Extraction of Method Signatures from Ontology Towards Reusability for the Given System Requirement Specification

S. Sagayaraj and Gopinath Ganapathy

Abstract **- Software reuse improves productivity, quality, and maintainability of software products. Only few completed projects are achieved and documented. The method signatures in a completed project are stored in the Ontology and the source code components are stored in Hadoop Distributed File System (HDFS). Methods are needed for the new project can be extracted from the Ontology using Software Requirement Specification (SRS) document. UML design document will evolve after many phases from SRS and hence this work proposes a new framework to extract keywords from SRS and estimate the number of new methods to be developed and count the number of methods that can be reused from the Ontology. The SRS document for the project consists of purpose, scope, system requirements, functional requirements and non-functional requirements as metadata. The SRS document is given as input and the keywords are extracted. The keywords are searched in Ontology for the similar method prototypes and the appropriate code components would be extracted from the HDFS. These methods are integrated in the new project with a review process. The implementation is provided with the sample SRS text. The keywords are extracted and matched with the Ontology. The reusability is measured using reuse metrics, quality, and knowledge growth.**

Index Terms **- Metadata, Knowledge Management, Ontology, Reusability, WordNet.**

I. INTRODUCTION

ONTOLOGIES are built to represent generic knowledge about a target world [1]. Ontologies increase the efficiency and consistency of describing resources, by enabling more sophisticated functionalities in development of knowledge management and information retrieval applications. Software companies make use of the already developed code to build up a knowledge management system the software companies make use of prebuilt code base. In order to develop new software projects with reusable codes. Systematic reuse of previously written and tested code is a way to increase software development productivity as well as the quality of the software [2, 3, 4]. Software Reuse has been cited as the most effective means for improving the productivity in software development projects [5, 6]. Some general reusability guidelines, include ease of understanding, functional completeness, reliability, error and exception handling, information hiding, high cohesion and low coupling, portability and modularity [7]. For each new project, software teams design new components and code by employing new developers. If the firm archives the completed code and components, they can be used with no further testing. To reuse the code, a tool can be created to extract the metadata such as function, definition, type, arguments, brief description, author, and so on from the source code and store them in Ontology. For a new project, the developer can search for components in the Ontology and retrieve them at ease. The Ontology represents the knowledgebase of the company for the reuse code. The Ontology can be used to search [8], retrieve, maintain and view informations. The projects are stored in Ontology and the source code is stored in the HDFS [9]. The UML class diagram is a design document considered as the input. The method metadata is extracted from the UML and passed to the SPARQL to extract the available methods from the Ontology. By selecting appropriate method from the list the code component is retrieved from the HDFS [10]. But for this paper SRS is used as input. After extracting the keywords from the SRS document these keywords are matched with the Ontology. From the retrieved methods, the developer can account for how many are already available in the repository and how many to be developed. By uploading projects in Ontology and HDFS the corporate knowledge grows and the developers can reuse code than developing newly.

The paper begins with a note on the related technology and precedent work is in section 2. The detailed features and framework for Source Code Retriever is found in section 3. The Keyword Extractor for SRS Text file is in section 4. The Method Retriever by Jena framework and Source Retriever from the HDFS are in section 5. The implementation Scenario is in section 6. The software measures of metrics, quality and knowledgebase growth is explained in section 7. Section 8 deals with the findings and future work of the paper.

Date of paper submission is March 6th 2011. Revised on April 18th 2011.

S. Sagayaraj is with Department of Computer Science, Sacred Heart College, Tirupattur, India. (+91 9443035624; fax: +91 4179 225060; e-mail: sagi_sara@yahoo.com).

Gopinath Ganapathy is with the Department of Computer Science, Bharathidasan University, Trichy, India. (e-mail: gganapathy@gmail.com).

II. RELATED WORK

A. *Hadoop AND HDFS*

Hadoop is a framework for the development of highly scalable distributed computing applications [11]. It supports the processing of large data sets in a distributed computing environment. Hadoop is designed to efficiently process large volumes of information [12]. It is a simplified programming model, which allows the user to write and test distributed systems quickly. The monitoring system re-replicates the data in response to system failures, which can result in partial storage. Even though the file parts are replicated and distributed across several machines, they form a single namespace, so their contents are universally accessible. Map

Reduce [13] is a functional abstraction, which provides an easy-to-understand model for designing scalable, distributed algorithms.

