IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 3, March 2016

A Novel Approach to Semantic Search for Accurate Source Code

Divya Kumari Tankala[1], Vikas Boddu[2]

Asst. Professor, CSE Department, Vignan Institute of Technology and Science, Hyderabad[1]

Asst. Professor, CSE Department, GITAM University, Andhra Pradesh[2]

Abstract: Reusability is one of the assets of software. The software engineering helps for reusability. It helps in reducing the cost and time in development of software which are one of main influencing factors in software development life cycle. It increases the productivity and results by the programmer. Even it good still scale and approximate matches then complex specification is an open issue. We propose a novel approach to semantic search that addresses some of these limitations and is designed for queries that can be described using an example. In this approach, programmers write lightweight specifications as inputs and expected output examples for the behavior of desired code. Using these specifications, an SMT solver identifies source code from a repository that matches the specifications. This research contributes the first work toward using SMT solvers to search for existing source code. In this, we motivate the study of code search and the utility of a more semantic approach to code search. We introduce and illustrate the generality of our approach using subsets of three languages, Java, Yahoo! Pipes, and SQL. Our approach is implemented in a tool, Satsy, for Yahoo! Pipes and Java. Finally, we show that this approach is adaptable to finding approximate matches when exact matches do not exist, and that programmers are capable of composing input/output queries with reasonable speed and accuracy. These results are promising and lead to several open research questions that we are only beginning to explore.

Keywords: Novel Approach, Semantic Search, SMT Solver, SNIFF, Java, Yahoo! Pipes, and SQL.

1. INTORUDCITON

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.

2. LITERATURE SURVEY

Manber presented approximate index concept to measure similarity between strings in different documents (Manber 1994 [6]). A tool called "Sif" is developed to find similar files in a large file system. He proposed the concept of approximate index to measure the similarity of character strings between documents, which was adopted later by many similar systems.

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.

DOI 10.17148/IJARCCE.2016.53176

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 3, March 2016

Some papers tried to tackle the performance problem of finding plagiarism in documents through using indexing (Mozgovoy *et al.*, 2005 [7]). Such concept is utilized also in search engines for fast document retrieval.Several papers tried to compare between different source code analyses tools (*e.g.*, Jun-Peng et al 2003 [10], Maurer *et al.*, 2006 [11], Kustanto and Liem 2009 [12], Hage *et al.*, 2010 [13], *etc.*).

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] .

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.

3. RELATED WORK

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. In order to reduce the mismatch in between the high level intention associated with software development, Programmers frequently search for source code to reuse using keyword searches.

When effective and efficient, a code search can boost programmer productivity; however, the search effectiveness depends on the programmer's ability to specify a query that captures how the desired code may have been implemented. Further, the results often include many irrelevant matches that must be filtered manually. More semantic search approaches could address these limitations, yet existing approaches either do not scale, are not flexible enough to find approximate matches, or require complex specifications. We propose a novel approach to semantic search that addresses some of these limitations and is designed for queries that can be described using an example. In this approach, programmers write lightweight specifications as inputs and expected output examples for the behavior of desired code.

Using these specifications, an SMT solver identifies source code from a repository that matches the specifications. The repository is composed of program snippets encoded as constraints that approximate the semantics of the code. This research contributes the first work toward using SMT solvers to search for existing source code. In this, we motivate the study of code search and the utility of a more semantic approach to code search. We introduce and illustrate the generality of our approach using subsets of three languages, Java, Yahoo! Pipes, and SQL. Our approach is implemented in a tool, Satsy, for Yahoo! Pipes and Java. The evaluation covers various aspects of the approach, and the results indicate that this approach is effective at finding relevant code. Even with a small repository, our search is competitive with state-of-the-practice syntactic searches when searching for Java code. Further, this approach is flexible and can be used on its own, or in conjunction with a syntactic search.

SEMANTIC SEARCH

Information retrieval by searching information on the web is not a fresh idea but has different challenges when it is compared to general information retrieval. Different search engines return different search results due to the variation in indexing and search process. Google, Yahoo, and Bing have been out there which handles the queries after processing the keywords. They only search information given on the web page, recently, some research group's start delivering results from their semantics based search engines, and however most of them are in their initial stages. Till none of the search engines come to close indexing the entire web content, much less the entire Internet. Current web is the biggest global database that lacks the existence of a semantic structure and hence it makes difficult for the machine to understand the information provided by the user.

