A Novel Software Reuse Method – An Ontological Approach

R.Jayasudha, S.Subramanian, L.Sivakumar

Abstract—Software Industries develop various projects in various domain and store it in the disk as archives. These resources are not used to its fullest for the future reuse because of unstructured storage and retrieval methods. In this paper the Ontology based Storage and retrieval is proposed. The developer uses domain based ontology for understanding the domain of the project and the relevant semantic based keyword is used for retrieving the needed reusable artifacts. The folder ontology is the index to the actual archives where the needed data is stored. The retrieved component is finally updated and integrated for successful reuse. By doing this process the time, manpower system resources and cost will be reduced in Software development

Index Terms— Software Engineering, Software Reuse, Semantic Web, Ontology, Information Retrieval

1. INTRODUCTION

Software reuse is the process of developing a software from the already existing software components. Software Components means any software artifacts such as software design software coding, software test cases etc. Software reuse is a very old concept, but it does not play an vital role because of unavailability of the reusable components. Reusable components are the one which is reused. There is no benchmark for the components to be reused. Certification of reusable components has to be done to have a secure component reuse.

Reuse can be for or with reuse. Development for reuse is the concept of developing a software component in an generalized way so it can be reused for the further projects. All the aspects of the reusable components are taken into consideration and the reusable components are developed.

Development with reuse is the concept of actual reuse where in the reuse of actual software component has been taken place. The Software's developed with the existing software reusable components.

The advantages of Software reuse are it saves the cost, reduces the effort, reduces bug. The main goal

of Software Reuse is to reduce cost of production by replacing creation with recycling. The main problem with the reuse are identification of reusable components, Storage of reusable components, Knowledge based reusable repositories, Searching the reusable components, Modification and integration of reusable components with the current software development.

In this paper the various software reuse methods has been discussed. This paper has been classified into three sections, Section 1 describes about the different types of existing software. Section 2 describes about the proposed method and its advantages, Section 3 gives the result and discussions.

- *Jayasudha.R, Phd Scholar, Anna University, Chennai, Tamilnadu, India jayasudhasubburaj@gmail.com*
- *Dr.S.Subraminian, Vice Chancellor, Karpagam Univeristy, Coimbatore, Tamilnadu, India, drsraj@gmail.com*
- *Dr.L.Sivakumar, Vice Principal, Sri Krishna College of Engineering and Technology, Coimbatore, Tamilnadu, India, vp@skcet.ac.in*

2. LITERATURE SURVEY

Some of the other specific types of software reuse discussed by Ian Sommer in his book are

2.1 Application system reuse

The whole application system may be reused either by incorporating it without change into other Systems (COTS reuse) or by developing application families.

2.2 Component reuse

Components of an application from sub-systems to single objects may be reused.

2.3 Object and function reuse

Software components that implement a single well-defined object or function may be reused.

2.4 Design Patterns

Generic abstractions that occur across applications are represented as design patterns that show abstract and concrete objects and interactions.

2.5 Application Framework

Collections of abstract and concrete classes that can be adapted and extended to create application systems is called Application Framework..

2.6 Component based development
Systems are developed by integrating components that conform to component-model standards.

2.7 Aspect Oriented Software Development
Shared components are woven into an application at different places when the program is compiled.

2.8 Application product lines
An application type is generalized around a common architecture so that it can be adapted in different ways for different customers.

2.9 COTS Integration
Systems are developed by integrating existing application systems.

2.10 Legacy System Wrapping
Legacy system can be overwrapped by defining a set of interfaces.

2.11 Configurable vertical Application
A generic system is designed so that it can be configured to the needs of specific system customers.

2.12 Service Oriented System
Systems are developed by linking shared services that may be externally provided.

2.13 Program Libraries
Class and function libraries implementing commonly-used abstractions are available for reuse.

2.14 Program generators
A general system embeds knowledge of a particular types of application and can generate systems or system fragments.

In the [6] paper focused only on the CBS module of Knowledge-Based Tutoring System for Software Reuse Practices.

A CBS-SRRM provides software engineers with .a way to be tutored using positive lessons learned by their organizations. Our research focuses on achieving more effective means for software development organizations to find alternative educational (training) solutions to problems in software reuse practices. This system does not support distance learning and reuse self-assessment.

