CHAPTER 10
CONCLUSION AND FUTURE DIRECTIONS

The work achieved the task of automatic generation of ‘method ontology’ semantically derived from various Java source code. The thesis has analyzed the difficulties in the management of software reuse and the problems in automatic generation framework. This framework, has analyzed, identified and integrated many open source tools and frameworks for the creation of automatic method ontology. There are three main topics where this technology can improve the code reuse limitations:

i. automatic ontology creation for methods of Java codes.
ii. methods are matched by extracting methods from UML class diagram
iii. methods are matched by extracting keywords from SRS Use Case Model
iv. methods are matched by extracting keywords from SRS Description Model

The first topic is a unique and innovative piece of process in the code reuse engineering. It targets an ontology repository where the software enterprises can store their methods in semantic structure and the projects are stored in the Hadoop framework to facilitate the huge storage and the distributed environment. Before the automatic creation of ontology generation the work has also addressed the manual creation of method ontology. The methods are extracted automatically using document generator from the java source and converted to XSD using Trang tool. The manual process converted XML to OWL. This module could help to create upcoming projects by reusing the existing methods from the repository. Consequently it could lower the barrier to software development.

The second topic aims to match the methods extracted from the UML class diagram and converts it into method signatures using a parser. The methods are passed as a query to the ontology and the matched methods are listed. The user could select appropriate methods and the code is extracted from Hadoop for the new project
development. In absolute terms, it could provide a complete automation in identifying how many methods could be reused from the UML class diagram.

The final topic aims at extracting the keywords from the two models of SRS documents. The Use Case model has used the API’s for the keyword extraction and matched with the ontology. The Description model has used the WordNet for the keyword extraction and matched with ontology. For both forms of model lists the matched methods and the user could select the appropriate methods for the new project development. Hence this SRS document could come much earlier than the UML class diagram in the process of Software development so that the Project Leader can estimate and allocate the resources for the forthcoming projects.

Finally implementations are provided to automatic method ontology creation, matching from the methods extracted from UML class diagram and the two models of SRS documents. The implementation phase also displayed the statistics of methods available in method ontology.

The thesis started with the evaluation of more relevant works on systems aiming at automatic ontology generation. Throughout the analysis the researcher has highlighted some limitations of the current system. In particular, current system usually provide part of the whole ontology generation process only; the generation process is often done over a collection of text documents.

Concerning the approach, the work has observed that systems adopting a framework approach with the integration of an intermediary semantic model perform the automation of the ontology generation better. The research work shows that, even though few ontology generation systems start from internal sources, the extraction of ontological knowledge from source code is viable, as proved by the research out come, as below:

- Showing that methods extracted from source code well fits the required semantics and structural knowledge to build an ontology manually and automatically;
• Showing that the matching of methods extracted from UML class diagram from the automatic methods ontology are able to automatically extract conceptual knowledge from such sources;
• Matching of methods extracted from Use Case and Description Models of SRS from the automatic methods ontology are able to automatically extract conceptual knowledge from Hadoop;
• Provides an implementation that brings together all elements such as creation of automatic method ontology, matching methods for UML class diagram with ontology, and matching of methods for the two models of SRS with ontology allowing the research to reach the goals.

Code development is more expensive to create phase in a project. This is shortened and accomplished faster after going through this framework. The automatic generation of ontology helps in avoiding the overheads in integrating reusable code in the development process. Maintaining the reusable resources are done with the help of Hadoop framework, which supports the multiple storage and distributed environment.

As per this research, software reuse results in improvements of quality, productivity, performance, and interoperability. Software Reuse provides a reduction in redundant work and thus development time yields a shorter time to market. It is important to consider the importance of timing and early availability of software systems. Additionally, documentation, cost, usage of resources, and team size is greatly reduced. Reusable components can provide an effective basis for quickly building a prototype of a software system.

