www.ijecs.in
International Journal Of Engineering And Computer Science ISSN: 2319-7242
Volume 5 Issue 12 Dec. 2016, Page No. 19789-19792

An Efficient Usage of APIs in Software Code Reuse

Dr. Madhavi Karanam
Gokaraju Rangaraju Institute of Engineering and Technology,
Department of CSE,
Hyderabad, India
madhaviranjan@griet.ac.in

Abstract: ***Web search tools have been produced to address this issue by coordinating catchphrases in questions to words in the portrayals of utilizations, remarks in their source code, and the names of program factors and sorts.. On the off chance that code sections are recovered with regards to executable applications, it makes it simpler for developers to see how to reuse these code fragments. In request to address this issue an application seek framework called Application Program Interface (API) for anticipating pertinent applications as a portion of Searching, Selecting, and Synthesizing (S3) design. This web index helps clients find very significant executable applications for reuse. It consolidates diverse wellsprings of data about applications keeping in mind the end goal to find important programming: the printed depictions of utilizations, the API calls utilized inside every application, and the dataflow among those API calls such as Class relations, circling requirements, and remarks on code. By including more semantic portrayal of venture this aides in recovering unimportant outcomes i.e., more than at least one number of result identified with current programming improvement assignment identified with client query. This can be valuable to the client keeping in mind the end goal to get various significant programming applications related current advancement undertaking for reusability of venture.***

Keywords: API, code reuse, searching, relevant extractor, fragments**.**

1. Introduction

Programming reuse or source code reusability is one of principle part of programming building which helps in reuse of programming segment or code scraps which are as of now created and very much tried. It helps in decreasing the cost and time being developed of programming which are one of principle impacting elements in programming advancement life cycle But a central of discovering pertinent programming applications that are being created in programming improvement undertaking is because of an exceptionally expected errand connected with the improvement of programming and low level executions points of interest of use in storehouses. To diminish the confusion in the middle of the abnormal state goal connected with programming improvement and low level points of interest of the venture. With a specific end goal to address this issue utilizes an Application Program Interface (API) based code web index for foreseeing pertinent application. Fundamentally the web index works by taking two rankings of use. In the first place consider the depiction of use and second look at the API being utilized as a part of utilization. The execution has been assessed by consolidating the both positions of use with semantic pursuit from so it can recover more-pertinent applications.

An immediate approach for finding profoundly significant applications is to seek through the depictions and source code of utilizations to match catchphrases from inquiries to the names of program factors and sorts. This approach expect that developers pick significant names when making source code, which is regularly not the situation .This issue, is somewhat tended to by software engineers who make important depictions of the applications in programming archives. Nonetheless, cutting edge web search tools utilize correct matches between the catchphrases from inquiries, the words in the portrayals, and the source code of utilizations. Shockingly, it is troublesome for clients to figure correct watchwords in light of the fact that no single word can be depicted a programming idea in the most ideal way. The vocabulary picked by a software engineer is likewise identified with the idea task issue in light of the fact that the terms in abnormal state portrayals of uses may not coordinate terms from the low-level usage (e.g. Identifier names and remarks).

2. Literature Survey

Writing study is essentially done keeping in mind the end goal to dissect the foundation of the present venture which discovers blemishes in the current framework and aides on which unsolved issues that can work out. Along these lines, the accompanying themes show the foundation of the venture as well as reveal the issues and imperfections which reused to propose arrangements and work on this venture. An assortment of research has been done on learning of aggregate conduct. Taking after segment investigates diverse references that talk about around a few points identified with aggregate conduct.

Today Programmers confront many difficulties when attempt to discover source code to reuse in current programming advancement assignment [2].The major issue of finding pertinent code is the bungle between the abnormal state aim identified with the portrayals of programming and low level execution points of interest of programming undertaking. The above issue is depicted as the idea task issue [3]. Source code web indexes are created to recover important source code by coordinating watchwords in questions to words in the depictions of utilizations, remarks in their source code, and the names of program factors and sorts. Source code web indexes dive into programming stores to discover significant source code which contain a great many programming ventures. Yet, many source code archives are dirtied with ineffectively working activities [4], by essentially utilizing a match between catchphrases from the question of client with the depiction of programming venture in the vault and it doesn't ensure that the recovered venture or source code is important to the inquiry of client. Today many source code web crawlers return just scraps or bit of code that are important to client inquiries. For software engineers this make perplexity [5] how to reuse these code bits or bit of code. Be that as it may, the issue of reuse is the code pieces recovered look fundamentally the same as [6]. On the off chance that internet searchers recover code scraps in

the conditions of executable applications; it makes simple for developers to see how to reuse these code pieces.

The Present day code web indexes like e.g., Google Code Search, Source Forge for the most part regard code as an ordinary content where the source code doesn't have semantics Software applications contain API calls which contain work deliberation of programming tasks and the semantics of API calls are all around characterized which helps in recovering the significant source code. The idea of utilizing API calls was proposed however not executed somewhere else [7], [8]. In any case, this was assessed over large databases of source code by utilizing standard data recovery strategies [9].