B. Ontology

The key component of the Semantic Web is the collections of information called 'ontologies'. Gruber defines ontology as a specification of a conceptualization. Ontology defines the basic terms and their relationships comprising the vocabulary of an application domain and the axioms for constraining the relationships among terms. This definition explains what an ontology looks like [14].The most typical kind of ontology for the Web has taxonomy and a set of inference rules. The taxonomy defines classes of objects and relations among them.

C. Ontology Construction

After the completion of a project, all the project files are sent to source code extraction framework that extracts metadata from the source code. Only java projects are used for this framework. The java source file or folder that consists of java files is passed as input along with the project information like description and version. The framework extracts the metadata from the source code using QDox code generators and stores it in the Ontology using Jena framework. The source code is stored in the HDFS [15].

D. Source Code Retriever for UML

Source Code Retriever is a framework that takes UML class diagram or XMI (XML Metadata Interchange) file as an input. The Source Code Retriever consists of three components: Keyword Extractor for UML, Method Retriever and Source Retriever. The Keyword Extractor for UML extracts the metadata from the UML class diagram. Method Retriever component retrieves the matched methods from the repository. Method Retriever constructs SPARQL query to retrieve the matched results. The user should select the appropriate method from the list of methods and retrieve the source code by Source Retriever component, which interacts with HDFS and displays the source code.

III. SOURCE CODE RETRIEVER

The Source Code Retriever [10] assumes that the Ontology is constructed for the project and the source code of the project is stored in the HDFS. Source Code Extractor form Ontology is a framework that takes SRS document as an input from the user and suggests the reusable methods for the given extracted keywords. The Source Code Retriever process flow is shown in Figure 1. The Source Code Retriever consists of three components: Keyword Extractor for SRS, Method Retriever and Source Retriever. The Keyword Extractor for SRS extracts the keywords from the SRS document. The SRS document is stored as a word file. The Keyword Extractor for SRS retrieves keywords from the Word file. The keywords extracted by the Keyword Extractor for SRS are passed to the Method Retriever component. Method Retriever component retrieves the methods matched from the repository. Method Retriever construct SPARQL query to retrieve the matched results. The user should select the appropriate method from the list of methods and retrieve the source code by Source Retriever component, which interacts with HDFS and displays the source code.

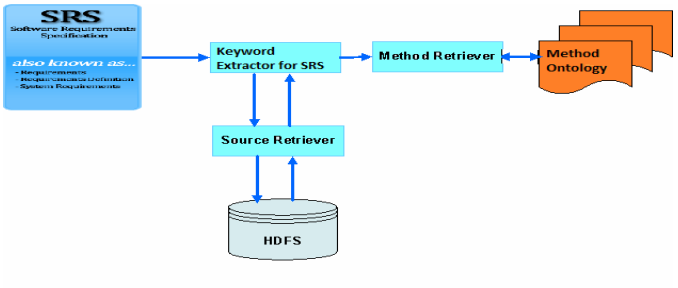

Figure 1. Process of Source Code Extractor from Ontology

IV. KEYWORD EXTRACTOR FROM SRS

The SRS document can have different types of formats out of which the following two types are used for extraction:

- Use Cases model
- Description model

A. Use Case Model

SRS document contains the keywords as Use Cases, which relate to the method names of the project. There is no built-in API in sun JDK to read or write word document. The contents of the SRS document is converted to simple text format using Apache Poor Obfuscation Implementation (POI) API's and it is given to java API's for text extraction. This POI API is capable of manipulating different types of Microsoft office suite. A major use of the Apache POI API is for Text Extraction applications. The Apache POI project is the master project for developing pure Java ports of file formats based on Microsoft's Object Linking and Embedding (OLE) 2 Compound Document Format. OLE 2 Compound Document Format is used by Microsoft Office Documents, as well as by programs using Microsoft Foundation Class (MFC) property sets to serialize their document objects. The Text extraction identifies the use cases and extracts the name of the use case as the keywords. The Apache POI project contains many subcomponents out of which Horrible Word Processor Format (HWPF) aims to read and write Microsoft Word 97 format files. HWPF is a port of Microsoft Word file format for Java. It supports read and limited write capabilities. The SRS source document is given as input using the absolute path or a file name or a workspace related URL. The process checks for the respective path and return the URL. A constructed input stream is passed to the POIFS is to read the word file. *org.apache.poi.hwpf.extractor.WordExtractor* class is used to extract the basic text such as lines or paragraphs. The word extractor of Apache POI API accepts POIFS or a HWPFDocument to read the text. The **getText()** method of word extractor can be used to get the text from all the paragraphs, or **getParagraphText()** can be used to fetch the text from each paragraph in turn. The extracted texts from the word file using Apache POI API is given as input text to java.util.regex API. The Keyword Extractor for SRS component workflow is shown in figure 2. In the SRS document the keywords are available as the Use cases. Keyword Extractor for SRS is going to match the word "Use Case" till the end of the line.

ISBN: 978-988-19251-5-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

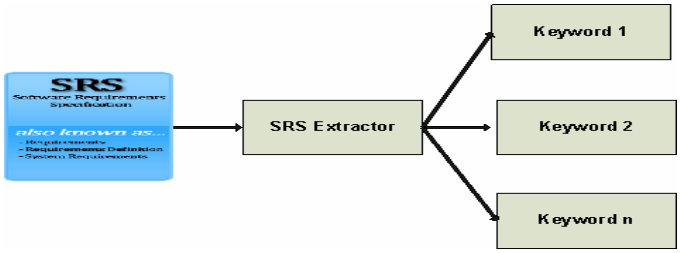

Figure 2. Process of Keyword Extractor for SRS

Regular Expression matching is a crucial task in several applications. Research interest has recently moved toward designing data structures, algorithms and architectures to support regular expression, which are more expressive than exact-match string and therefore able to describe a wide variety of pattern signatures [16][17]. The regular expression is a string which has to be compiled as a pattern. The matcher class attempts to match the entire input sequence against the pattern. The **find()** method of the matcher class scans the input sequence looking for the next subsequence that matches the pattern. The **group()** method returns the input subsequence matched by the previous match. The string use case is removed and the rest of the sentence is taken to form the keywords. From the use case name, each character is taken and checked for three conditions. The first one is whether it is a letter; if so it is concatenated to form a word. The second one is whether it is a whitespace; if so the concatenated word is stored in the keyword array. The last one is for the formatting characters; if so go for the next match. Finally the extracted keywords from the input text are stored in an array.

B. Description Model

Description model takes SRS text, which is pasted into the text box and extracts the necessary keywords to match the methods in the Ontology file. The process replaces all the non alphabetic characters by a white space. To remove the non alphabetic characters from the SRS text, a regular expression is used like "[^a-zA-Z]". The meaning of the regular expression is all characters except alphabets of small and caps to be removed. So the **removeAll(String regex, String targetString)** method of the String Class will remove all the match of the regular expression found in the SRS text with the target String that is white space. Next, the SRS text is compared with the irrelevant words. The words are listed in the "skip words" text file. Each word of the SRS text is compared with the words listed in the skip words file. If the word in the SRS text is matched with any unwanted words, the SRS word will be removed from the text. Finally the words in the SRS text will be checked for the existence in the Wordnet database. To check for the existence of the words in the SRS text a third party java library called *RiWordnet* is used[18]. *RiWordnet* provides library support for application and applet access to Wordnet. The **exists(String)** method of the *RiWordnet* takes a word and checks for the existence in the wordnet database. If the word is not found in the wordnet database, the SRS word will be removed from the SRS text. So the remaining words in the SRS text are considered as qualified keywords.