10.1 EXTENSION OF RESEARCH

The research work has explained the implementation of Code Reuse to save effort, time and energy. Also it helps to improve the quality in Software Development. The extension work of this work might be the following:
• Integrating more advanced semantic web technologies remains to be investigated. For example, the integration of more specific matching and selection algorithms will improve precision and recall methods.
• A generic framework that can be constructed for any domain.
• Reuse of other artifacts such as scripts, and deployable code can be analyzed.
• This framework can be implemented in many fields such as health care where the patients’ information is transformed into ontology, and the entire case history is stored in Hadoop. As a result, the treatment history of a patient is made as Knowledge and reused for similar patients with same complaints.
• Research work could be implemented in various languages other than Java. By creating ontology for other languages for broader scope of a corporate.
• A generalized equation can be derived to calculate time and cost by considering complexity, capability of the developer and the technology.
• New project design document methods or keywords can be used to construct ontology, which can be matched with method ontology.

At the end, the research work has tied all the loose ends that were disjoint in the field of reusability engineering, by composing a simple framework. A fresh and new attempt has proven the reusability through semantics, which is a very innovative solution. This will pave the new way for further researchers, and a new dimension for Software Corporates.
APPENDIX A – CONSTRUCTED ONTOLOGY
# APPENDIX B – UML CLASS DIAGRAM

**Department**

- deptId : int
- DepartmentName : String

+ addDepartment(deptName : String): boolean
+ editDepartment(deptId : int, newDeptName : String): boolean
+ deleteDepartment(deptId : int): boolean

**OntoOperation**

- model : Model
- fileLocation : String

+ load(OWLPath : String) : Model
+ addInstances(instanceName : String) : Model
+ updateInstance(instanceName : String, owlmodel : Model)

**ExtractionManager**

- projectName : String
- projectVersion : float
- projectDescription : String
- projectArea : charset
- ProjectDate : datetime

+ getProjectPath() : String
+ ExtractionManager() : ExtractionManager
+ getMethod(filePath : File) : String
+ getClass(filePath : File) : string
+ getSource(methodName : Manager) : String

**HadoopManager**

- HadoopPath : String
- HadoopUser : String
- Hadoopport : int

+ initiateConnection()
+ uploadFile(fileName : String, HadoopPath : String) : boolean
1.0. Introduction

1.1. Purpose

The purpose of this document is to present a detailed description of the Web Publishing System. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli. This document is intended for both the stakeholders and the developers of the system and will be proposed to the Regional Historical Society for its approval.

1.2. Scope of Project

This software system will be a Web Publishing System for a local editor of a regional historical society. This system will be designed to maximize the editor’s productivity by providing tools to assist in automating the article review and publishing process, which would otherwise have to be performed manually. By maximizing the editor’s work efficiency and production the system will meet the editor’s needs while remaining easy to understand and use.

More specifically, this system is designed to allow an editor to manage and communicate with a group of reviewers and authors to publish articles to a public website. The software will facilitate communication between authors, reviewers, and the editor via E-Mail. Preformatted reply forms are used in every stage of the articles’ progress through the system to provide a uniform review process; the location of these forms is configurable via the application’s maintenance options. The system also contains a relational database containing a list of Authors, Reviewers, and Articles.

2.0. Overall Description

2.1 System Environment

The Web Publishing System has four active actors and one cooperating system. The Author, Reader, or Reviewer accesses the Online Journal through the Internet. Any Author or Reviewer communication with the system is through email. The Editor accesses the entire system directly. There is a link to the (existing) Historical Society.

<< The division of the Web Publishing System into two component parts, the Online Journal and the Article Manager, is an example of using domain classes to make an explanation clearer. >>
2.2  *Functional Requirements Specification*

This section outlines the tasks for each of the active readers separately. The reader, the author and the reviewer have only one task apiece while the editor is main actor in this system.

Use case:  **Search Article**

**Brief Description**
The Reader accesses the Online Journal Website, searches for an article and downloads it to his/her machine.

**Initial Step-By-Step Description**
Before this the Reader has already accessed the Online Journal Website.

1. The Reader chooses to search by author name, category, or keyword.
2. The system displays the choices to the Reader.
3. The Reader selects the article desired.
4. The system presents the abstract of the article to the reader.
5. The Reader chooses to download the article.
6. The system provides the requested article.

