International Journal of Information and Computation Technology.
ISSN 0974-2239 Volume 3, Number 7 (2013), pp. 647-654
© International Research Publications House
http://www. irphouse.com /ijict.htm

Application Program Interface (API) Based Code Search Engine for Predicting Relevant Applications

Rahul Kumar Kurba and V. Koti Reddy

Department of Computer Science and Engineering
JNTUA College of Engineering Anantapuram (Autonomous), India.

Abstract

Software reuse or source code reusability is one of main aspect of software engineering which helps in reuse of software component or code snippets which are already developed and well tested. It helps in reducing the cost and time in development of software which are one of main influencing factors in software development life cycle. But a fundamental of finding relevant software applications that are being developed in software development task is due to a highly intended task associated with the development of software and low level implementations details of application in repositories. In order to reduce the mismatch in between the high level intention associated with software development and low level details of the project. In order to address this problem this paper uses an Application Program Interface (API) based code search engine for predicting relevant application. Basically the search engine works by taking two rankings of application. First consider the description of application and second examine the API being used in application. The performance has been evaluated by combining the both ranks of application with semantic search so that it can retrieve more-relevant applications.

Keywords: Source code search engine, software reuse, information retrieval, concept location.

1. Introduction

Now a days programmers face many challenges when try to find source code or snippets to reuse in current software development task [2].The fundamental problem of discovering relevant code is the mismatch between the high level intention related to the descriptions of software and low level implementation details of software project. The above problem is described as the concept assignment problem [3]. Source code search engines are developed to retrieve relevant source code by matching keywords in queries to words in the descriptions of applications, comments in their source code, and the names of program variables and types. Source code search engines dig into software repositories to find relevant source code which contain thousands of software projects. But many source code repositories are polluted with poorly functioning projects [4], by simply using a match between keywords from the query of user with the description of software project in the repository and it does not guarantee that the retrieved project or source code is relevant to the query of user. Today many source code search engines return only snippets or piece of code that are relevant to user queries. For programmers this create confusion [5] how to reuse these code snippets or piece of code. But the problem of reuse is the code fragments retrieved look very similar [6]. If search engines retrieve code snippets in the circumstances of executable applications; it makes easy for programmers to understand how to reuse these code fragments.

The Present day code search engines like e.g., Google Code Search, Source Forge generally treat code as a normal text where the source code doesn't have semantics Software applications contain API calls which contain function abstraction of software projects and the semantics of API calls are well defined which helps in retrieving the relevant source code. The concept of using API calls was proposed but not implemented elsewhere [7],[8]. But this was evaluated over larges databases of source code by using standard information retrieval methods [9].

In order to retrieve the software application by using API calls this paper addresses a search system called API code search engine for predicting relevant application as a part of Searching, Selecting, and Synthesizing (S3) architecture [10].This source code search engine retrieve highly relevant applications in order to reuse in current software development task. It considers mainly three things in order to locate software applications.

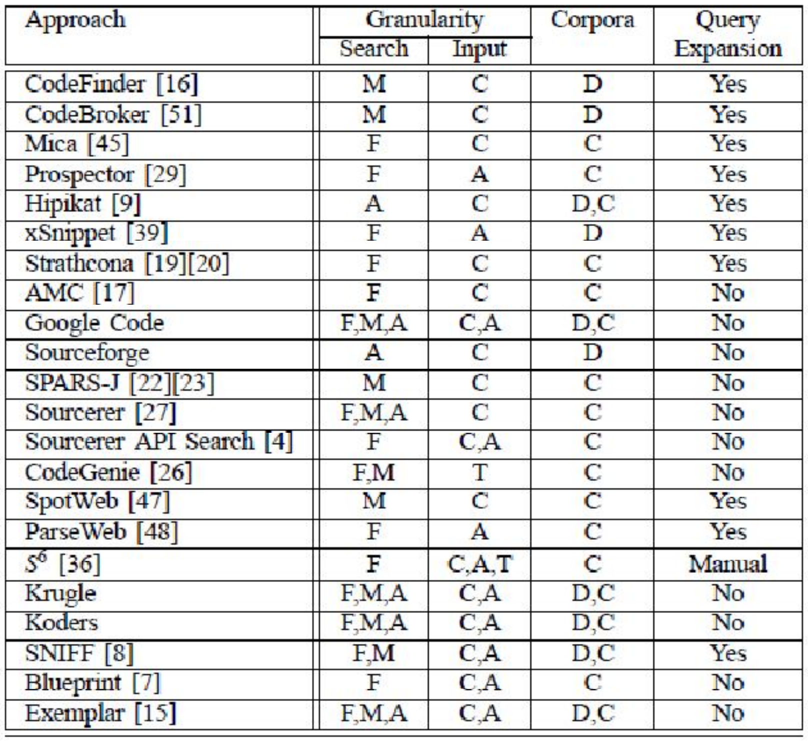
1. Description of application
2. API calls user by software application which contain functional abstraction of software project
3. Dataflow between these API calls.