Domain analysis is the process of recording the commonalites and variabilities in a set of related software systems. Domain Implementation methods can take many forms including components, domain specific languages including little languages and application generators. All of these methods have been used in practice. Several examples of successful reuse component collections have already been described. An application generator creates a software system, or a large portion of a software system, based on a high level specification of the desired system. It shows that the generator encodes domain knowledge and design knowledge, and draws on components to produce code for a new system in a domain.

The paper introduced a methodology for applying machine learning techniques for systematic data exploration. Furthermore, it introduces a variation of an attribute selection technique, which is important in analysis of data with high dimensionality, such as the Reuse data set. One obstacle in the search for a software reuse model is the scarcity of the data.

While BTC's results and their specific business needs may be unique, it is likely that the business and technology practices supporting reuse may be generalizable to other banks and other technology users. Good system architecture, supporting reuse, and an established business case that identify the business value of the reuse were fundamental to establishing the global reuse accomplished by BTC, and should be readily scalable to smaller and less global environments. Other research subjects within the banking industry may also be available and should be studied to identify commonalities and variations from the BTC/BigBank success model.

The growing interest of software reuse by software organizations makes adoption and evaluation of reuse an essential activity [9]. Many organizations struggle in their attempts to select appropriate reuse practices (methods, techniques and tools support) in their processes. In this paper various reuse assessment methods are evaluated.

In the paper [10], the architecture-centric software processes that results in traceable component model. It differs with the traditional software process models:

Firstly, architectural patterns are the key elements of software process and the means to express the work products of the different phases. Secondly, patterns are used to describe the development expertise and experience and become important parts of component model. Software reuse based on patterns means that the development expertise and experience is reused

also. Thirdly, the component relationships are specified explicitly, and the traceability between different models at different abstraction levels is created as the side effect of development process of the component model.

The main problem with the existing these methods are it increases the maintenance cost, there is lack of tool support, the programmers point of view not-invented here syndrome. Apart from this the main problem of unsuccessful software reuse is that the creation of component libraries or repository, retrieval of the needed relevant component, understanding and adaptability of the software components to the existing system.

3. PROPOSED SYSTEM

Ontology is an explicit specification of a conceptualization. The term is borrowed from philosophy, where an Ontology is a systematic account of Existence. For AI systems, what "exists" is that which can be represented. When the knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called the universe of discourse. This set of objects, and the describable relationships among them, are reflected in the representational vocabulary with which a knowledge-based program represents knowledge. Thus, in the context of AI, we can describe the ontology of a program by defining a set of representational terms. In such an ontology, definitions associate the names of entities in the universe of discourse (e.g., classes, relations, functions, or other objects) with human-readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms. Formally, an ontology is the statement of a logical theory.[1]

Ontologies are often equated with taxonomic hierarchies of classes, but class definitions, and the subsumption relation, but ontologies need not be limited to these forms. Ontologies are also not limited to conservative definitions, that is, definitions in the traditional logic sense that only introduce terminology and do not add any knowledge about the world (Enderton, 1972) . To specify a conceptualization one needs to state axioms that do constrain the possible interpretations for the defined terms.

3.1 ONTOLOGY BASED SOFTWARE REUSE

The new framework based Ontology Based Software Reuse (OBSR) which represents a concept as a node in ontology. Ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents. There are many ways to represent concepts and conceptual relationships in ontology. In this case, semantic network representation as in directed labeled graph. It is a simplified conceptual graph. Representing conceptual relationships as edges between nodes in the graph rather than representing them as nodes like in conceptual graphs. The detailed information on conceptual graphs can be found in related work section. Our ontology representation schema is defined below:

Definition: Our Ontology Representation

G = (V, E)

V = <Concept Identifier, Concept Label, [Concept Description] >

E = <u, v, relationshipName> where u,v Î V

Concept Identifier = <literals and numbers>

Concept Label = < string literal>

Concept Description = <character string>

G represents the semantic network graph consisting of vertices V as concepts and edges E as relationship among concepts. The Proposed technique currently using Code repository for ontology but the general architecture of the system applies to other kinds of ontology's as well.