An Author submits an article for consideration. The Editor enters it into the system and assigns it to and sends it to at least three reviewers. The Reviewers return their comments, which are used by the Editor to make a decision on the article. Either the article is accepted as written, declined, or the Author is asked to make some changes based on the reviews. If it is accepted, possibly after a revision, the Editor sends a copyright form to the Author. When that form is returned, the article is published to the Online Journal. Not shown in the above is the removal of a declined article from the system.

2.2.2  **Author**

In case of multiple authors, this term refers to the *principal author*, with whom all communication is made.
Use case: Submit Article

**Brief Description**
The author either submits an original article or resubmits an edited article.

**Initial Step-By-Step Description**
Before this the Author has already connected to the Online Journal Website.

1. The Author chooses the *Email Editor* button.
2. The System uses the *send to HTML* tag to bring up the user’s email system.
3. The Author fills in the Subject line and attaches the files as directed and emails them.
4. The System generates and sends an email acknowledgement.

2.2.3 Reviewer

Use case: Submit Review

**Brief Description**
The reviewer submits a review of an article.

**Initial Step-By-Step Description**
Before this can be initiated, the Reviewer has already connected to the Online Journal Website.

1. The Reviewer chooses the *Email Editor* button.
2. The System uses the *send to HTML* tag to bring up the user’s email system.
3. The Reviewer fills in the Subject line and attaches the file as directed and emails it.
4. The System generates and sends an email acknowledgement.
APPENDIX D – SRS DESCRIPTION MODEL

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.

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 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 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.
To Generate the Ontology from Java Source Code

OWL Creation

Gopinath Ganapathy 1
1 Department of Computer Science,
Bharathidasan University,
Trichy, India.
gganapathy@gmail.com

S. Sagayaraj 2
2 Department of Computer Science,
Sacred Heart College,
Tirupattur, India
sagisara@gmail.com

Abstract—Software development teams design new components and code by employing new developers for every new project. If the company archives the completed code and components, they can be reused with no further testing unlike the open source code and components. Program File components can be extracted from the Application files and folders using API’s. The proposed framework extracts the metadata from the source code using QDox code generators and stores it in the OWL using Jena framework automatically. The source code will be stored in the HDFS repository. Code stored in the repository can be reused for software development. By Archiving all the project files in to one ontology will enable the developers to reuse the code efficiently.

Keywords- component: Metadata; QDox, Parser, Jena, Ontology, Web Ontology Language and Hadoop Distributed File System;.

I. INTRODUCTION

Today’s Web content is huge and not well-suited for human consumption. An alternative approach is to represent Web content in a form that is more easily machine-processable by using intelligent techniques. The machine processable Web is called the Semantic Web. Semantic Web will not be a new global information highway parallel to the existing World Wide Web; instead it will gradually evolve out of the existing Web [1]. Ontologies are built in order to represent generic knowledge about a target world [2]. In the semantic web, ontologies can be used to encode meaning into a web page, which will enable the intelligent agents to understand the contents of the web page. Ontologies increase the efficiency and consistency of describing resources, by enabling more sophisticated functionalities in development of knowledge management and information retrieval applications. From the knowledge management perspective, the current technology suffers in searching, extracting, maintaining and viewing information. The aim of the Semantic Web is to allow much more advanced knowledge management system.

For every new project, Software teams design new components and code by employing new developers. If the company archives the completed code and components, it can be used with no further testing unlike open source code and components. File content metadata can be extracted from the Application files and folders using API’s. During the development each developer follows one’s own methods and logic to perform a task. So there will be different types of codes for the same functionalities. For instance to calculate the factorial, the code can be with recursive, non-recursive process and with different logic. In organizational level a lot of time is spent in re-doing the same work that had been done already. This has a recursive effect on the time of development, testing, deployment and developers. So there is a base necessity to create system that will minimize these factors.

