q) \;=\; \frac{1}{Z} \prod_{k=1}^{K} \phi(C_k) \;\stackrel{rank}{=}\; \sum_{k=1}^{K} log(\phi(C_k)) \quad (1)$$

where $\psi(C_k) = log(\phi(C_k))$ is a potential function defined over the cliques of the graph and Z is the normalization constant. This formulation of the dependency between a method and the query terms can be used to assign retrieval scores to the methods with respect to a given query while taking into account the order and the proximity of the terms in the query and in the source code. This is done by assuming the Markov property that given a method, the probability of a term depends

only on the neighboring terms. As to what these neighboring terms are, that is dictated by the inter-term connectivities used in the graph.

If there exist no inter-term edges in the graph, that is tantamount to saying that we do not care about any dependencies between the terms. Referred to as the full-independence (FI) assumption, this is the same as the bag-of-words (BOW) assumption in IR. At the other end of the spectrum, we can assume that there is an edge between every pair of terms in the graph. That is referred to as the full-dependency (FD). Under the FD assumption, the relevancy degree of a method to a query is measured by the frequencies for all possible pairings of the terms in a query matching the frequencies for all similar pairings in the method.

There are obviously many possibilities between the FI and the FD assumptions, the most popular being the sequential-dependency (SD) modeling because its semantics best capture word order and proximity in a natural language and due to its computationally efficiency. The SD assumption is captured by assuming an edge between the consecutive pairs of nodes that represent the query terms in the graph. The SD assumption is a more general way of capturing the spatial proximity assumption used in [11].

With that general introduction to MRF modeling, for the discussion in this paper, m would stand for a method whose relevance is being evaluated vis-a-vis a given query Q. The FI and SD assumptions are shown diagrammatically in Figure 1 for the case of a query that has just three terms. With the FI variant, the graph consist of only 2-node cliques that contain a method and a query term. Associating the following potential function with each clique in Figure 1(a) results in the well-known Dirichlet Language Model (DLM):

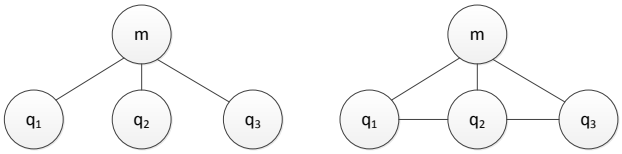
$$\psi_{FI}(q_i, m) = \lambda_{FI} log(\frac{tf(q_i, m) + \mu P(q_i|C)}{|m| + \mu}) \quad (2)$$

where we have used Dirichlet smoothing to account for the missing query terms in the method under consideration [14]. $P(q_i|C)$ denotes the probability of the term q_i in the whole collection, $tf(q_i, m)$ is the term frequency of q_i in a method m, $|m|$ denotes the length of the method in terms of the total number of tokens it contains, and μ the Dirichlet smoothing parameter.

The SD variant, on the other hand, has 3-node cliques that contain two consecutive query terms and the method in addition to the 2-node cliques that is common to both modeling approaches. We now employ the following potential function for the 3-node cliques that contain a method m and two consecutive query terms q_{i-1} and q_i:

$$\psi_{SD}(q_{i-1}, q_i, m) = \lambda_{SD} log(\frac{tf_W(q_{i-1}q_i, m) + \mu P(q_{i-1}q_i|C)}{|m| + \mu}) \quad (3)$$

where $tf_W(q_{i-1}q_i, m)$ is the number of times that the terms q_{i-1} and q_i appear in the same *order* as in the query within a window length of $W \geq 2$ in the method. Note that the model constant λ_{FI} has no impact on the rankings with the FI assumption. However, we use this parameter in SD model together with λ_{SD} to combine the scores obtained with the 2-node cliques and the 3-node cliques by enforcing $\lambda_{FI} + \lambda_{SD} = 1$ [12].



(a) Full Independence (b) Sequential Dependence

Figure 1. *Markov Networks for Dependence Assumptions for a method and a query with three terms.*

B. Program Structure

Sometimes simple is better. Our next positional proximity approach simply looks for co-occurring query words within the same statement or comment (leading method comments are processed as one contiguous block). We approximate a statement as a line of code ending with a semicolon or curly brace ([;{}]), ignoring line breaks. If a statement contains at least two query words, then the number of occurrences of each query word is incremented. At the end of each method, these frequencies are used to calculate a tf-idf score.