IV. METHOD DEFINITION SOURCE RETRIEVERS

Method Retriever component interacts with the Ontology and returns the available methods for the given keywords. The extracted information from the SRS document by the Keyword Extractor for SRS is passed to the Method Retriever component. It interacts with Ontology and retrieves matched method information using SPARQL query. Source Retriever component retrieves the appropriate source code of the user selected method from the HDFS. The source code file location of the Hadoop repository path is obtained from the Ontology and retrieved from the HDFS by the **copyToLocal(FromFilepath,localFilePath)** method. QDox is a high speed, small footprint parser for extracting class/interface/method definitions from source files. QDox finds the methods from the source code. The file that is retrieved from the HDFS is stored in the local temporary file. This file is passed to the QDox **addSource()** method for parsing. Using QDox each method is retrieved one by one. The retrieved methods are compared with the user requested methods. In Hadoop repository, the files are organized in the same hierarchy of java folder. So, it is easy to get the source location from the Ontology and store the java source file to a temp file. The temporary file is loaded into QDox to identify methods. Each method is compared with method to be searched. If it matches; the source code of the method is retrieved by **getMethodSourceCode()** method.

There are two processes in code reuse: Impression and Reuse. For a requested method, some methods are matched and listed. The user visits each method before deciding on reuse is called Impression. After going through the method code, a particular method's code is used that is called 'Reuse'. To keep track of these two processes whenever method is used as Impression or Reused, a record is created using MySQL. The structure of the record pertaining to the methods is the project, package and class from which the method is originated, the developer name, the data and time of development, whether the method is used as Impression or Reuse and the comment or review of the user about the method. The database structure will help to identify the usage of the method. The review of the record can help the users to further identify the credibility of the method.

V. CASE STUDY

The two variants of Keyword Extraction from SRS are implemented. But to curtail the length of the paper the implementation of the Description model of SRS is presented in this section. The Sample SRS input is given below:

The CISWAAD web site will be operated from the departmental server. When an Alum connects to the University Web Server, the University Web Server will pass the Alum to the Departmental Server. The Departmental Server will then interact with the Alumni Database through BDE, which allows the Windows type program to transfer data to and from a database.

2.2. Functional requirements definitions

Functional Requirements are those that refer to the functionality of the system, i.e., what services it will provide to the user. Nonfunctional (supplementary) requirements pertain to other information needed to produce the correct system and are detailed separately.

2.3. Use cases

The system will consist of CIS Alumni Home page with five selections. The first selection is to fill out a survey. The questions on the survey will be created by a designated faculty member. The survey will ask the Alum questions concerning their degree, job experience, how well their education prepared them for their job, and what can the CIS department do to improve itself. This information will be retained on the departmental server and an e-mail will be sent to the designated faculty member.

The second selection is to the Entries section. There are two choices on this page. One choice is to add a new entry. A form is presented to the Alum to be filled in. Certain fields in the form will be required, and list boxes will be used where appropriate. A password typed twice will be required of all new entries. The second selection of the Entries page is to update an Alum entry. A form will be presented

ISBN: 978-988-19251-5-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

allowing the Alum to enter their year of graduation and then to select themselves from a list. A password will be required before the information will be presented to the Alum to be updated.

The third selection is to search or e-mail an Alum. A form will be presented requiring the requested Alum's year of graduation. The requesting Alum will search a table to see if the requested Alum is in the database, and if so non-sensitive information will be returned. At this time the Alum can select to e-mail the Alumnus or search for another Alumnus. If the Alum chooses to e-mail the Alumnus a form will be presented for the message to be entered with the sending Alum's name and e-mail. The message, with all necessary information will be forwarded to the requested Alum. The e-mail address of the requested Alum will not be seen by the sending Alum as a privacy measure. All pages will return the Alum to the CIS Alumni Home Page.

The entire SRS text is copied from the source file and pasted in the text box of the interface tool as shown in Figure 3.

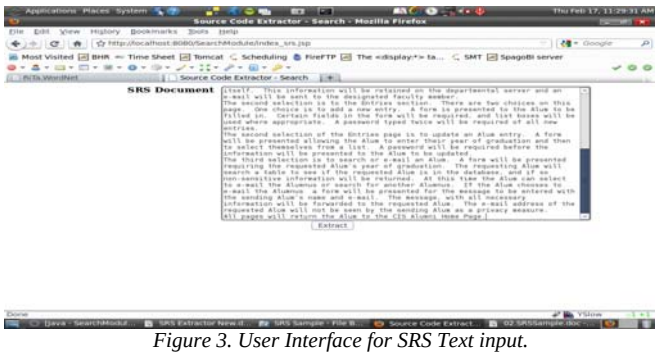

Figure 3. User Interface for SRS Text input.