Code re-usability is the only solution for this problem. This will reduce the development of an existing work and testing. As the developed code has undergone the rigorous software development life cycle, it will be robust and error free. There is no need to re-invent the wheel. Code reusability was covered in more than two decades. But still it is of syntactic nature. The aim of this paper is to extract the methods of a project and store the metadata about the methods in the OWL. OWL stores the structure of the methods in it. Then the code will be stored in the distributed environment so that the software company located in various geographical areas can access. To reuse the code, a tool can be created that can extract the metadata such as function, definition, type, arguments, brief description, author, and so on from the source code and store them in OWL. This source code can be stored in the HDFS repository. For a new project, the development can search for components in the OWL and retrieve them at ease[3].

The paper begins with a note on the related technology required in Section 2. The detailed features and framework for source code extractor is found in Section 3. The metadata extraction from the source code is in section 4. The metadata extracted is stored in OWL using Jena framework is in section 5. The implementation scenario is in Section 6. Section 7 deals with the findings and future work of the paper.

II. RELATED WORK

A. Metadata

Metadata is defined as “data about data” or descriptions of stored data. Metadata definition is about defining, creating, updating, transforming, and migrating all types of metadata that are relevant and important to a user’s objectives. Some metadata can be seen easily by users, such as file dates and file sizes, while other metadata can be hidden. Metadata standards include not only those for modeling and exchanging metadata,
but also the vocabulary and knowledge for ontology [4]. A lot of efforts have been made to standardize the metadata but all these efforts belong to some specific group or class. The Dublin Core Metadata Initiative (DCMI) [5] is perhaps the largest candidate in defining the Metadata. It is simple yet effective element set for describing a wide range of networked resources and comprises 15 elements. Dublin Core is more suitable for document-like objects. IEEE LOM [6], is a metadata standard for Learning Objects. It has approximately 100 fields to define any learning object. Medical Core Metadata (MCM) [7] is a Standard Metadata Scheme for Health Resources. MPEG-7 [8] multimedia description schemes provide metadata structures for describing and annotating multimedia content. Standard knowledge ontology is also needed to organize such types of metadata as content metadata and data usage metadata.

### B. Hadoop & HDFS

The Hadoop project promotes the development of open source software and it supplies a framework for the development of highly scalable distributed computing applications [9]. Hadoop is a free, Java-based programming framework that supports the processing of large data sets in a distributed computing environment and it also supports data intensive distributed application. Hadoop is designed to efficiently process large volumes of information[10]. It connects many commodity computers so that they could work in parallel. Hadoop ties smaller and low-priced machines into a compute cluster. It is a simplified programming model which allows the user to write and test distributed systems quickly. It is an efficient, automatic distribution of data and it works across machines and in turn utilizes the underlying parallelism of the CPU cores.

In a Hadoop cluster even while, the data is being loaded in, it is distributed to all the nodes of the cluster. The Hadoop Distributed File System (HDFS) will break large data files into smaller parts which are managed by different nodes in the cluster. In addition to this, each part is replicated across several machines, so that a single machine failure does not lead to non-availability of any data. The monitoring system then 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 [11] is a functional abstraction which provides an easy-to-understand model for designing scalable, distributed algorithms.

### C. Ontology

The key component of the Semantic Web is the collections of information called ontologies. Ontology is a term borrowed from philosophy that refers to the science of describing the kinds of entities in the world and how they are related. Gruber defined ontology as a specification of a conceptualization [12]. Ontology defines the basic terms and their relationships comprising an application of an application domain and the axioms for constraining the relationships among terms [13]. 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. Classes, subclasses and relations among entities are a very powerful tool for Web use.

A large number of relations among entities can be expressed by assigning properties to classes and allowing subclasses to inherit such properties. Inference rules in ontologies supply further power. Ontology may express rules on the classes and relations in such a way that a machine can deduce some conclusions. The computer does not truly “understand” any of this information, but it can now manipulate the terms much more effectively in ways that are useful and meaningful to the human user. More advanced applications will use ontologies to relate the information on a page to the associated knowledge structures and inference rules.