Based on our experience working with tf-idf for the purposes of feature and concern location, we do not use the default configuration of local, global, and length normalization that was most successful in natural language document retrieval in Salton's original comparison [15]. Instead, we use a variation of tf-idf. Given a method m and query Q:

$$tfidf(m, Q) = \sum_{i=1}^{|Q|} (1 + ln(tf(q_i, m))) * ln\left(\frac{N}{df(q_i, C)}\right) \quad (4)$$

where $tf(q_i, m)$ is the term frequency of query word q_i in method m, $df(q_i, C)$ is the number of methods in the program containing q_i, and N is the number of methods in the program. Before calculating word frequencies, we apply camel case split rules and a stemmer specialized for software [16].

We also include a second variant on this approach that only includes tf-idf values if more than one query word appears in a method. The purpose of these approaches is to confirm whether *any* positional proximity information, no matter how crude, improves over baseline search techniques that do not account for positional proximity.

C. Natural Language Semantics

In prior work [4], a positional proximity search technique was proposed, based on the natural language (NL) semantics of phrases occurring in source code structure. Also known as *phrasal concepts*, which are concepts expressed as groups of words such as noun phrases (e.g., 'cell attributes for DB') or verb phrases (e.g., 'get current task from list'), phrasal concepts have been used to improve search accuracy [4] by

more highly weighting occurrences of query words within the same phrasal concept. In this approach, we automatically extract phrasal concepts for a given method signature or call using the Software Word Usage Model (SWUM) [17]. SWUM automatically identifies NL phrase structures such as actions and themes from arbitrary method calls and signatures. When extracting information from method calls, information about both the actual and formal parameters are used (whereas signatures only have formal parameter information). For example, given the signature addToList(Item i), SWUM would identify the action as "add" and the theme as "item".

III. Case Study

In this study, we investigate the impact of positional proximity on search effectiveness by comparing 8 approaches: MRF-SD (the SD variant) and its baseline DLM (described in Section II-A); a statement-based proximity approach (STMT), and method-based proximity approach (MTHD), and their baseline TF-IDF (see Section II-B), two NL phrase-based techniques (NL-Sig and NL-Body), as well as a hybrid approach from prior work (SWUM) [4]. NL-Sig uses phrasal concepts extracted from method signatures only, while NL-Body uses phrasal concepts extracted from method calls in the entire method body. SWUM is a hybrid approach that combines NL-Sig with TF-IDF applied to the method body.

A. Subject Features and Queries

To compare the approaches, we need a ground-truth (i.e., gold) set of queries and features for which to search. In this study, we use two sets of subject features and human-formulated queries: a set of action-oriented features from four Java programs, and a larger set from the documentation of a single Java program.

1) Action-oriented feature set: The first set comprises 8 of 9 features and queries from a previous concern location study of 18 developers searching for action-oriented concerns [9]. One of the techniques in the study, Google Eclipse Search (GES) [18], uses keyword-style queries suitable for input to the search techniques used in this study. For one feature no subject was able to formulate a query returning any relevant results, leaving us with 8 features in our study. For each feature, 6 developers interacted with GES to formulate a query, resulting in a total of 48 queries, 29 of which are unique. The features are mapped at the method level, and contain between 5–12 methods each. The programs contain 23–75 KLOC, with 1500–4500 methods [9]. Note that the average number of words per query is only 1.792, and range in length from 1–5 words.

2) Documentation-based concern set: The second set consists of 215 documented features from the 45 KLOC JavaScript/ECMAScript interpreter and compiler, Rhino, from a set of 415 [19]. Each feature maps to a subsection of the documentation, which is used as the feature description. The number of program elements (methods and fields) in each of the 415 features varies from 1 to 334. For the purposes of this study, we only consider features containing at least 10 program elements and no more than 110 elements, leaving us with 215 concerns.

To obtain human-formulated queries for the set, we asked 8 volunteer software developers familiar with Java programming to read the documentation for a subset of 80-81 concerns. The developers had varying levels of programming and industry experience. The subjects were asked to formulate a query containing words they thought were relevant to the feature and would be the first query they would type into a search engine such as Google when searching. They could include specific identifiers as keywords if those were listed in the documentation. The developers were randomly assigned blocks of features such that 3 different subjects formulated queries for each feature, yielding a total of 645 concern-query combinations. It should be noted that these queries are slightly longer, with an average length of 4.009 words, and ranging in length from 1–9 words.