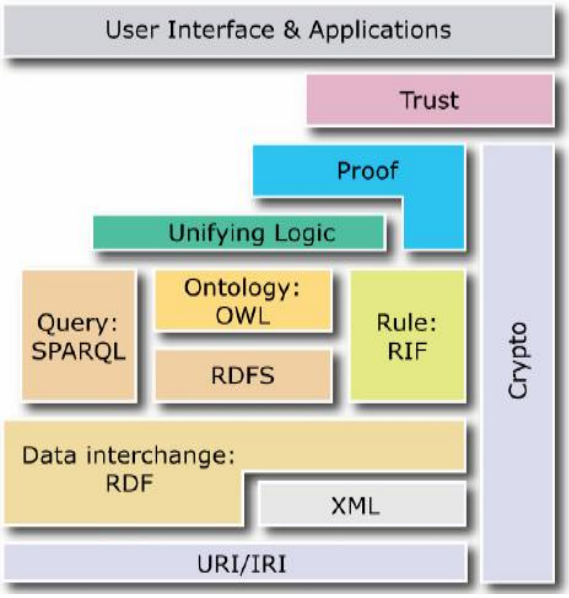

Fig.2. Semantic Search

When the information was distributed in web, we have two kinds of research problems in search engine i.e. How can a search engine map a query to documents where information is available but does not retrieve in intelligent and meaning full information? The query results produced

DOI 10.17148/IJARCCE.2016.53176

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 3, March 2016

by search engines are distributed across different documents that may be connected with hyperlink. How search engine can recognize efficiently such a distributed results?

Semantic web [4] [5], can solve the first problem in web with semantic annotations to produce intelligent and meaningful information by using query interface mechanism and ontology's. Other one can be solved by the graph-based query models [6]. The Semantic web would require solving extraordinarily difficult problems in the areas of knowledge representation, natural language understanding. The following figure depicts the semantic web frame work it also referred as the semantic web layercake by W3C.

Current Web & Limitations

Present World Wide Web is the longest global database that lacks the existence of a semantic structure and hence it becomes difficult for the machine to understand the information provided by the user in the form of search strings. As for results, the search engines return the ambiguous or partially ambiguous result data set; Semantic web is being to be developed to overcome the following problems for current web.

• The web content lacks a proper structure regarding the representation of information.
• Ambiguity of information resulting from poor interconnection of information.
• Automatic information transfer is lacking.
• Usability to deal with enormous number of users and content ensuring trust at all levels.
• Incapability of machines to understand the provided information due to lack of a universal format.

Finally, we show that this approach is adaptable to finding approximate matches when exact matches do not exist, and that programmers are capable of composing input/output queries with reasonable speed and accuracy. These results are promising and lead to several open research questions that we are only beginning to explore.

4. PROPOSED WORK

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.

We propose a novel approach to semantic code search that addresses several of these limitations and is designed for queries that can be described using an input/output example. In this approach, programmers write lightweight specifications as inputs and expected output examples. Unlike existing approaches to semantic search, we use an SMT solver to identify programs or program fragments in a repository, which have been automatically transformed into constraints using symbolic analysis, that match the programmer provided specification.

We instantiated and evaluated this approach in subsets of three languages, the Java String library, Yahoo! Pipes mashup language, and SQL select statements, exploring its generality, utility, and tradeoffs. The results indicate that this approach is effective at finding relevant code, can be used on its own or to alter results from keyword searches to increase search precision, and is adaptable to find approximate matches and then guide modifications to match the user specifications when exact matches do not already exist. These gains in precision and existibility come at the cost of performance, for which underlying factors and mitigation strategies are identified.

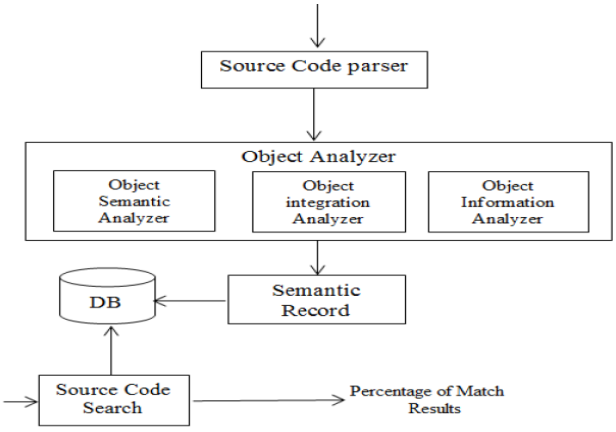

Figure 2: Working of Search engine.