Keeping in mind the end goal to recover the product application by utilizing API calls this paper addresses a pursuit framework called API code web search tool for anticipating significant application as a piece of Searching, Selecting, and Synthesizing (S3) engineering [10].This source code internet searcher recover very pertinent applications with a specific end goal to reuse in web and flow programming advancement assignment. It considers fundamentally three things keeping in mind the end goal to find programming applications.

Miner is a device that incorporates parts of code in light of client questions that contain input sorts and fancied yield sorts [15]. Miner is a compelling instrument to help software engineers in composing confused code; in any case, it doesn't give support to an undeniable code web crawler. Catchphrase writing computer programs is a strategy which interprets a few user-gave watchwords into a substantial source code explanation. Watchword programming matches the catchphrases to API calls and the parameters of those calls. At that point, it connects those parameters to factors or different capacities additionally said in the watchwords. Model is like watchword programming in that Exemplar matches client inquiries to API calls, and can suggest utilization of those calls. Dissimilar to catchphrase programming, Exemplar indicate cases of past use of those APIs, and does not endeavor to incorporate those calls into the client's own source code.

The Hipikat apparatus [16] suggests pertinent improvement artifacts(i.e., source updates connected with a past change errand) from a venture's history to a designer. Dissimilar to Exemplar, Hipikat is a programming errand arranged device that does not suggest applications whose functionalities coordinate abnormal state prerequisites.

Strathcona is an instrument that heuristically matches the structure of the code a work in progress to the case code. Strathcona is valuable while helping software engineers while working with existing code [17],[18], in any case, its utility is not pertinent when hunting down important tasks given a question containing abnormal state ideas with no source code.

FRAN is a system which helps software engineers to find capacities like given capacities [19], [20]. At long last, XSnippet [21] is a setting delicate instrument that permits engineers to inquiry a specimen archive for code pieces that are important to the programming job needing to be done.

Model is like these calculations in that it utilizes relations between API calls as a part of the recovered tasks to process the level of enthusiasm (positioning) of the extend. Dissimilar to these methodologies, Exemplar requires only a normal dialect question depicting a programming assignment. They found in [22] considering the dataflow among API calls does not enhance the pertinence of results for our situation.

3. Proposed Implementations

Theoretical background highlighting some topics related to this paper. The description contains several topics which are worth to discuss and also highlight some of their limitation that encourage going on finding solution as well as highlights some of their advantages for which reason these topics and their features are used in this paper.

3.1 Predicting Relevant Applications

Suppose that a programmer needs to encrypt and compress data. A programmer will naturally turn to a search engine such as SourceForge and enter keywords such as encrypt and compress. The programmer then looks at the source code of the programs returned by these search engines to check to see if some API calls are used to encrypt and compress data. The presence of these API calls is a good starting point for deciding whether to check these applications further.

Code search engine include help documentations of widely used libraries, such as the standard Java Development Kit (JDK). Existing engines allow users to search for specific API calls, but knowing in advance what calls to search for is hard. Our idea is to match keywords from queries to words in help documentation for API calls. These help documents are descriptions of the functionality of API calls as well as the usage of those calls.

In Exemplar, they extract the help documents that come in the form of Java Docs. Programmer's trust these documents because the documents come from known and respected vendors, were written by different people, reviewed multiple times, and has been used by other programmers who report their experience at different forums.

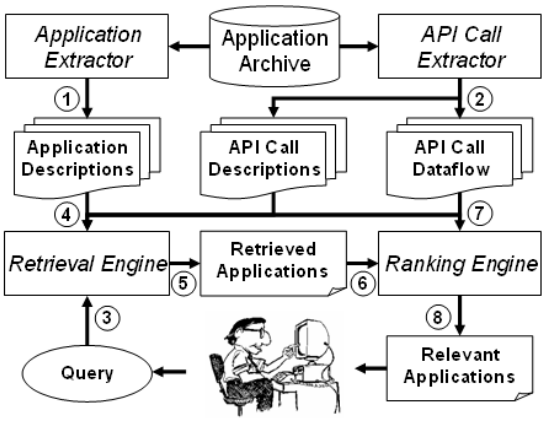

Figure 1: Brief description about working of a search engine.

In this search engine relations between concepts entered in queries are often reflected as dataflow links between API calls that implement these concepts in the program code. This observation is closely related to the concept of the software reflection models formulated by Murphy, Notkin, and Sullivan. In these models, relations between elements of high-level models (e.g., processing elements of software architectures) are

preserved in their implementations in source code. For example, if the user enters keywords secure and send the corresponding API calls encrypt and email are connected via some dataflow, then an application with these connected API calls are more relevant to the query than applications where these calls are not connected.

Consider two API calls string encrypt() and void email(string). After the call encrypt is invoked, it returns a string that is stored in some variable. At some later point a call to the function email is made and the variable is passed as the input parameter. In this case these functions are connected using a dataflow link which reflects the implicit logical connection between keywords in queries. Specifically, the data should be encrypted and then sent to some destination.