TABLE I
METHODS MATCHED FOR THE KEYWORD PROGRAM

S.No	Information	
Keyword : program		
1	Project Name	LIMS 1.0
	Package	com.lims.beans
	Class Name	Memberbean
	Method	
	Name	setProgram
	Parameters	program
	Return Type	void

The number of keywords extracted by the process is 124. The Extracted Keywords are given below:

filled time correct created refer page search selections year connects information transfer services form privacy questions job windows table third requested system separately ask faculty home choice data operated use alum graduation twice allowing nonfunctional detailed entries alumnus sensitive member site cases requesting choices pages return web supplementary seen selection sending new degree interact returned produce provide second functional entered requiring used type password sent updated survey required concerning presented fields retained departmental appropriate consist select designated enter forwarded definitions program prepared update other measure can department one pertain well improve education add requirements chooses typed another allows name cis experience mail functionality university five boxes fill list section database needed pass message two see address entry there server necessary certain first user out

The initial test is done with Liturgy Information management System project, 108 methods are matched for the above keywords from the Ontology. The output of Keyword Extractor from SRS is given to the Method Extractor and generates the SPAQL query and extracts the matched methods. For the keyword **Program** more than ten methods are matched, but only one matched method detail is presented in Table I. It has matched ten methods in various projects. From the list, the appropriate method will be selected and the QDox retrieves the source code from the HDFS and displays the method definition.

VII. MEASURING THE CODE REUSE

This section deals with the metrics and models of software reuse. A metric is a quantitative indicator of an attribute of a thing. A model specifies relationships among metrics. Many measurable impacts of software reuse are available, out of which Reuse Density, Quality and Growth of the Knowledgebase are used in this paper.

A. Reuse Metrics

Many software reuse metrics are available such as reuse level, reuse frequency and reuse density. Reuse Level is a metric [19] that calculates the number of methods reused in the project related to the total number of methods in the knowledgebase. It is one of the simplest and well-known reuse metric. For every new project the matched methods from Ontology are retrieved and related to the total number of methods to be developed. If a new project needs 100 methods to be developed, all are matched with the repository and matches for 30 methods. The reuse level will be (30/100) = 0.30. Also Reuse percentage can be calculated by multiplying the reuse level by hundred. From the above reuse level value 0.30 * 100 give 30 percent. The reuse percentage shows the amount of reuse in the new project. When the reuse percentage goes higher and higher the resources are used more from the knowledgebase. It shows that the software development cost will come down if the reuse percentage is higher.

Reuse Frequency is a metric that calculates the number of references to reused items related to the total number of references. Reuse frequency is highly correlated to the Reuse Level metric [20]. In a project, a method matches with repository and lists 50 methods for the user's choice. The number of methods visited for a given method is 7. Reuse Frequency is (7/50) = 14. The reuse frequency shows the strength of the knowledgebase. If the reuse frequency is more for the given method many retrieved methods are relevant. Reuse Density is a metric [21] that measures the number of reused method related to the total number of instructions. Reuse Density, Reuse Level and Reuse Frequency all are related so the Reuse Density metric alone is used in Table II. To test the performance of this framework, the reusable Ontology files are created by uploading the completed projects. The first Ontology file is uploaded with first java project. The second Ontology file is uploaded with first and the second java projects. The third Ontology file is uploaded with first, second and third java projects. Similarly five Ontology files are constructed. The purpose of creating Ontology is to show how Reuse Density increases when the knowledgebase grows. The first entry in Table II shows the worst case scenario where only 32 lines of code is reused compared to the 1320 of total lines of code gives the 0.02878787 as reuse density. The average case has 372 lines of reuse code to the 6740 total code that gives the reuse density as 0.5519287. In the same way, the best case reuse density is 0.6159695 for 972 lines of reuse code to 15780 total lines of code. When the reuse code is more the reuse density will also get increased. Figure 4 shows the relationships between

ISBN: 978-988-19251-5-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

TABLE II
REUSE DENSITY FOR THE VARIOUS KNOWLEDGE BASES

Knowledge Base	Searched Methods	Searched Methods Lines of Code	Total Lines of Code	Reuse Density
1	2	38	1320	0.02878787
2	5	137	2890	0.04740484
3	11	372	6740	0.05519287
4	13	688	11340	0.06067019
5	16	972	15780	0.06159695

searched methods lines of code, total lines of code and reuse density. The Ontology knowledgebase is shown in X-axis and the lines of code in the Y-axis. The graph and the table show that the reuse density increases when more number of projects are uploaded in to the Ontology.

