Spotting Familiar Code Snippet Structures for Program Comprehension

Venkatesh Vinayakarao
Indraprastha Institute of Information Technology Delhi
New Delhi, India
venkateshv@iiitd.ac.in

ABSTRACT

Developers deal with the persistent problem of understanding non-trivial code snippets. To understand the given implementation, its issues, and available choices, developers will benefit from reading relevant discussions and descriptions over the web. However, there is no easy way to know the relevant natural language terms so as to reach to such descriptions from a code snippet, especially if the documentation is inadequate and if the vocabulary used in the code is not helpful for web search. We propose an approach to solve this problem using a repository of topics and associated structurally variant snippets collected from a discussion forum. In this on-going work, we take Java methods from the code samples of three Java books, match them with the repository, and associate the topics with 76.9% precision and 66.7% recall.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Retrieval models; D.3.3 [**Language Constructs and Features**]: Control structures

General Terms

Code Search, Program Comprehension

Keywords

Variant Repository, Structure Matching

1. INTRODUCTION AND MOTIVATION

What does the code snippet in Listing 1 do? During software maintenance, developers need to comprehend such snippets written by other developers. The vocabulary used in this snippet is not helpful to make an effective web search. A way to reach the relevant discussion on StackOverflow [1] from Listing 1 will help the developer to understand this snippet related to *Chebyshev Type 2 LPF* [2]. We define snippets that are discussed over the web as *familiar snippets*. Spotting familiar snippets in a given project has not received much attention. We study the challenges involved in structurally matching arbitrary code snippets with familiar snippets. From any given Java project, our implementation is able to locate method definitions that contain snippets discussed in StackOverflow, and show the relevant topics.

```
float process( float in ) {
    double out = in * b[0];
    for( int i=0; i<9; ++i ) out += x[i]*b[i+1];
    for( int i=0; i<9; ++i ) out -= y[i]*a[i+1];
    for( int i=9; i>=1; --i ) y[i] = y[i-1];
    y[0] = out;
    for( int i=9; i>=1; --i )x[i] = x[i-1];
    x[0] = in;
    return out;
}
```

Listing 1: Direct Form I code from StackOverflow.

2. BACKGROUND AND RELATED WORK

We use code snippet structure to locate familiar snippets. We use a repository containing structural representation of familiar snippets and their variant implementations.

Spotting Topics. Spotting topics in source code has been researched under the titles of Feature Location [7], Topic Modeling [5], Code Summarization [9], and Annotation [11]. Research on these lines can be broadly classified into two types: a) Vocabulary based, and b) Structure based. Purely vocabulary based topic models such as Latent Dirichlet Allocation [5] expect helpful user defined terms (UDT) in source code. The closest work to ours is on Structural Semantic Indexing [4]. It uses call hierarchy information, and hence cannot work at intra-procedural level.

Building a Variant Repository. Variant implementations for a set of topics can be associated using the approach proposed by Vinayakarao et al. [15]. For each topic, StackOverflow posts are ranked by the augmented term frequency [12] of topic terms. Snippets from these posts are transformed into a textual representation which enables dropping structurally similar snippets belonging to the same topic. Our objective is to spot these structures in any given arbitrary input program. To the best of our knowledge, this is the first work to use a variant repository for topic identification.

ESEC/FSE'15, August 30 – September 4, 2015, Bergamo, Italy
ACM. 978-1-4503-3675-8/15/08
http://dx.doi.org/10.1145/2786805.2807560

Algorithm 1 Overview: Algorithm to match structures.

```
1: procedure StructureMatcher(snippet)
2:     structure ← structure(snippet)
3:     structureLen ← structure.length
4:     loc ← loc(snippet)
5:     if ((structureLen < minLen) or (loc < minLen))
   then exit
6:     for i from maxLen to minLen do
7:         results ← proximitySearch(structure, i)
8:         if (topicCount(results) > 1) then
9:             vocabulary ← udt(snippet) / stops
10:            if (match(vocabulary, input) == 0) then
11:                results.remove(structure)
12:        matchedList.add(results)
       return matchedList
```

3. APPROACH AND UNIQUENESS

For each method in an arbitrary input program, we check if it is related to a known topic. To achieve this, we use a novel approach of using a variant repository. A variant repository contains structurally heterogeneous implementations for known set of topics as discussed in Section 2. While constructing a variant repository is a solved problem [15], matching variants to code snippets is a hard problem [10] and that is the focus of this paper.

In this paper, we deal with two specific challenges: *Interleaved Code*, and *Overlapping Structures*. Firstly, input code snippet typically contains interleaved code which should not be matched. A trivial example is the occurrence of a print statement. A more complicated example is the use of factorial implementation to compute sine value. So, an exact match of structures does not work. A proximity search [8] on textual representation of structure solves this problem. Secondly, same code structures may apply for several topics. We refer to disambiguating them as a problem of overlapping structures. Figure 1 shows the frequency of all 372 distinct structures for Java method definitions in StackOverflow with more than three lines of code. To deal with overlapping structures, we use UDT in the input snippet for disambiguation. Algorithm 1 puts these ideas together. *stops* denotes a list of manually compiled stop words. We extract the structural representation of the input *snippet* into *structure*. We skip processing if either the snippet length or structural elements count (*structureLen*) are less than *minLen* which is three in our case. Increasing *maxLen* increases query time. For StackOverflow dataset, matching beyond seven terms does not improve precision or recall. Let topicCount(*results*) give the number of topics in the query results. If there are multiple topics, we use udt(*snippet*) in the snippet for disambiguation. Snippet heuristics to replace the use of vocabulary in this algorithm, is under experimentation.