3.1.1 Example

API search engine returns applications that implement the tasks described in by the keywords in user queries. Consider the following task: find an application for sharing, viewing, and exploring large data sets that are encoded using MIME, and the data can be stored using key value pairs. Using the following keywords MIME, type, data, an unlikely candidate application called BIOLAP is retrieved using Exemplar with a high ranking score. The description of this application matches only the keyword data, and yet this application made it to the top ten of the list.

BIOLAP uses the class MimeType, specifically its method getParameterMap, because it deals with MIME-encoded data. The descriptions of this class and this method contain the desired keywords, and these implementation details are highly-relevant to the given task. BIOLAP does not show on the top 300 list of retrieved applications when the search is performed with the SourceForge search engine.

3.2 Ranking Schema

Ranking mechanisms for retrieving source code are centered on locating components of source code that match other components. Quality of match (QOM) ranking measures the overall goodness of match between two given components, which is different from this search engine which retrieves applications based on high-level concepts that users specify in queries. Component rank model (CRM) is based on analyzing actual usage relations of the components and propagating the significance through the usage relations.

The RAS component is responsible for ranking applications based on the API calls made in those applications. This component first locates a number of descriptions of API calls which match the keywords provided in the user's query. It then matches those API calls to applications which use those calls.

3.3 Executing the User Query

This section describes the way of finding the results retrieved by search engine. User provides query in terms of semantic description and the query evaluator does semantic query execution to provide the matching results. This helps in retrieving more number of results related to the search of user query. The results retrieved are categorized based on semantic description provided to the retrieved results.The user provides his query in the form of semantic description and executes the query for predicting the relevant application.

(a) Semantic Description

(b) Query Execution

(a) Semantic description

User expresses the query in terms of semantics of the application to be searched. This contains the description application user want to search and the domain, related names of classes or a name of variables.

Semantic information is expressed in the form of Resource Description Framework(RDF). RDF is defined as the Resource Description Framework (RDF) is a family of World Wide Web Consortium (W3C) specifications originally designed as a metadata data model.

It has come to be used as a general method for conceptual description or modeling of information that is implemented in web resources, using a variety of syntax notations and data serialization formats. This information is directly given to the search engine for finding relevant applications.

(b) 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 and 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.

Performance Graph:-

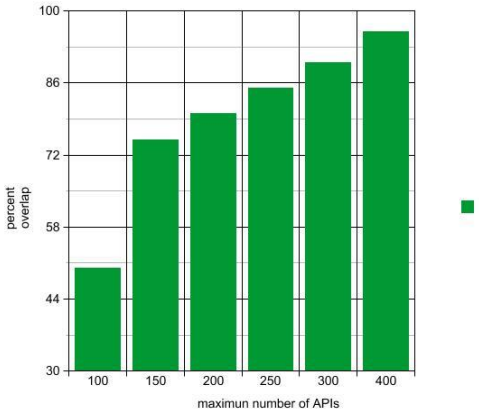

Fig 2: Comparison between APIs and overlap.

This graph comparison between the performances of Exemplar code search with API based code search engine.

4. Conclusion

The Exemplar system suggests that for finding relevant application it consider three things and rank the relevant applications in following way. Description of applications, API calls used by applications and analyse the dataflow in API calls. And this will not provide more results for the search of the user query.

By considering more semantic information of the software application it helps in retrieving more number of results for a query of user. It helps user in order to get more relevant software applications and helps in increasing the scope of reusability. Class relation and looping conditions are also evaluated. This search engine helps in retrieving trivial result for the query of a user.

The future work scope of this project is displaying the API calls description on the result page of search engine. This helps user in order to identify more required results regarding his/her query and it helps user to select more results related to his/her project.

And another scope of future work is a way of sorting and filtering the API calls user by application because some time the code search engine retrieve results with same type check method many times this retrieve some irrelevant result of user query.

5. 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, MedhaUmarji, SukanyaRatanotayanon, 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] JamesHowison 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] ShaunakChatterjee, SudeepJuvekar, and Koushik Sen. Sniff: A search engine for java using free-form queries. In *FASE*, pages 385–400, 2009.

[9] Christopher D. Manning, PrabhakarRaghavan, and HinrichSchtzeIntroduction 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 ICSECompanion, pages 283–286, 2009.

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

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

[13] Jeffrey Stylos and Brad A. Myers. A web-search tool for finding API component 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.

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

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

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

[22] NaiyanaSahavechaphan and KajalT.Claypool.XSnippet: mining for sample code. In *OOPSLA*, pages 413–430, 2006.

[23] Steven P. Reiss. Semantics-based code search. In *ICSE*, pages 243–253, 2009.

Author Profile

Dr. K. Madhavi, working as a Professor in Computer Scince and Engineering Department, Gokaraju Rangaraju Instittute of Engineering and Technology. She has completed her B.E in 1997, M.Tech from JNTUA in 2003 and awarded Ph.D from JNTUA in 2013. She has 19 years of teaching experience. She has published several papers in reputed international journals and international conference. Her research interest include sofware engineering , Model Driven Engineering, Data Mining, and other areas.