Class relation and looping conditions are also evaluated and implemented in two different case studies. The key finding here is this source code search engine retrieve more accurate results when API calls are considered instead of only the description of application.

2. Related Work

Many code mining techniques and tools are proposed and developed in order to retrieve more accurate relevant software component belong to software application for reuse form different repositories as it shown in Table 1. Code finder try to locate similar code by using terms in source code. But this search engine uses API documentation for locating source code based on keywords from users. This search won't retrieve source code by comparing keywords in source code itself.

Today many source code search engines uses code broker system for retrieving the source code by using comments done on program by programmer to find relevant artifact's [12]. But in this search engine the code broker system depend upon the documentation, meaningful names of program variables and its types and this leads to retrieve more precise results.



Approach	Granularity		Corpora	Query Expansion
	Search	Input		
CodeFinder [16]	M	C	D	Yes
CodeBroker [51]	M	C	D	Yes
Mica [45]	F	C	C	Yes
Prospector [29]	F	A	C	Yes
Hipikat [9]	A	C	D,C	Yes
xSnippet [39]	F	A	D	Yes
Strathcona [19][20]	F	C	C	Yes
AMC [17]	F	C	C	No
Google Code	F,M,A	C,A	D,C	No
Sourceforge	A	C	D	No
SPARS-J [22][23]	M	C	C	No
Sourcerer [27]	F,M,A	C	C	No
Sourcerer API Search [4]	F	C,A	C	No
CodeGenie [26]	F,M	T	C	No
SpotWeb [47]	M	C	C	Yes
ParseWeb [48]	F	A	C	Yes
S^6 [36]	F	C,A,T	C	Manual
Krugle	F,M,A	C,A	D,C	No
Koders	F,M,A	C,A	D,C	No
SNIFF [8]	F,M	C,A	D,C	Yes
Blueprint [7]	F	C,A	C	No
Exemplar [15]	F,M,A	C,A	D,C	No

Figure 1: Comparison with other code search engines.

MICA is a web search tool which helps for finding relevant applications based on API components documents and help pages related to API calls for searching source code of project [14]. Mica uses documentation about the API calls in order to refine search results.

SNIFF is another idea for searching which uses API calls for retrieving source code [7],[13] and mainly used for searching java using free form quires and it also check interaction between these retrieved code snippets [8] .

Prospector is a tool which combines snippets of code retrieved based on a user query and this code also contain both input and required output types and very useful for program to write more complex code but it can’t retrieve full source code [15].

Hipikat [16] is a tool which helps user to retrieve developed artifacts which contain source code revisions and changes that were made to source code in the past. It is useful for inexperienced software developers to learn from previous experiences of their colleagues.

Strathcona is a heuristic tool which helps developer to learn from experience for problem solving, learning and discovering the solution [17],[18].This helps in guiding the programmer to learn things.

Like strathcona there are other technique which helps in retrieving programs according to the interest of developer [19], [20]. FRAN is one such a technique which helps programmer to locate a function looks similar to given function. And one more tool called XSnippet [21] which is a context sensitive tool helps developer in order to find code snippets of relevant programs [22].

3. Overview of Proposed System

The proposed system implements an important component semantic query evaluation. User provides query in terms of semantic description and the query evaluator does semantic query execution to provide the matching results.

There are two ranking models for ranking the results retrieved for quires of user; first one is Quality Of Match (QOM) which ranks the things by matching between overall goodness between the components [23] and another ranking model Component rank model (CRM), which analyze the actual usage relations of the components and propagating significance through usage relations [24],[25]. Unlike CRM, [26] this search engine uses a combination of API calls and relations between those API calls that helps in searching for more relevant projects.

Software architecture in Figure 2., gives a brief description about working of this source code search engine how it analyses the software application and categorize the applications which helps in retrieving more relevant applications.

