Automatic Search Term Identification for Change Tasks

Katja Kevic, Thomas Fritz
Department of Informatics
University of Zurich, Switzerland
{kevic, fritz}@ifi.uzh.ch

ABSTRACT

At the beginning of a change task, software developers search the source code to locate the places relevant to the task. As previous research and a small exploratory study that we conducted show, developers perform poorly in identifying good search terms and therefore waste a lot of time querying and exploring irrelevant code. To support developers in this step, we present an approach to automatically identify good search terms. Based on existing work and an analysis of change tasks, we derived heuristics, determined their relevancy and used the results to develop our approach. For a preliminary evaluation, we conducted a study with ten developers working on open source change tasks. Our approach was able to identify good search terms for all tasks and outperformed the searches of the participants, illustrating the potential of our approach. In addition, since the used heuristics are solely based on textual features of change tasks, our approach is easy and generally applicable and can leverage much of the existing work on feature location.

Categories and Subject Descriptors

D.2.7 [**Distribution, Maintenance, and Enhancement**]

General Terms

Human Factors, Experimentation

Keywords

Initial search, search term, heuristic, change task

1. INTRODUCTION

Software developers generally start a change task by performing searches and exploring code to locate the places that have to be changed for the task [12]. Coming up with good search terms for these initial searches is difficult for developers, sometimes pure guessing and likely to fail [16, 10, 23]. In a small exploratory study we conducted, we found that developers perform poorly in identifying good search terms for unfamiliar change tasks and that only every eighth search term yielded a relevant search result.

ICSE'14, May 31 - June 07 2014, Hyderabad, India

While there is a multitude of research approaches that support developers in locating features within a code base (see [4] for an overview), most of these approaches require the developer to provide an initial search query and then help to reformulate it based on the search results. Since all of these textual feature location approaches are highly dependent on the initial user input [17, 7], the identification of a good search term is as, or even more important than the retrieval algorithm [8]. Only few research approaches support developers in the formulation of a query that generates relevant results, and these either enrich initial queries by adding new terms and leveraging source code information or rely on continuous user assessment of search results (e.g., [23, 15, 6]). All of these approaches still require an initial search query by the user.

To support developers in identifying good search terms in this initial phase, we developed a heuristics-based approach for the automatic identification of search terms in a change task. The heuristics are solely based on textual features of change tasks. This makes the approach easy and generally applicable to any project with a change task repository and does not require any further information, such as the source code of the project. Using logistic regression over a set of open source change tasks, we analyzed the predictive value of the heuristics and determined their weight for the model in our automatic search term identification approach. In a preliminary evaluation on six open source change tasks, we found that our approach was able to identify search terms that generate relevant search results in each one of the tasks and that it outperformed the search terms that developers used when investigating these six change tasks.

This paper makes the following contributions:

- It determines a set of heuristics for the identification of good search terms and analyzes their predictive value based on a logistic regression.
- It provides a general and easy applicable approach that automatically identifies good search terms within a change task.
- It provides a preliminary evaluation of the approach, showing that it is effective at recommending relevant search terms for change tasks and could support and improve developers' performance.

This approach represents a valuable step in supporting developers working on change tasks that has often been ignored. The presented automatic identification of relevant search terms might not only improve developers' performance in locating relevant parts in the code, it is also complimentary and can leverage existing work on feature location that requires a developer to input an initial search term.

2. EXPLORATORY STUDY

To investigate the initial search phase when working on a change task, we conducted a small exploratory study with ten graduate and undergraduate students[1]. For this study, we chose six recently resolved change tasks of the Sando open source project (5.9kLOC, [1]) for which we also had change sets. Each change set contained on averages changes to 3 ($\pm$1.3) methods. We asked each participant to work on a set of three change tasks for at most ten minutes each and identify places in the code that are relevant for completing the change task. The participants had on average 7.4 years ($\pm$3.1) of programming experience and were all unfamiliar with the source code of the Sando project. During the study, we observed participants, captured their screen and took notes. We gathered 30 individual instances of change task investigations. In each of these instances, developers performed several individual queries resulting in a total of 90 separate searches with an average of 3 searches per task and participant. To determine whether a query resulted in a relevant result, we examined whether any of the search results was contained in the change set of the task, a commonly used method in the evaluation of feature location approaches [4].