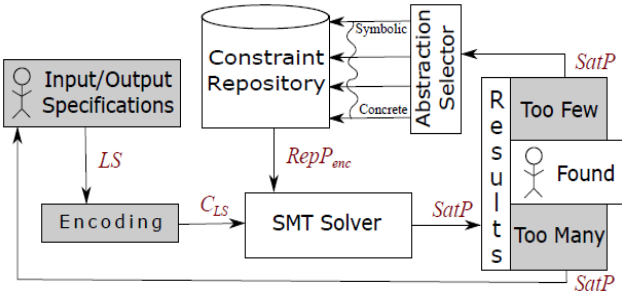

Fig.3.Search Process

5. EXPERIMENTAL WORK

We have motivated, defined, instantiated and evaluated a new approach to source code search that uses input/output examples as specifications and an SMT solver to identify search results. In this section, we discuss the related work. Our approach is related to recent work in code search, code reuse, verification and validation, and program synthesis.

Code Search

The ability to search for source code on the Internet has proven to be essential for many common software development and maintenance tasks. However, available code search engines are typically limited to lexical searches and do not take in consideration the underlying semantics of source code such as the program structure or language. Especially object-oriented source code, which includes inheritance and polymorphism, is often not modeled to a level in which it can be queried sufficiently. In this paper, we present a Semantic Web-based approach

to source code search that uses ontologies to model and connect source code fragments extracted from repositories on the Internet. This modeling approach allows us to reason and search across project boundaries while dealing with incomplete knowledge and ambiguities. In comparison with other search tools, we incrementally build our knowledge base without the need to re-visit fragments or compile the source code. Our search engine is scalable and allows us to process and query a large code base collected from open source projects.

Different definitions exist as what constitutes a large scale source code search. In [7] the term Internet-scale Code Search was introduced and defined as the process of searching over a large number of open source projects and other complementary information resources such as, web pages and wiki pages that include source code. In [10] the term Next-Generation Code Search was presented to describe a source code search that considers structural code information and provides services to support other software engineering activities. In the context of our research we combine and extend the functionalities covered by these existing source code search definitions by three major factors: (1) The ability to search over a large amount of source code such as found on the Internet, (2) fine-grained structural search capabilities and dynamic code analysis and (3) the support for incomplete source code fragments. Code fragments, in this work, are defined as physically separated pieces of code (e.g. a class file or a few lines of code within a web page) which are syntactically correct and have a logical dependency to other code fragments. In this paper, we are interested in improving both the completeness and precision of search results compared to existing source code engines available on the Web. It has to be pointed out, that dealing with source code fragments is inherently more difficult than source code search within a project that can be build (has all source code available e.g., within Eclipse). Dependencies among code fragments (e.g. library dependency represented by import statements in Java) need to be resolved without holding any assumption about the order and the availability of other fragments and without compromising the scalability of the engine. This is one of the main challenges of source code search on the Internet and a motivating start point for our research. In what follows we will discuss the type of source code search scenarios that should be supported.

We have described an approach to code search that is semantic and uses input/output examples to define the queries, which is closely related to research in code search. Recent studies have revealed that programmers frequently use general search engines to find code for reuse [Sim et al. 2011], and our own study contains these findings [Stolee and Elbaum 2012a]. More specialized syntactic code search engines in the state-of-the-practice (e.g., Koders, Krugle) also incorporate filtering capabilities (e.g., language, libraries) and program syntax into the query to guide the matching process, such as type signatures of desired code [Sim et al. 2011]. These approaches search at an internet-scale, whereas our search approach operates over repositories. Other approaches in the state-of-the-art add natural language processing to increase the potential matches [Grechanik et al. 2010][McMillan et al.2011]. Our work is di_erent in that the search is semantic, but as we show (Section 5.2.3), both approaches are complementary and can be combined.