### III. SOURCE CODE EXTRACTOR FRAMEWORK

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 project information like description of the project, version of the project. The framework extracts the metadata from the source code using QDox code generators and stores it in the OWL using Jena framework. The source code is stored in the Hadoop’s HDFS. A sketch of the source code extractor tool is shown in “Fig. 1”.

Source code extraction framework performs two processes: Extracting Meta data from the source code using QDox and storing the meta-data in to OWL using Jena. Both the operations are performed by API’s. This source code extractor will integrate these two operations in a sequenced manner. The given pseudo code describes the entire process of the framework.

![Figure 1. The process of Semantic Stimulus Tool](image)

The framework takes project folder as input and counts the number of packages. Each package information is stored in the OWL. Each package contains various classes and each class has many methods. The class and method information is stored in the OWL. For each of method, the information such as return type, parameters and parameter type information are stored in the OWL. The framework which places all the information in the persistence model and it is stored in the OWL file.
extracted by the QDox. These metadata are passed to the next section for storing in the OWL.

V. Storing Metadata in OWL

To store the metadata extracted by QDox, the Jena framework is used. Jena is a Java framework for manipulating ontologies defined in RDFS and OWL Lite [15]. Jena is a leading Semantic Web toolkit [16] for Java programmers. Jena1 and Jena2 are released in 2000 and August 2003 respectively. The main contribution of Jena was the rich Model API. Around this API, Jena1 provided various tools, including I/O modules for: RDF/XML [17], [18], N3 [19], and N-triple [20]; and the query language RDQL [21]. In response to these issues, Jena2 has a more decoupled architecture than Jena1. Jena2 provides inference support for both the RDF semantics [22] and the OWL semantics [23].

Jena contains many APIs out of which only few are used for this framework like addProperty(), createIndividual() and write methods. The addProperty() method is to store data and object property in the OWL Ontology. CreateIndividual() creates the individual of the particular concepts. Jena uses in-memory model to hold the persistent data. So this has to be written in to OWL Ontology using write() method.

The OWL construction is done with Protégé. Protégé is an open source tool for managing and manipulating OWL[24]. Protégé [25] is the most complete, supported and used framework for building and analysis of ontologies [26, 27, 28]. The result generated in Protégé is a static ontology definition [29] that can be analyzed by the end user. Protégé provides a growing user community with a suite of tools to construct domain models and knowledge-based applications with ontologies. At its core, Protégé implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats. Protégé can be customized to provide domain-friendly support for creating knowledge models and entering data. Further, Protégé can be extended by way of a plug-in architecture and a Java-based API for building knowledge-based tools and applications.

Based on the java source code study the ontology domain is created with the following attributes. To store the extracted metadata, the ontology is created with project, packages, classes, methods and parameters. The project is concept that holds the information like name, project repository location, project version and the packages. The package is a concept that holds the information like name and the class. The class is a concept that holds the class informations such as author, class comment, class path, identifier, name and the methods. The method is a concept that holds the information like name, method Comment, method identifier, isConstructor, return type, and the parameter. The parameter is a concept that holds the information like name and the data type.

Concepts/Classes provide an abstraction mechanism for grouping resources with similar characteristics. Project, package, class, method, parameter are concepts in source code extractor ontology.

Individual is an instance of the concept/ class.
Property describes the relation between concepts and objects. It is a binary relationship on individuals. Each property has domain and range. There are two types of property namely object and data property.

Object Property links individuals to individuals. In source code ontology, the object properties are hasClass, hasMethod, hasPackage and hasParameter. hasClass is an object property which has domain Package and range Class. hasMethod is an object property which has domain class and range method. hasPackage is an object property which has domain Project and range Package. hasParameter is an object property which has domain method and range method.