Overall, participants performed very poorly in coming up with good search terms. Out of the 90 queries performed by the ten participants, only eleven queries (12.2%) returned a relevant result, despite the fact that each of the six change task contained terms that produce relevant search results. Similarly, if one only examines the first search a participant performed per task, out of the 30 queries only 4 (13.3%) returned a relevant result. These results show that it is difficult for developers to identify good search terms when working on an unfamiliar change task. An approach supporting developers in this initial search phase bares great potential since it might allow developers find relevant results faster, requiring less queries overall and also less navigation steps to explore irrelevant results and start making the changes for the task at hand.

3. RELATED WORK

Research addressing the improvement of search terms for feature location engines can broadly be categorized into approaches that improve the query efficiency before the first search is accomplished (e.g., [7]) and approaches that improve the search efficiency depending on the search results generated after the first query is accomplished (e.g. [17, 6]). Approaches which reformulate the query before it is executed by the feature location engine, enrich or reduce the initial user input. For instance, Roldan-Vega et al. [14] and Yang and Tan [23] enrich the original query with alternative search terms based on the frequency of co-occurring terms within the source code. Other approaches (e.g., [21]) leverage verb-direct-object pairs from method signatures and comments to expand the initial user query or employ ontology fragments to generate alternative search terms [16]. *Refoqus* [7] selects the best reformulation strategy (i.e. query expansion or reduction) depending on several properties of the query before the query is executed by feature location engine. All of these approaches depend on an initial user input and leverage the source code of the project to determine the query efficiency, while our approach operates solely on change task information. Furthermore, our approach is complimentary in that it could be used to identify a search term without requiring user input for these approaches.

[1]This study was part of a bigger study that compared a novel search approach with a current state of the art tool [11].

4. APPROACH

Given a change task description, the goal of our approach is to automatically identify terms in the description that provide relevant search results to the developer for starting. For the automatic search term identification, our approach uses a set of heuristics that focuses on features in change task descriptions. By solely focusing on textual features, our approach is light-weight and can be used complimentary with existing approaches that exploit and require explicit links to source code.

For our approach, we derived heuristics from a manual analysis of existing change tasks as well as existing work in the field. We then examined the predictive value and weight of each heuristic applying logistic regression.

Since a good search term produces relevant results when used in a query, we assess the value of a search term by the relevance of the results. In the following, we define a search result as relevant if the result is within the top ten search results. We only consider the top ten since previous research [22], and our observations in the exploratory study have shown that developers tend to ignore results further down in the list. Based on the commonly used method to evaluate feature location approaches [4], we consider a search result as relevant if it is in the change set.

4.1 Heuristics for Search Term Identification

In our approach, we take advantage of heuristics that have previously been identified as useful in determining the importance of a term in a document, such as the tf-idf (term frequency-inverse document frequency) measure commonly used in information retrieval.Since change task descriptions often contain features specific to the software development domain, such as terms in camelCase notation that refer to specific elements in the code, we also derived heuristics by examining existing change tasks. Therefore, we randomly selected 20 resolved change tasks that had change sets associated from the `org.eclipse.mylyn.context` open source project. Both authors of this paper examined the change task descriptions with and without marking the terms in the description that produced relevant search results. To determine which terms produce a relevant search result, we used the textual feature location capabilities of $FLAT^3$ [18] and queried the source code with all terms in each of the change tasks. Using an open coding like approach, each author individually derived a set of heuristics that was discussed, merged and finally discussed with a professional software developer for feedback and further suggestions.