B. Methodology

Search effectiveness is typically calculated in terms of *precision*, which measures the proportion of relevant documents retrieved out of all of the documents returned, and *recall*, which measures the proportion of relevant documents retrieved out of all the relevant documents for a given query. Precision and recall are typically calculated at a particular rank. In traditional IR, rank-based precision/recall measurements are converted into the calculation of Average Precision (AP) [20], [21], which is the area under the precision-recall curve that results from the individual precision and recall values at different ranks. When AP is averaged over the set of all queries, we get what is known as the Mean Average Precision (MAP), which signifies the proportion of the relevant documents that will be retrieved on average for a given query. A MAP of 0.2 means that, on average, a retrieval for a query returns one relevant document for every 5 documents retrieved. The higher the MAP, the more effective the retrieval algorithm.

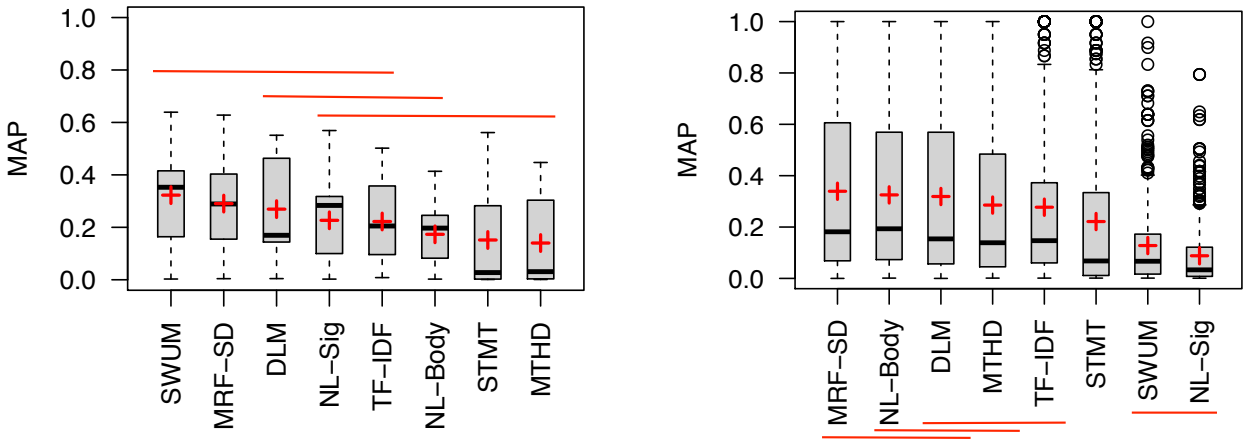
While MAP values capture the effectiveness of a search technique across all relevant ranks, in practice, developers searching software want to see relevant results right at the top of the list. We use the precision at rank 1 (P1) to capture search effectiveness at the top of the list. It should be noted that due to multiple documents having the same score, there may be multiple methods at the top rank. We apply Tukey's Honest Significant Differences (HSD) to determine significant differences between the techniques [22].

C. Threats to Validity

Because our focus is on Java software, these results may not generalize to searching other programs or written in other programming languages. Slight tweaks to the configurations used in the study may result in slight differences in the results. However, these results provide a relative baseline from which to guide further investigations.

D. Results

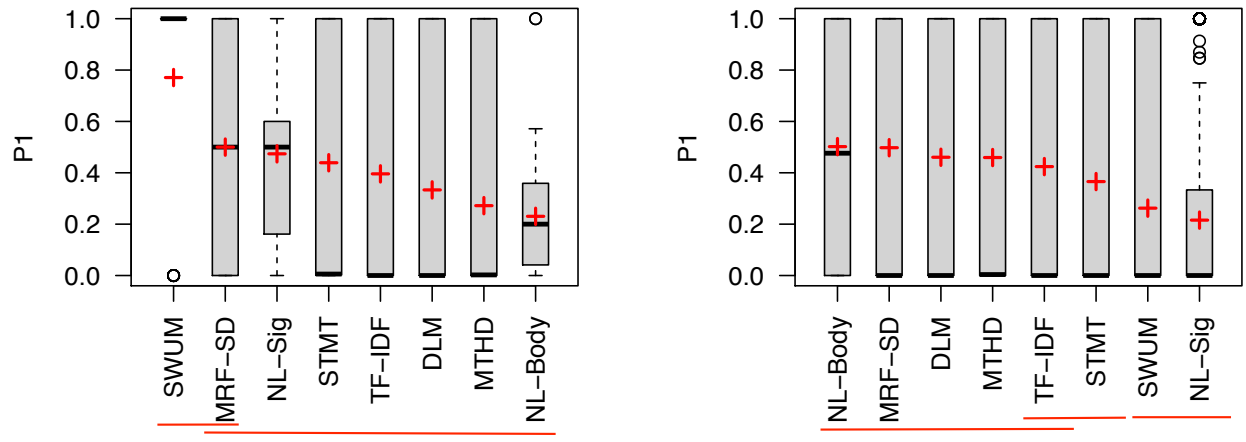
Figures 2 and 3 show box plots of the MAP and precision at rank 1 results for the 8 techniques in the study, sorted



(a) AOC Feature Set

(b) Rhino Feature Set

Figure 2. MAP values, sorted by decreasing mean. Techniques labelled with the same line are not significantly different at $\alpha = 0.05$.