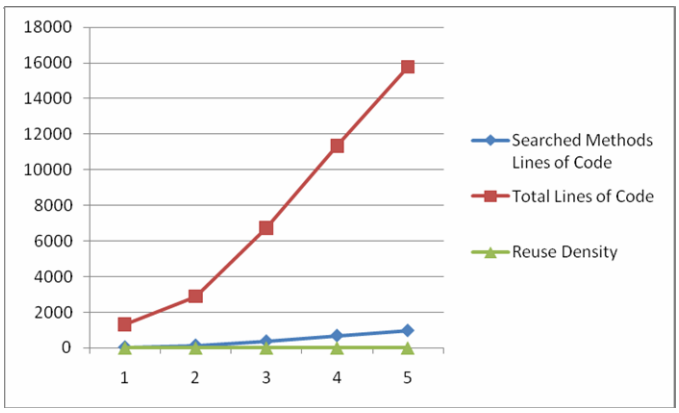

Fig. 4. Reuse Density

B. Quality

The fundamental cause of "software bottleneck" is that new software systems are usually developed from the scratch. Software Reuse not only improves productivity: it also has a positive impact on the quality and maintainability of software products [22]. It is generally assumed that the reuse of existing software will enhance the reliability of a new software application. Potential quality attributes include: reusability, maintainability, accuracy, clarity, replaceability, interoperability, scalability, performance, flexibility, adaptability, and reliability. The Quality of the software can be measured using the following formula:

Quality = 100 – Defect Density * New Code

To calculate the quality expected, quality is always hundred percent. The Defect density is a measure of the total known defects divided by the size of the software entity being measured. Normally for the MNC's it will be 5% to 10%. Normal companies will have 20% defect density. Table III shows how the quality gets increased when using reusable code. The first entry in that table shows the worst case scenario where there is no code is reused so the quality percentage is 80 percent. It is because of the 20 percent defect density considered for all the entries. The average case has 1000 lines of new code and the 1000 lines of reuse code that makes the 90 percent quality. The best case quality percentage is 100 percent for the no new code. This 100 percent for the best case is possible because the reuse code will not have the defect density. So, more usage of reuse code will bring better quality to the software process.

TABLE III
QUALITY WITH REUSE

Project Code	Defect Density Percentage	New Code	Reuse Code	Quality Percentage
1000	20	1000	0	80
2000	20	1600	400	84
2000	20	1000	1000	90
3000	20	1000	2000	93.33
4000	20	0	4000	100

The Quality of software reuse for the various scenarios is presented diagrammatically in Figure 5. The graph shows the quality percentage progresses with the higher reuse code. The X-axis represents the various projects and the Y-axis represents the lines of code. To have better quality in the software the reuse will become inevitable.