Based on our manual analysis and related work, we determined a set of eleven heuristics that can be categorized into frequency, part of speech, location and notation of terms (see Table 1). While tf-idf measures the importance of a term (e.g., [9, 7]), part of speech heuristics differentiate whether a term is a noun, verb or adjective and has previously been shown to provide value (e.g., [13, 3]). Location heuristics try to capture whether a term occurs in parts of a change task that might be more relevant than others, such as the summary or the beginning of the body (e.g., [9]). Finally, the notation heuristics try to capture whether a term is in a code like notation, such as camelCase or in quotes, and might thus directly refer to code (e.g., [19]).

Table 1: Heuristics for search term identification.

Category	Heuristics
Frequency	tf-idf — based on frequency of a term in a document and all documents in the corpus
Part of Speech	term is a noun, verb or adjective
Location	term occurs in summary
	term occurs in summary & also in body
	term occurs in beginning, middle or end of summary or body
Notation	term is written in camelCase
	term is enclosed in quotes

4.2 Determining Weights for Heuristics

To evaluate the predictive power of each of the eleven heuristics, we performed a logistic regression. We chose another 20 change tasks from the same open source project that again had change sets associated with them, but that were different to the previously analyzed ones. For each change task, we performed text preprocessing steps such as removing stop words and punctuation as well as stemming. Out of the 20 change task descriptions, 17 contained terms that produced relevant search results when queried for, which we then used for the logistic regression. In total, these 17 change tasks contained 492 search terms out of which 395 (80.3%) yield only irrelevant and 97 (19.7%) yield relevant search results. Due to the exploratory nature of this study, we performed a stepwise logistic regression. Furthermore, to counteract suppressor effects the likelihood ratio method was chosen, since it includes less Type II errors than the Wald statistic method [5].

Performing the stepwise logistic regression to optimize the overall fit of the model shows that the predictive value of the model significantly increases when the heuristic for a term being in the summary and at the same time in the body(*isInSumAndBody*), the heuristic for a term being in camelCase notation (*isCamelCase*), the tf-idf measure of a term (*tfidf*) and the heuristic for a term being in the middle of the body or summary (*isInMiddle*), are added to the model (chi square = 52.559, p<.000 with df=4). Table 2 shows the result of the fourth step of the logistic regression ordered by their predictive power from top to bottom. While *isInSumAndBody*, *isCamelCase* and *tfidf* all have a positive effect on the relevance of a term and *isInSumAndBody* has the highest predictive power, the case in which a term is in the middle of the summary or body of a change task (*isInMiddle*) has a negative effect on its relevance. Adding any of the other heuristics did not improve the predictive power of the model significantly. This results in the following model:

$$Relevance(term\ t) = \quad 1/(1+e^{-f(t))}), \text{ with}$$

$$f(t) = \quad -2.100+3.332 * tfidf(t)+1.217 * inSumAndBody(t)- 0.568\ *\ isInMiddle(t) + 0.907\ *\ isCamelCase(t)$$

While the initial model without any heuristics classifies all terms as irrelevant (precision and recall of 0) due to the unilateral dataset, the model based on the four determined heuristics achieves a precision of 0.51 and a recall of 0.23.

4.3 Prototypical Implementation

Based on the above approach, we implemented a fully functional prototype as a Visual Studio extension that supports the identified heuristics and integrates with the Team Foundation Server (TFS). The integration with TFS is necessary to retrieve the corpus of all change tasks of a project and calculate the tf-idf measure of terms. When the user opens a change task with the prototype, it automatically preprocesses the description, calculates the relevance of each term based on the heuristics and ranks the terms. Then, the prototype selects the top three ranked terms and automatically highlights these terms in the change task description. By clicking on one of the highlighted terms in the change task description, a query with the search term will automatically be run in the code search tool Sando [20]. Furthermore, hovering over one of the highlighted terms pops up the estimated relevancy of the term [2].

Table 2: Coefficients in Logistic Regression.