Early work in semantic code search required developers to write complex specifications using first-order logic or specialized languages (e.g., [Ghezzi and Mocci 2010][Penix and Alexander 1999][Zaremski and Wing 1997]), which can be expensive to develop and error-prone. The cost of writing specifications can be reduced by using incomplete behavioral specifications, such as those provided by test cases (a form of input/output) [Lazzarini Lemoset al. 2007][Podgurski and Pierce 1993][Reiss 2009], but these approaches require that the code be executed to _nd matches. Some approaches also require a keyword query to first prune the search space, which could miss some solutions [Reiss 2009]. Further, executing test cases only returns exact matches, missing many relevant matches that may have a slightly different signature (e.g., extra parameter). Other search approaches use sequences of API calls [Mishne et al. 2012] or sequences of textual statements [Chan et al. 2012] as queries to find code that performs the specified actions in a specified order, but implementation details are required for an effective search.

Code Reuse

In the code reuse process, there are two primary activities: finding and integrating. Our approach focuses on finding, which is what we have evaluated, but it has potential to be useful with integration. For effective reuse, scope and dependencies must be understood for developers to effectively integrate code [Garlan et al. 1995]. Some recent work assists programmers with integrating new code by matching it to structural properties in their development environment (e.g.,method signature, return types) [Cottrell et al. 2008][Holmes et al. 2006]. Real-time clone detection can promote reuse by identifying code clones as they are developed, but again this depends on a developer having a sense of how to implement code [Lee et al. 2010]. Further, while these approaches guarantee structural matching, the behavior of the integrated code may not be well understood.

Verification and Validation

In this work, we have talked about how symbolic analysis is used to generate constraints that represent the program behavior, and that this representation is used in the search process. Symbolic execution [Clarke 1976][Clarke and Richardson 1985][King 1976] is a technique that executes code with symbolic, rather than concrete, values, and can generate such symbolic summaries of source code. These are similar to the summaries that our implementation generates to represent code behavior. For two of our languages presented in this work, SQL and Yahoo! Pipes, symbolic execution tools are not readily available. For Java, however, tools like the symbolic execution extension [Khurshid et al. 2003][JPF-symbc 2012] to the Java

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 3, March 2016

Q	Title	Input String	Output String
1	Just copy a substring in java	Animal.dog World.game	Animal World
2	extract string including whitespaces within string (java)	23 14 this is random	this is random
3	How to get a 1.2 formatted string from String?	1.500000154	1.5
4	How to pull out sub-string from string (Android)?	<TD>TextText</TD>	TextText
5	Trim last 4 characters of Object	Breakfast($10)	Breakfast
6	Removing a substring between two characters (java)	I <str>really</str> want ...	I really want ...
7	Splitting up a string in Java	i i i block_of_text	block_of_text
8	How to find substring of a string with whitespaces in Java?	c not in(5,6)	true
9	Limiting the number of characters in a string, and chopping off the rest	124891 difference 22.348 montreal	1248 diff 22.3 mont
10	Trim String in Java while preserve full word	The quick brown fox jumps	The quick brown...
11	How to return everything after x characters in a string	This is a looong string	is a looong string
12	Slice a string in groovy	nnYYYYYYnnnnnnn	YYYYYY
13	How to replace case-insensitive literal substrings in Java	FooBar fooBar	Bar Bar
14	Removing first character of a string	Jamaica	amaica
15	How to find nth occurrence of character in a string?	/folder1/folder2/folder3/	folder3
16	Java finding substring	**tok=zHVVMHy...	zHVVMHy
17	Finding a string within a string	...MN=5,DTM=DIS...	DTM=DISABLED

Table.1. Java Artifacts Specifications

PathFinder model checker [Visser et al. 2003] can generate symbolic summaries that we can use, but are limited in the data types that are supported. At this point, part of our ongoing work is to integrate our encoding process with such tools, taking advantage of their capabilities to generate summaries for certain complex code structures. In validation, constraint and SMT solvers have been used extensively for test case generation. Toward the goal of database generation for testing, reverse query processing takes a query and a result table as inputs and using a constraint solver, produces a database instance that could have produced the result [Binnig et al. 2007]. Other work in test case generation for SQL queries has used SMT solvers to generate tables based on queries [Veanes et al. 2010].