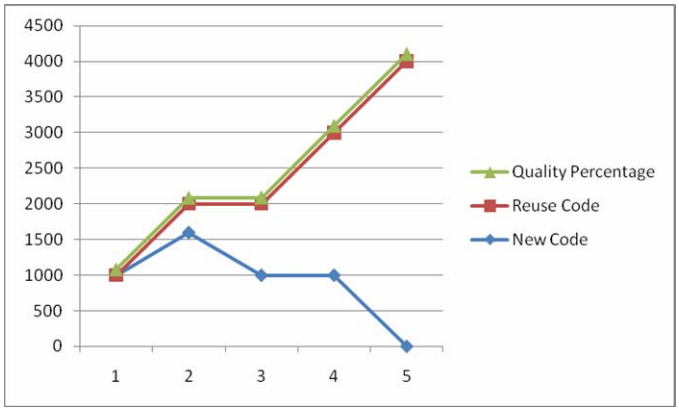

Fig. 5. Quality percentage for New and Reuse code

C. Knowledgebase Growth

Five completed projects are uploaded in to Ontology and a new project is going to be uploaded. Using this framework, the number of methods already available in the repository is counted and it is going to be redundant method but, with different process. So these remaining methods are going to add knowledge to the repository. By uploading many new projects to Ontology the knowledgebase grows. Ontology consists of five projects and five new complete projects are going to be uploaded. The knowledge strength is shown in Table IV. The first entry in the table shows the worst case scenario where 100 methods are going to get uploaded out of which only 15 methods are matched with the exiting methods. They become the redundant methods with different process which are stored in the repository. In the same way the remaining 85 methods become the new to the Ontology. The knowledge growth for this scenario is 85 percent. The average case has 80 methods are to be uploaded out of which 42 methods are already available and 38 methods are new to the Ontology. The knowledge growth for this case is 47.5 percent. The best case knowledge growth is 43.33 percent when the total methods are 180, the number of methods in Ontology is 102 and the number of new methods to Ontology is 78. When

ISBN: 978-988-19251-5-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

many new projects are uploaded in to the storage the knowledge growth will get reduced. It shows evidently that most of the requested methods are already in the storage. The knowledge growth for the various scenarios is shown diagrammatically in Figure 6. The continuous upload of the

TABLE IV
KNOWLEDGE GROWTH IN OWL

New Project	Number of Methods in Project	Number of Methods in OWL	Number of Methods New to OWL
1.	100	15	85
2.	150	34	116
3.	80	42	38
4.	120	54	66
5.	180	102	78

projects to the repository decreases the knowledge growth percentage. The X-axis represents the various projects and the Y-axis represents the number of methods. The knowledge growth will tend to decrease because most of required methods are already available in the storage. By these three

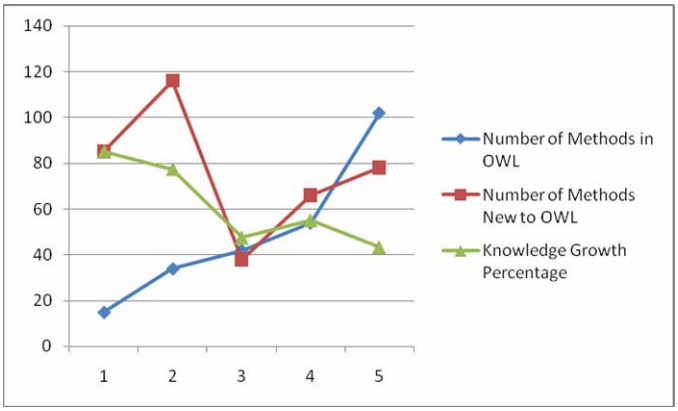

Fig. 6 Knowledge Growth for various Projects

measures, the software reuse claims that how the reuse code can perform better than the developed code.

VIII. CONCLUSION

The paper presents a framework to extract the method code components from the *Ontology* using the SRS document. After developing Ontology and storing the source code in the HDFS, the code components can be reused. With these sample tests the paper shows that it is indeed possible to extract code from *Ontology* using the SRS document. This paper has taken SRS from the user as input, extracted the method signature, searched and matched in the *Ontology*. The keywords can be used to search and match with the *Ontology* and the required method definition can be retrieved from the HDFS. The purpose of storing the metadata in *Ontology* is to minimize the factors like time of development, time of testing, time of deployment and developers. By creating *Ontology* using this framework can reduce these factors. The purpose of the paper is to achieve the code reusability for the software development. Using this tool, developer's progress and worth fullness can be assessed. Reuse could provide improved profitability, higher productivity and quality, reduced project costs, quicker time to market and a better use of resources. The software reuse considered in this paper deals only with the entire code of the method. The future work can take the partial usage and code modification in the extracted method's code in to account. A batch process can be created to monitor the completed project from the server to upload to the Ontology. After the method matching the list of methods are listed and chosen by the developer manually can be automated.