	Weight	Std. Err.	p-value
$inSumAndBody$	1.217	0.292	<0.001
$isCamelCase$	0.907	0.351	0.010
$tf-idf$	3.332	1.581	0.035
$isInMiddle$	-0.568	0.270	0.036

5. PRELIMINARY EVALUATION

To evaluate whether our approach can be applied to different projects and support developers in identifying relevant search terms, we conducted an empirical analysis on the six change tasks of the Sando project from our exploratory study (see Section 2). Specifically, we wanted to investigate the following two questions:

RQ1 Is our approach able to identify search terms in existing change tasks that produce relevant results?

RQ2 Are the automatically identified search terms better than (a) the ones used by developers working on the change task and (b) using the whole change task description as query input?

As the search tool for this analysis, we used Sando [20], a code search tool for Visual Studio that already incorporates several state-of-the-art techniques. We define a search result as relevant if the result is within the top ten results of the search and at most one call reference away from the change set (i.e. the changed elements) for the change task. Previous work (e.g. [12]) and observations in our exploratory study have shown that developers often explore the code of good results and structurally closely related elements. Therefore, in this evaluation and comparison with developers' performance, we assume that a result is relevant if the developer reaches the location for the change either right away, i.e., the search result is in the change set, or the code of the search result contains the location and it is thus at most one structural dependency away from the change set.

To answer $RQ1$, we examine if at least one of the top three search term recommendations produces a relevant result. We chose the top three since participants in our exploratory study used an average of 3 queries ($\pm$1.0) per change task. Applying our approach to each of the six change tasks resulted in at least one search term within the top three recommendations that generated a relevant search result. The relevant search results were on average at the 4.8^{th} position ($\pm$3.0) and in four cases directly in the change set and one step out in the other two cases (see Table 3).

To answer $RQ2$ we collected three sets of search inputs for each of the six change tasks: (a) the first search terms participants used in the exploratory study when working on a change task (in total five first search terms per task,

Table 3: Results of automatic search term identification (ASTI) of top 3 recommendations with position and adjacency as well as comparison with the ones from study participants for all search instances.

Task	ASTI		Search Instances	
	Pos.	Adj.	ASTI > User	User > ASTI
T1	6	1	5	0
T2	1	0	4	0
T3	9	1	0	2
T4	3	0	5	0
T5	7	0	0	0
T6	3	0	3	0

since each task was investigated by five participants); (b) the whole change task after removing stopwords and punctuation (similar as described in [6]); (c) the top most recommendation by our approach. For each change task and each input in the three sets we then queried the latest revision of the Sando code base that did not yet include the changes made for the change task. When comparing the search results produced from the participants' search terms with the ones from the top most recommendation by our approach, our approach produced a more relevant search result in 17 cases, equally relevant results in 11 cases and worse results in only 2 cases. Table 3 presents the results for each task. For change task T3 and T5, for which our approach did not improve upon the search terms of participants, the stemming algorithm applied in our approach impacts the ranking of search terms leading to valuable terms being ranked undesirably low, since, for instance, the terms "upload" and "unupload" are considered as different terms. When comparing the search results produced from the whole, preprocessed change task description with the ones from the top most recommendation, our approach produced a more relevant result in 4 of 6 cases and equally relevant results in 2 cases.

Overall, the results of this preliminary evaluation illustrate that the presented approach automatically identifies search terms that produce relevant search results and that these search terms are in most cases better or at least as good as the ones developers come up with. So while this evaluation is preliminary, it already demonstrates the benefits and the great potential of our pretty light-weight approach for search term recommendation.

6. CONCLUSION AND FUTURE WORK

In this paper, we provided some evidence on the poor performance of ten graduate and undergraduate students when trying to identify search terms. Based on these findings, we developed a light-weight approach that can support developers by automatically identifying good search terms. A preliminary evaluation showed that the search terms identified by our approach generate relevant search results and are better than the ones that developers came up with in the study. Also, since the approach only requires change task information, it can easily be deployed for any project with a task repository. In addition, the approach can be used to leverage a variety of existing approaches, such as feature location approaches that require user input, approaches that reformulate queries or possibly change task summarization and the indexing of tasks. Currently, the approach focuses on identifying individual search terms within a given change task. In future work, we intend to integrate synonyms that are not necessarily part of the task description but might provide even better results, and we plan to integrate support for suggestions that are composed of multiple terms.