6. CONCLUSION

In this paper, a novel Semantic-Web enabled source code search is introduced. Our approach does not only scale in regards to the large amount of source code available on the Internet but is also optimized to work with web crawlers to incrementally build a semantic knowledge base. Our approach is scalable as we support one-pass parsing and do not make assumptions on the order of fragments analyzed. This makes our Semantic Web based approach unique among other currently adopted approaches for Internet scale source code search research. We have pointed out various usage scenarios of source code search in which traditional search engines fail to provide (complete) results and show how we out-perform them by incorporating source code semantics into the knowledge base. We have discussed our ontology design in regards to its strengths and weaknesses and pointed out common pitfalls in designing source code ontologies in general, and our highly optimized ontology for source code search in particular. Furthermore, we have analyzed various source code meta modeling approaches under the aspect of source code search to make our approach scalable i.e. the potential and tradeoffs of our search approach over the state-of-the-practice and the state-of-the-art, describe how to encode search queries and programs in three languages, the Java string library, Yahoo! Pipes, and SQL select statements, and explore the effectiveness of our approach in each of these domains. Generality and efficiency in the context of richer programs, such as those contains loops and other complex constructs, are concerns that still need to be addressed.

As part of our future work, we plan to add the versioning information extracted from metadata to the repository and deal with contradictory information (e.g. through levels of trust). In addition, more detailed source code query capabilities will be added to our Eclipse plug-in. We also plan to make our tool available to the community in form of an open source code search engine and are currently working on the web interface for this project. This is just one step toward our ultimate goal of leveraging existing resources, such as source code repositories, to positively impact programmer productivity.

REFERENCES

1. Erdos, K., Sneed, H.M.: Partial comprehension of complex programs (enough to perform maintenance). In: 6th International Workshop on Program Comprehension (1998)
2. Welty, C.: Augmenting abstract syntax trees for program understanding. In: 12th IEEE International Conference Automated Software Engineering (1997)
3. Bajracharya, S., Ngo, T., Linstead, E., Dou, Y., Rigor, P., Baldi, P., Lopes, C.: Sourcerer: a search engine for open source code supporting structure-based search. In: 21h ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (2006)
4. Hoffmann, R., Fogarty, J.: Assieme: finding and leveraging implicit references in a web search interface for programmers. In: 20th annual ACM Symposium on User Interface Software and Technology (2007)
5. Holmes, R., Murphy, G.C.: Using structural context to recommend source code examples. In: 5th International Conference on Software Engineering (2005)
6. Grove, D., DeFouw, G., Dean, J., Chambers, C.: Call graph construction in object-oriented languages. In: 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (1997)
7. Gallardo-Valencia, R.E., Sim S.E.: Internet-scale code search. In: 1st ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation (2009)
8. Auer, S., Lehmann, J.: What have Innsbruck and Leipzig in common? Extracting Semantics from Wiki Content. In: European Semantic Web Conference (2007)
9. Rilling, J., Witte, R., Schuegerl, P., Charland, P.: Beyond Information Silos - an Omnipresent Approach To Software Evolution. J. Semantic Computing. 2, 431--468 (2008)
10. Bajracharya, S., Ossher, J.: Sourcerer: An internet-scale software repository. In: 1st ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation (2009).
11. 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*
12. Susan Elliott Sim, Medha Umarji, Sukanya Ratanotayanon, and Cristina V Lopes. How well do internet code search engines supportopen source reuse strategies? *TOSEM*, 2009.
13. Ted J. Biggerstaff, Bharat G. Mitbander, and Dallas E. Webster. Program understanding and the concept assigment problem. *Commun. ACM*,37(5):72–82, 1994.
14. James Howison and Kevin Crowston. The perils and pitfalls of mining Sourceforge. In *MSR*, 2004.
15. Charles W. Krueger. Software reuse. *ACM Comput. Surv.*, 24(2):131–183, 1992.
16. 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.
17. Mark Grechanik, Kevin M. Conroy, and Katharina Probst. Finding relevant applications for prototyping. In *MSR*, page 12, 2007.
18. Shaunak Chatterjee, Sudeep Juvekar, and Koushik Sen. Sniff: A search engine for java using free-form queries. In *FASE*, pages 385–400, 2009.

BIOGRAPHIES

Ms. Divya Kumari Tankala received Master Degree in Computer Science and Engineering form Jawaharlal Nehru Technological University, Hyderabad (JNTUH). Her research interest includes Information and Communication Technologies and Software engineering. Presently she is working as an Asst.Prof in CSE Department, Vignan Institute of Technology and Science, Hyderabad

Mr. Vikas Boddu received Master Degree in Bio-Informatics from Jawaharlal Nehru Technological University, Hyderabad (JNTUH). Hir research interest includes cloud computing and computer networks. Presently she is working as an Asst.Prof in CSE Department, GITAM University, Andhra Pradesh.

DOI 10.17148/IJARCCE.2016.53176