Search based solutions have been used for debugging [3]. Our approach for structural comparison of code snippets, opens up opportunities for solving problems that are otherwise difficult to solve using traditional program analysis techniques.

4. RESULTS AND CONTRIBUTIONS

We manually setup a 156 topic variant repository from StackOverflow data. We used 1322 Java methods mentioned

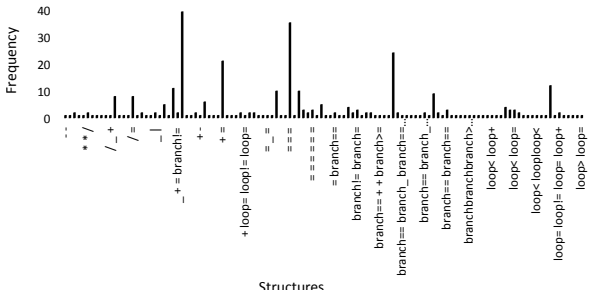

Figure 1: There exist 372 distinct structures in StackOverflow. Here, we show some of them and their frequency.

in three Java books [14, 6, 13] as our input dataset. The vocabulary used in the books were different when compared with those in our repository. Hence, a purely vocabulary based comparison does not work. For each topic, we are interested in knowing if it was associated with input programs precisely (Precision) and if all relevant methods were associated for any specific topic (Recall). We extract identifiers from all snippets stored in our repository and find that their frequency follows power law. We use the elbow of this curve to capture most occurring names and use it as our stopwords list. With this configuration, we surfaced 14 distinct topics out of the 39 topics that were expected to match (recall of 35.9%). We got 16 correct out of 26 matched topics (61.5% precision).

Discussion. We observe that the loss of precision and recall, is caused by three major factors: a) Setup of IR system b) Nature and limitations of data, and c) Effectiveness of topics. We do a post-analysis on the results to understand the potential of our approach and segregate the results that lost precision because of these limitations. Based on these insights, we update the IR system configuration and also re-phrase the topics. With such a well-configured setup, we achieve **76.9% precision** (20 out of 26) and **66.7% recall** (26 out of 39). Deriving best configurations based on data and query logs is an open area of research in IR.

5. CONCLUSION AND FUTURE WORK

We present a new approach to spot familiar snippets by modeling it as a structural search problem over a repository of variants without being solely dependent on program vocabulary. Our work supports and enhances existing vocabulary based techniques and is not proposed as an alternative. On the downside, our approach depends on data availability and heavy computing infrastructure. In this work, we have used program comprehension as a case to show the utility of a variant repository. In future, we hope to address scalability issues, refine the structural comparison technique and use it to solve more problems in software engineering such as bug detection, bug localization, and clone detection.

6. ACKNOWLEDGMENTS

The author would like to thank Rahul Purandare (IIITD), Aditya Nori (MSR) and Matthew Dwyer (UNL) for their guidance. This work is supported by MSR and CII.

7. REFERENCES

[1] http://StackOverflow.com/.

[2] How to implement Chebyshev Type 2 LPF in Java? http://StackOverflow.com/questions/16879642/how-to-implement-chebyshev-type-2-lpf-in-java.

[3] B. Ashok, J. Joy, H. Liang, S. K. Rajamani, G. Srinivasa, and V. Vangala. Debugadvisor: A recommender system for debugging. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, pages 373–382, New York, NY, USA, 2009. ACM.

[4] S. K. Bajracharya, J. Ossher, and C. V. Lopes. Leveraging usage similarity for effective retrieval of examples in code repositories. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '10, pages 157–166, 2010.

[5] L. R. Biggers, C. Bocovich, R. Capshaw, B. P. Eddy, L. H. Etzkorn, and N. A. Kraft. Configuring latent dirichlet allocation based feature location. *Empirical Softw. Engg.*, 19(3):465–500, June 2014.

[6] J. Bloch. Effective Java, Second Edition.

[7] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk. Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process*, 25(1):53–95, 2013.

[8] A. Feuer, S. Savev, and J. A. Aslam. Implementing and evaluating phrasal query suggestions for proximity search. *Inf. Syst.*, 34(8):711–723, Dec. 2009.

[9] S. Haiduc, J. Aponte, and A. Marcus. Supporting program comprehension with source code summarization. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ICSE '10, pages 223–226, 2010.

[10] S. Lahiri, C. Hawblitzel, M. Kawaguchi, and H. Rebelo. Symdiff: A language-agnostic semantic diff tool for imperative programs. In *Computer Aided Verification (CAV '12) (Tool description)*. Springer, July 2012.

[11] A. Lucia, M. Penta, R. Oliveto, A. Panichella, and S. Panichella. Labeling source code with information retrieval methods: An empirical study. *Empirical Softw. Engg.*, 19(5):1383–1420, Oct. 2014.

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

[13] H. Schildt. Java: A Beginner's Guide, Sixth Edition. http://www.mhprofessional.com/product.php?isbn=0071809252.

[14] K. Sierra and B. Bates. Head First Java, Second Edition. http://www.headfirstlabs.com/books/hfjava/.

[15] V. Vinayakarao, R. Purandare, and A. V. Nori. Structurally heterogeneous source code examples from unstructured knowledge sources. In *Proceedings of the 2015 Workshop on Partial Evaluation and Program Manipulation*, PEPM '15, pages 21–26, 2015.