7. REFERENCES

[1] `sando.codeplex.com/`.

[2] `www.ifi.uzh.ch/seal/people/kevic/tools/SearchTermIdentification.zip`.

[3] G. Capobianco, A. D. Lucia, R. Oliveto, A. Panichella, and S. Panichella. Improving ir-based traceability recovery via noun-based indexing of software artifacts. *Journal of Software: Evolution and Process*, pages 743–762, 2013.

[4] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk. Feature location in source code: a taxonomy and survey. *Journal of Software: Evol. and Process*, pages 53–95, 2013.

[5] A. Field. *Discovering Statistics Using SPSS*. SAGE Publications, 2005.

[6] G. Gay, S. Haiduc, A. Marcus, and T. Menzies. On the use of relevance feedback in ir-based concept location. In *Proc. of ICSM*, pages 351–360, 2009.

[7] 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. of ICSE*, pages 842–851, 2013.

[8] S. Henninger. Using iterative refinement to find reusable software. *IEEE Softw.*, 11(5):48–59, 1994.

[9] E. Hill, L. Pollock, and K. Vijay-Shanker. Improving source code search with natural language phrasal representations of method signatures. In *Proc. of ASE'11*, pages 524–527.

[10] E. Hill, L. Pollock, and K. Vijay-Shanker. Automatically capturing source code context of nl-queries for software maintenance and reuse. In *Proc. of ICSE*, pages 232–242, 2009.

[11] K. Kevic. Identifying a starting context of code elements for a change task. Master's thesis, University of Zurich, 2013.

[12] A. Ko, B. Myers, M. Coblenz, and H. Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Trans. Softw. Eng.*, 32(12):971–987, 2006.

[13] C. Lioma and I. Ounis. Examining the content load of part of speech blocks for information retrieval. In *Proc. of COLING/ACL*, pages 531–538, 2006.

[14] E. H. Manuel Roldan-Vega, Greg Mallet and J. Fails. Conquer: A tool for nl-based query refinement and contextualizing source code search results. In *Proc. of ICSM*, Sep 2013.

[15] A. Marcus, A. Sergeyev, V. Rajlich, and J. Maletic. An information retrieval approach to concept location in source code. In *Proc. of WCRE*, pages 214–223, 2004.

[16] M. Petrenko, V. Rajlich, and R. Vanciu. Partial domain comprehension in software evolution and maintenance. In *Proc. of ICPC*, pages 13–22, 2008.

[17] D. Poshyvanyk, A. Marcus, and Y. Dong. Jiriss - an eclipse plug-in for source code exploration. In *Proc. of ICPC*, pages 252–255, 2006.

[18] T. Savage, M. Revelle, and D. Poshyvanyk. Flat3: feature location and textual tracing tool. In *Proc. of ICSE*, volume 2, pages 255–258, 2010.

[19] N. Sawadsky, G. C. Murphy, and R. Jiresal. Reverb: Recommending code-related web pages. In *Proc. of ICSE*, pages 812–821, 2013.

[20] D. Shepherd, K. Damevski, B. Ropski, and T. Fritz. Sando: an extensible local code search framework. In *Proc. of FSE*, pages 15:1–15:2, 2012.

[21] 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. of AOSD*, pages 212–224, 2007.

[22] J. Starke, C. Luce, and J. Sillito. Searching and skimming: An exploratory study. In *Proc. of ICSM'09*, pages 157–166.

[23] J. Yang and L. Tan. Inferring semantically related words from software context. In *Proc. of MSR'12*, pages 161–170.