Datatype Property links individuals to data values. Author is a datatype property which has domain Class and the String as range. ClassComment is a data property which has domain class and string as range. Data Type is a data property which has domain parameter and the range string as range. Identifier is a data property which has domain method, class and the range boolean as range. IsConstructor is a data property which has domain method and string as range. MethodComment is a data property which has domain method and string as range. Name is a data property which has domain project, package, class, method, parameter and string as range. Project_Date is a data property which has domain project and string as range. Project_Description is a data property which has domain project and string as range. Returns is a data property which has domain method and string as range. Project_Version is a data property which has domain project and string as range. Project_Repository_Location is a data property which has domain project and string as range.

VI. CASE STUDY

To evaluate the proposed framework the following simple java code is used.

```java

public class OntoManager {
  * @author Sagayaraj */

  public void createIndividual(OntModel model, String concept, String individual) {
    OntClass ontClass = model.getOntClass(addNameSpace(individual, model));
    model.createIndividual(ontClass); ontClass.createIndividual(ontClass.addNameSpace(individual, model));
    try { if (ontClass != null) {
      ontClass.createIndividual(ontClass.addNameSpace(individual, model));
    } else {
      LOGGER.error("Direct Class is null");// todo
    } finally {
      model.leaveCriticalSection();
    }
  }
}
```

The sample java code is given as input to QDox document generator through the Graphical User Interface (GUI) provided in the “Fig. 2”.

Using the QDox API’s metadata is extracted as given in the Table 1. The output of the QDox stores metadata in the form of strings. To store the metadata the OWL ontology, template is created using Protégé. The strings are passed to the Jena framework and the APIs place the metadata in to the Table 1. The output of the QDox stores metadata in the form of strings. To store the metadata the OWL ontology, template is created using Protégé. The strings are passed to the Jena framework and the APIs place the metadata in to the OWL Ontology. The entire project folder, stored in the HDFS, is linked to the method signature in the OWL ontology for retrieval purpose. The components will be reused for the new project appropriately. The obtained OWL Ontology successfully loads on both Protégé Editor and Altova Semantics. The sample OWL file is given below as the output of the framework.

```
<owl:Class rdf:about="http://www.owl-ontologies.com/SourceExtractor.j.owl#Package"/>
<owl:Class rdf:about="http://www.owl-ontologies.com/SourceExtractor.j.owl#Project"/>
<owl:Class rdf:about="http://www.owl-ontologies.com/SourceExtractor.j.owl#Method"/>
<owl:Class rdf:about="http://www.owl-ontologies.com/SourceExtractor.j.owl#Parameter"/>
<owl:Class rdf:about="http://www.owl-ontologies.com/SourceExtractor.j.owl#Parameter"/>
<owl:ObjectProperty rdf:about="http://www.owl-ontologies.com/SourceExtractor.j.owl#hasClass"/>
<owl:ObjectProperty rdf:about="http://www.owl-ontologies.com/SourceExtractor.j.owl#hasMethod"/>
```

Figure 2. GUI for locating folder

Select the Source Folder

```
/home/prathap/sourceExtractor/src/
```

Project Name

Project Description

Version of Project

Extract Close

Click here to Start Extraction

Figure 2. GUI for locating folder
VII. CONCLUSION AND FUTURE WORK

This paper presents an approach for generating ontologies using the source code extractor tool from source code. This approach helps to integrate source code into the Semantic Web. OWL is semantically much more expressive than needed for the results of our mapping. With these sample tests the paper argues that it is indeed possible to transform source code into OWL. After developing OWL and storing the source code in the HDFS, the code components can be reused. The future work can take off in two ways. One can take a design document from the user as input, then extract the method signature and try to search and match in the OWL. If the user is satisfied with the method definition, it can be retrieved from the HDFS where the source code is stored. Second one can take the project specification as input and text mining can be performed to extract the keywords as classes and the process as methods. The method prototype can be used to search and match the OWL and the required method definition can be retrieved from the HDFS. The purpose of storing the metadata in OWL is to minimize the factors like time of development, time of testing, time of deployment and developers. Creating OWL using this framework can reduce these factors.