(a) AOC Feature Set

(b) Rhino Feature Set

Figure 3. Precision at rank 1 values, sorted by decreasing mean. Techniques labelled with the same line are not significantly different at $\alpha = 0.05$.

by decreasing mean. The box represents the inner 50% of the data, the heavy middle line represents the median, the plus represents the mean, and '○' indicates outliers. In each figure, the lines under the technique labels are used to indicate statistical significance: techniques connected by a line are not significantly different at $\alpha = 0.05$.

As can be seen by the relative ordering of the techniques, the best techniques vary widely between data sets. on the action-oriented feature set AOC, SWUM performs the best, followed closely by MRF-SD and DLM. Recall that SWUM is a hybrid approach that combines NL-Sig with TF-IDF. Although the means for DLM and MRF-SD are not significantly different, MRF-SD (i.e., positional proximity) offers a clear advantage over DLM for 75% of the data set. We can see that the simplistic statement and method level proximity approaches perform worse than all the techniques, including TF-IDF.

The documentation-based Rhino features offer a different perspective. In Figure 2(b) we see that NL-Body, MRF-SD, DLM, MTHD, and to some extent TF-IDF all perform similarly well. The means and medians are quite close, with the major differences showing in the distribution of effectiveness from top 25% to the upper 50% of the data. For instance, TF-IDF has fewer queries with MAP values above 0.6 and more in the 0.2–0.4 range than the other techniques.

The precision values at rank 1 tell a similar story in Figures 3(a) and 3(b). Of particular note is the success of the hybrid technique, SWUM, on the AOC feature set. On this set, over 75% of the queries have 100% precision at the top rank, implying that the top ranked documents are overwhelmingly relevant (but as the MAP values show, the remaining relevant documents may not be so highly ranked).

Across both diverse data sets, MRF-SD and DLM per-

form consistently well. When analyzed separately, positional proximity plays a role in the most successful techniques, but different types of positional proximity are most effective. For example, in the AOC features, semantic information in NL-Sig combined with TF-IDF in the SWUM approach significantly outperforms the NL-Body, STMT, and MTHD approaches. In contrast, for the Rhino set, SWUM and NL-Sig perform significantly worse than the other approaches. Although the presence of multiple query words is important for relevance, it is not yet clear whether positional proximity is the best way to capture that information for the problem of feature location.

IV. Related Work

Traditional feature location approaches apply bag of word (BOW) techniques using a variety of information retrieval mechanisms, such as LSI [5], LDA [23], ICA [24], FCA [8] or by applying automated query reformulations [2]. A full survey has been recently published [1].

In contrast, FindConcept [9] and SWUM [4] take advantage of natural language (NL) positional proximity by utilizing the semantics of words within phrasal concepts. Recent approaches have applied a multi-faceted approach to feature location [10], and taken advantage of structural dependencies [6].

V. Conclusion

In this paper, we explored using multiple types of positional proximity applied to the problem of feature location. We compared the formal Markov Random Fields (MRF) framework, along with its well-known variant, the Dirichlet Language Model (DLM), with natural language based semantic positional proximity and other ad-hoc approaches based on tf-idf. In a study of over 200 features across 5 open source Java systems, our results indicate that MRF, and its simple variant DLM, are more consistently effective across multiple data sets.

The presence of multiple query words is important for relevance, but it is not yet clear whether positional proximity is the best way to capture that information for the problem of feature location. Prior work has shown that MRF's positional proximity significantly outperforms DLM's simplified model for the problem of bug localization applied at the file level [12]. From this initial study, we see that the problem of feature location at the method level does not show the same consistent improvement from positional proximity as bug localization. This is likely due to the differences in the problem domain. Bug localization queries are typically much longer than 2-4 keywords, often derived from the title or description of the bug report. Plus, working at the file versus method level of granularity significantly changes the number of words in a document. Given the wide disparity in effective techniques for each data set, more research is needed into how the best IR-based feature location approaches can be selected in advance for a particular problem domain.

References

[1] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: A taxonomy and survey," *J. Software Maintenance & Evolution: Research and Practice*, vol. 25, no. 1, pp. 53–95, Jan 2013.