REFERENCES

[1]. Bung. M, *Treatise on Basic Philosophy. Ontology I*. The Furniture of the World. Vol. 3, Boston: Reidel.
[2]. Gaffney Jr., J. E,, Durek, T. A., "Software reuse - key to enhanced Productivity: Some quantitative models", Information and Software Technology 31(5): 258-267.
[3]. Banker, R. D., Kauffman, R. J., "Reuse and Productivity in Integrated Computer-Aided Software Engineering: An Empirical Study", MIS Quarterly 15(3): 374-401.
[4]. Basili, V. R.,Briand L. C., Melo, W. L., "How Reuse Influences Productivity in Object-Oriented Systems", Communications of the ACM 39(10): 104-116.
[5]. Boehm B.W., Pendo M., Pyster A., Stuckle E.D., and William R.D., "An Environment for Improving Software Productivity", In IEEE Computer, June 1984.
[6]. Paul R.A., "Metric-Guided Reuse", In proceedings of 7th International Conference on tools with artificial Intelligence (TAI'95), 5-8 November, 1995, pp. 120-127.
[7]. Poulin Jeffrey S., "Measuring Software Reusability", In proceedings of 3rd International Conference on Software Reuse, Brazil, 1-4 November 1994, pp. 126-138.
[8]. Gopinath Ganapathy and S. Sagayaraj, "Studies on Architectural Aspects of Searching using Semantic Technologies", in International Journal of Research and Reviews in Computer Science,Vol.1, No. 2, June 2010, pp.119-126.
[9]. Gopinath Ganapathy and S. Sagayaraj, "Automatic Ontology Creation by Extracting Metadata from the Source code", in Global Journal of Computer Science and Technology,Vol.10, Issue 14(Ver.1.0) Nov. 2010, pp.310-314.
[10]. Gopinath Ganapathy and S. Sagayaraj, "Extracting Code Resource from OWL by Matching Method Signatures using UML Design Document", *in* International Journal of Advanced Computer Science and Applications(IJACSA), Volume 2 No 2, February 2011, pp. 90-96.
[11]. Jason Venner, *Pro Hadoop : Build Scalable, distributed Applications*, in the cloud, Apress, 2009.
[12]. Gopinath Ganapathy and S. Sagayaraj, "Circumventing Picture Archiving and Communication Systems Server with Hadoop Framework in Health Care Services", in Journal of Social Science, Science Publication 6 (3) , pp.310-314.
[13].Tom White, *Hadoop: The Definitive Guide*, O'Reilly Media, Inc., 2009.
[14]. Bugaite, D., O. Vasilecas., "Ontology-Based Elicitation of Business Rules", In A. G. Nilsson, R. Gustas, W. Wojtkowski, W.G. ojtkowski, S. Wrycza, J. Zupancic Information Systems Development: Proc. of the ISD'2004. Springer- Verlag, Sweden, 2006, pp. 795-806.
[15]. Gopinath Ganapathy and S. Sagayaraj, "To Generate the Ontology from Java Source Code", *in* International Journal of Advanced Computer Science and Applications(IJACSA), Volume 2 No 2, February 2011, pp. 111-116.
[16]. R. Sommer and V.Paxson, "Enhancing byte-level network intrusion detection signatures with context", in CSS 2003.
[17]. J. Newsome et al., "Polygraph: Automatic Signature Generation for Polymorphic worms", in IEEE Security & Privacy Symp. 2005.
[18]. Daniel C. Howe., *RiTa: Creativity Support for Computational Literature*, in CHI 2008, Florence, Italy, 2008.
[19]. William Frakes, Carol Terry, *Software Reuse and Reusability Metrics and Models*", 1995.
[20]. W Curry, G Succi, M Smith, E Liu, R Wong, *Empirical Analysis of the Correlation between Amount-of-Reuse Metrics in the C Programming Language*. 1999.
[21]. Benedicenti, L., G. Succi, T. Vernazza, *Guidelines to Determine the Impact of Code Reuse on Productivity*, 1997.
[22]. Poulin Jeffrey S, "Measuring Software Reusability", In proceedings of 3rd International Conference on Software Reuse, Brazil, 1-4 November 1994, pp. 126-138.

ISBN: 978-988-19251-5-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)