REFERENCES


T. Berners-Lee et al.,” Primer: Getting into RDF & Semantic Web using N3”, http://www.w3.org/2000/10/swap/Primer.html


P. Hayes,” RDF Semantics”, 2004, W3C.


AUTHORS PROFILE

Gopinath Ganapathy is the Professor & Head, Department of Computer Science and Engineering in Bharathidasan University, India. He obtained his under graduation and post-graduation from Bharathidasan University, India in 1986 and 1988 respectively. He submitted his Ph.D in 1996 in Madurai Kamaraj University, India. Received Young Scientist Fellow Award for the year 1994 and eventually did the research work at IIT Madras. He published around 20 research papers. He is a member of IEEE, ACM, CSI, and ISTE. He was a Consultant for a 8.5 years in the international firms in the USA and the UK, including IBM, Lucent Technologies (Bell Labs) and Toyota. His research interests include Semantic Web, NLP, Ontology, and Text Mining.

S. Sagayaraj is the Associate professor in the Department of Computer Science, Sacred Heart College, Tirupattur, India. He did his Bachelor Degree in Mathematic in Madras University, India in 1985. He completed his Master of Computer Applications in Bharathidasan University, India in 1988. He obtained Master of Philosophy in Computer Science from Bharathiar University, India in 2001. He registered for Ph.D. programme in Bharathidasan University, India in 2008. His Research interests include Data Mining, Ontologies and Semantic Web.
APPENDIX F
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 knowledge base 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.

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

Date of paper submission is March 6th 2011. Revised on April 18th 2011.
S. Sagayaraj is with Department of Computer Science, Sacred Heart College, Tirupur, 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).
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.

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 HWPDFocument 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.
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 CISSWAAD 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 CJS 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
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 not 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

<table>
<thead>
<tr>
<th>S.No</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Keyword : program</td>
</tr>
<tr>
<td></td>
<td>Project Name</td>
</tr>
<tr>
<td></td>
<td>Package</td>
</tr>
<tr>
<td></td>
<td>Class Name</td>
</tr>
<tr>
<td></td>
<td>Method</td>
</tr>
<tr>
<td></td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td>Parameters</td>
</tr>
<tr>
<td></td>
<td>Return Type</td>
</tr>
</tbody>
</table>

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

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 0.0519287. 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

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.
TABLE II

<table>
<thead>
<tr>
<th>Knowledge Base</th>
<th>Searched Methods</th>
<th>Searched Methods Lines of Code</th>
<th>Total Lines of Code</th>
<th>Reuse Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>38</td>
<td>1320</td>
<td>0.02878787</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>137</td>
<td>2890</td>
<td>0.04740484</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>372</td>
<td>6740</td>
<td>0.05519287</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>688</td>
<td>11340</td>
<td>0.06067019</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>972</td>
<td>15780</td>
<td>0.06159695</td>
</tr>
</tbody>
</table>

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.

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:

\[
\text{Quality} = 100 - \text{Defect Density} \times \text{New Code}
\]

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 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. 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

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.

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

TABLE III

<table>
<thead>
<tr>
<th>Project Code</th>
<th>Defect Density Percentage</th>
<th>New Code</th>
<th>Reuse Code</th>
<th>Quality Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>20</td>
<td>1000</td>
<td>0</td>
<td>80</td>
</tr>
<tr>
<td>2000</td>
<td>20</td>
<td>1600</td>
<td>400</td>
<td>84</td>
</tr>
<tr>
<td>2000</td>
<td>20</td>
<td>1000</td>
<td>1000</td>
<td>90</td>
</tr>
<tr>
<td>3000</td>
<td>20</td>
<td>1000</td>
<td>2000</td>
<td>93.33</td>
</tr>
<tr>
<td>4000</td>
<td>20</td>
<td>0</td>
<td>4000</td>
<td>100</td>
</tr>
</tbody>
</table>

The Quality of software reuse for the various scenarios is presented diagrammatically in Figure 4. The graph and the table show that the reuse density increases when more number of projects are uploaded in to the Ontology.
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 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 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