[2] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies, "Automatic query reformulations for text retrieval in software engineering," in *Proc. 2013 International Conference on Software Engineering*, 2013, pp. 842–851.

[3] E. Hill, L. Pollock, and K. Vijay-Shanker, "Automatically capturing source code context of NL-queries for software maintenance and reuse," in *Proc. 31st International Conference on Software Engineering*, 2009, pp. 232–242.

[4] ——, "Improving source code search with natural language phrasal representations of method signatures," in *Proc. 26th IEEE International Conference on Automated Software Engineering, short paper*, 2011, pp. 524–527.

[5] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An information retrieval approach to concept location in source code," in *Proc. 11th Working Conference on Reverse Engineering*, 2004, pp. 214–223.

[6] M. Petrenko and V. Rajlich, "Concept location using program dependencies and information retrieval (depir)," *Inf. Softw. Technol.*, vol. 55, no. 4, pp. 651–659, 2013.

[7] X. Peng, Z. Xing, X. Tan, Y. Yu, and W. Zhao, "Improving feature location using structural similarity and iterative graph mapping," *J. Syst. Softw.*, vol. 86, no. 3, pp. 664–676, 2013.

[8] D. Poshyvanyk, M. Gethers, and A. Marcus, "Concept location using formal concept analysis and information retrieval," *ACM Transactions on Software Engineering and Methodology*, vol. 21, no. 4, 2012.

[9] D. Shepherd, Z. P. Fry, E. Hill, L. Pollock, and K. Vijay-Shanker, "Using natural language program analysis to locate and understand action-oriented concerns," in *Proc. 6th International Conference on Aspect-Oriented Software Development*, 2007, pp. 212–224.

[10] J. Wang, X. Peng, Z. Xing, and W. Zhao, "Improving feature location practice with multi-faceted interactive exploration," in *Proc. 2013 International Conference on Software Engineering*, 2013, pp. 762–771.

[11] B. Sisman and A. C. Kak, "Assisting code search with automatic query reformulation for bug localization," in *Proc. 10th Working Conference on Mining Software Repositories*, 2013, pp. 309–318.

[12] B. Sisman and A. Kak, "Exploiting spatial code proximity and order for improved source code retrieval for bug localization," Purdue University, Tech. Rep. TR-ECE-13-12, Oct 2013.

[13] D. Metzler and W. Croft, "A markov random field model for term dependencies," in *Proc. 28th annual international ACM SIGIR conference on Research and development in information retrieval*, 2005, pp. 472–479.

[14] B. Sisman and A. Kak, "Incorporating version histories in information retrieval based bug localization," in *2012 9th IEEE Working Conference on Mining Software Repositories*, 2012, pp. 50–59.

[15] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information Processing and Management*, vol. 24, no. 5, pp. 513–523, 1988.

[16] A. Wiese, V. Ho, and E. Hill, "A comparison of stemmers on source code identifiers for software search," in *Proc. 2011 16th IEEE International Conference on Software Maintenance and Reengineering*, 2011, pp. 496–499.

[17] E. Hill, "Integrating natural language and program structure information to improve software search and exploration," Ph.D. Dissertation, University of Delaware, Aug. 2010.

[18] D. Poshyvanyk, M. Petrenko, A. Marcus, X. Xie, and D. Liu, "Source code exploration with Google," in *Proc. 22nd IEEE International Conference on Software Maintenance*, 2006, pp. 334–338.

[19] M. Eaddy, T. Zimmermann, K. D. Sherwood, V. Garg, G. C. Murphy, N. Nagappan, and A. V. Aho, "Do crosscutting concerns cause defects?" *IEEE Trans. Soft. Eng.*, vol. 34, no. 4, pp. 497–515, 2008.

[20] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. NY, NY, USA: Cambridge University Press, 2008.

[21] M. D. Smucker, J. Allan, and B. Carterette, "A comparison of statistical significance tests for information retrieval evaluation," in *Proc. sixteenth ACM conference on Conference on information and knowledge management*, 2007, pp. 623–632.

[22] B. A. Carterette, "Multiple testing in statistical analysis of systems-based information retrieval experiments," *ACM Trans. Inf. Syst.*, vol. 30, no. 1, pp. 4:1–4:34, 2012.

[23] S. Lukins, N. Kraft, and L. Etzkorn, "Source code retrieval for bug localization using latent dirichlet allocation," in *Proc. 15th Working Conference on Reverse Engineering*, 2008, pp. 155–164.

[24] S. Grant, J. R. Cordy, and D. Skillicorn, "Automated concept location using independent component analysis," in *Proc. 2008 15th Working Conference on Reverse Engineering*, 2008, pp. 138–142.