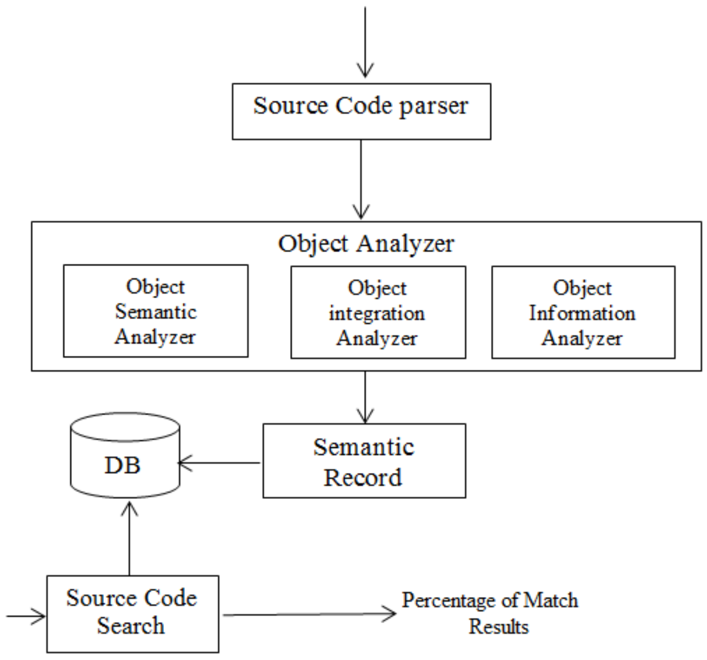


Figure 2: Working of Search engine.

4. Details of Proposed System

In this section, the detail explanation of proposed system is explained step by step

4.1 Semantic description

User expresses the query in terms of semantics of the application to be searched. The semantic relations considered for our work are

1. API calls
2. API data flow
3. Classes
4. Class relations
5. Comment texts
6. Iterations times

4.2 Query Execution

User semantic queries are executed to find the match in the repository of applications. Based on the number of matches the score is given by using a variable scoring strategy. Different semantic relations are given different score.

The score are given from highest to lowest in order of
API call – 25
API data flow - 20

Classes - 15
Class relations - 10
Comment texts - 5
Iterations times -5

The number of times matched is multiplied by its corresponding score and all semantic score are summed up to give the matching score. Applications are sorted in terms of their score from highest to lowest and result is provided.

5. Results

The accuracy of application search, the criteria used for measuring accuracy is how for a search query, how many relevant applications appear in the top 10 list. This is tested for different queries and measured the accuracy of the system and compared the results with Exemplar which is just based on API calls.

The proposed solution has better accuracy than Exemplar.

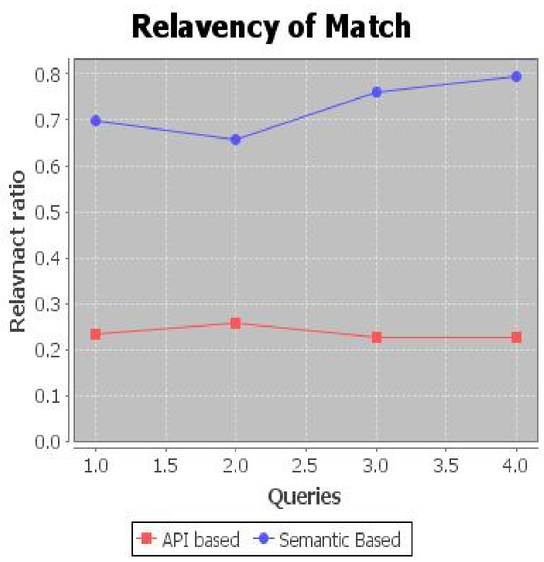


Figure 3: Accuracy of predicting relevant application between API based and semantic based.

6. Conclusion

This paper addresses an API based search engine for predicting highly relevant applications. This search engine retrieve trivial results (more than one result) this helps in getting more relevant application related current software development task. By adding the semantic information to API helps in retrieve more accurate results.

References

[1] Exemplar: A Source Code Search Engine ForFinding Highly Relevant Applications Collin McMillan, *Member, IEEE,* Mark Grechanik, *Member, IEEE,* Denys Poshyvanyk, *Member, IEEE,* Chen Fu, *Member, IEEE,* Qing Xie, *Member, IEEE*

[2] Susan Elliott Sim, Medha Umarji, Sukanya Ratanotayanon, and Cristina V Lopes. How well do internet code search engines supportopen source reuse strategies? *TOSEM*, 2009.

[3] Ted J. Biggerstaff, Bharat G. Mitbander, and Dallas E. Webster. Program understanding and the concept assigment problem. *Commun. ACM*,37(5):72–82, 1994.

[4] James Howison and Kevin Crowston. The perils and pitfalls of mining Sourceforge. In *MSR*, 2004.

[5] Charles W. Krueger. Software reuse. *ACM Comput. Surv.*, 24(2):131–183, 1992.

[6] Mark Gabel and Zhendong Su. A study of the uniqueness of source code. In *Foundations of software engineering*, FSE '10, pages 147–156,New York, NY, USA, 2010. ACM.

[7] Mark Grechanik, Kevin M. Conroy, and Katharina Probst. Finding relevant applications for prototyping. In *MSR*, page 12, 2007.

[8] Shaunak Chatterjee, Sudeep Juvekar, and Koushik Sen. Sniff: A search engine for java using free-form queries. In *FASE*, pages 385–400, 2009.

[9] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze*Introduction to Information Retrieval*. Cambridge University Press, NewYork, NY, USA, 2008.

[10] Denys Poshyvanyk and Mark Grechanik. Creating and evolving software by searching, selecting and synthesizing relevant source code. In *ICSE Companion*, pages 283–286, 2009.

[11] Scott Henninger. Supporting the construction and evolution of component repositories. In *ICSE*, pages 279–288, 1996.

[12] Yunwen Ye and Gerhard Fischer. Supporting reuse by delivering taskrelevant and personalized information. In *ICSE*, pages 513–523, 2002.

[13] Jeffrey Stylos and Brad A. Myers. A web-search tool for finding API components and examples. In *IEEE Symposium on VL and HCC*, pages 195–202, 2006.

[14] Sushil K. Bajracharya, Joel Ossher, and Cristina V. Lopes. Leveraging usage similarity for effective retrieval of examples in code repositories. In *Foundations of software engineering*, FSE '10, pages 157–166, NewYork, NY, USA, 2010. ACM.

[15] David Mandelin, Lin Xu, Rastislav Bod´ik, and Doug Kimelman. Jgloid mining: helping to navigate the API jungle. In *PLDI*, pages 48–2005.

[16] [16] Davor Cubranic, Gail C. Murphy, Janice Singer, and Kellogg S. Booth. Hipikat: A project memory for software development. *IEEE Trans. Software Eng.*, 31(6):446–465, 2005.

[17] Reid Holmes and Gail C. Murphy. Using structural context to recommend source code examples. In *ICSE*, pages 117–125, 2005.

[18] Reid Holmes, Robert J. Walker, and Gail C. Murphy. Strathcona examplrecommendation tool. In *ESEC/FSE*, pages 237–240, 2005.

[19] Martin P. Robillard. Automatic generation of suggestions for program investigation. In *ESEC/FSE*, pages 11–20, 2005.

[20] [20] Martin P. Robillard. Topology analysis of software dependencies. ACM Trans. Softw. Eng. Methodol., 17(4):1–36, 2008.

[21] Zachary M. Saul, Vladimir Filkov, Premkumar Devanbu, and Christian Bird. Recommending random walks. In Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, ESEC-FSE ’07, pages 15–24, New York, NY, USA, 2007. ACM.

[22] Naiyana Sahavechaphan and Kajal T.Claypool. XSnippet: mining fosample code. In *OOPSLA*, pages 413–430, 2006.

[23] Naiyana Tansalarak and Kajal T. Claypool. Finding a needle in the haystack: A technique for ranking matches between components. In *CBSE*, pages 171–186, 2005.

[24] Katsuro Inoue, Reishi Yokomori, Hikaru Fujiwara, Tetsuo Yamamoto, Makoto Matsushita, and Shinji Kusumoto. Component rank: Relative significance rank for software component search. In *ICSE*, pages 14–24, 2003.

[25] Katsuro Inoue, Reishi Yokomori, Tetsuo Yamamoto, Makoto Matsushita, and Shinji Kusumoto. Ranking significance of software components based on use relations. *IEEE Trans. Softw. Eng.*, 31(3):213–225, 2005.

[26] Reishi Yokomori, Harvey Siy, Masami Noro, and Katsuro Inoue. Assessing the impact of framework changes using component ranking. Software Maintenance, IEEE International Conference on, 0:189–198, 2009.