3.2 SOFTWARE DEVELOPMENT HIERARCHY

Software development lifecycle plays a major role in the software development. The Fig 1 explains about the hierarchy of the software development process in such a way that at the end of every process, the knowledge was stored in the form of documentation which will be the source for the future reuse. The major software artifacts are reusable components are stored in Software Requirement Document, Design document, Source Code, Testing document, various other user manuals etc.,

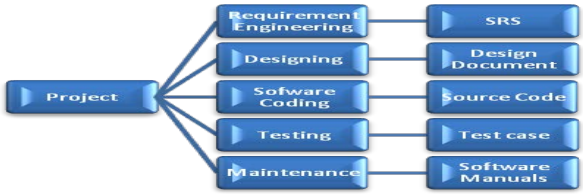

Figure 1 Software Development Hierarchy

3.3 Proposed Architecture

The Fig 2 proposes the noval architecture for the software development in order to make reuse a successful factor. The steps involved in the process are
Step 1: Requirement Engineering helps to form the requirement query for the current project.
Step 2: The Query analyzer helps o identify the domain of the project. The Domain Ontology helps to identify the various related terms in the domain.
Step 3: The refined query is now used for indentifying the needed component in the Folder Ontology which an index of the final storage of the data in the disk.
Step 4 : The retrieved objects are updated according to the current need of the project.
Step 5: Integration and the Implementation takes place once the needed reusable components are successfully retrieved.

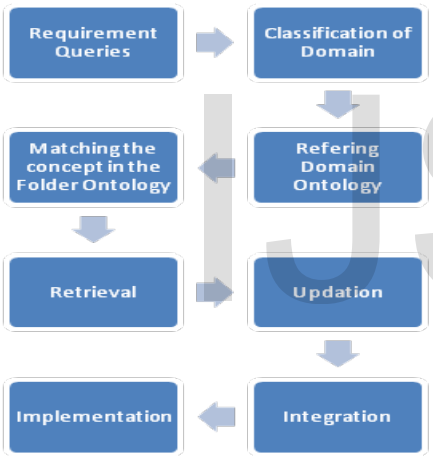

Figure 2. Ontology Based Software Reuse

The main advantage of this method is the usage of two different ontologies at the different phases namely domain ontology and the Folder ontology. Domain Ontology helps to retrieve the domain related term for the query. Folder ontology create the index of the actual storage of the data, which helps in the easy retrieval of the corresponding component.

4. Results and Discussion

The dataset for the experiment has been taken from the students final year project sets. Around 40 projects of different domain are taken and stored in the disk in the form of archives are used for discussion. Some of the reuse metrics which proposed by Davis [4] in his Reuse Capability Model are

5.1 Reuse proficiency (RP), which is the ratio of the value of the actual reuse opportunities exploited to the value of potential reuse opportunities,

5.2 Reuse efficiency(RE) , which measures how much of the reuse opportunities targeted by the organization have actually been exploited and
5.3 Reuse effectiveness (REF), which is the ratio of reuse benefits to reuse costs.

	RP(%)	RE(%)	REF(%)
Keyword Based	**25**	**35**	**50**
Ontology Based	**50**	**70**	**100**

Table 1 Key word based Vs Ontology Based Component Search (Reuse metrics)

From the table 1 it is depicted that the reuse proficiency, efficiency and effectiveness is less in the keyword based without using the semantic of the query and the ontology, whereas the Ontology based reuse gain more favorable reuse metrics.

Precision takes all retrieved documents into account, but it can also be evaluated at a given cut-off rank, considering only the topmost results returned by the system. This measure is called precision at n or P@n. High precision means that an algorithm returned substantially more relevant results than irrelevant. The table 2 depicts the comparison of the Keyword based and Ontology based retrieval precision. It is understood from the study that the Ontology based retrieval has got good precision over the other.

Domain Ontology has been created for the domain Banking, Payroll, Insurance, Inventory and Marketing. These five Domain Ontology is created using the tool Protégé and finally manipulated for the purpose of query expansion.

Query No	Query	Keyword Based	Ontology Based
Q1	Deposit Module	0.3	0.7
Q2	Payslip Generation	0.4	0.73
Q3	Premium Updation	0.45	0.8
Q4	Product Maintenance	0.56	0.5

Q5	Sales Report	0.6	1.0

Table 2 Key word based Vs Ontology Based Component Search (Precision)

The Graph shown in the fig 3 explains clearly the relationship between the Ontology based Software Reuse and the Keyword based. The Keyword based has got less precision which reduces the overall availability of the reusable components this leads to less reusability. The availability of the relevant reuse components are more in the case of the Semantic based reuse, thus improves the ability of the reusability in the software development life cycle.

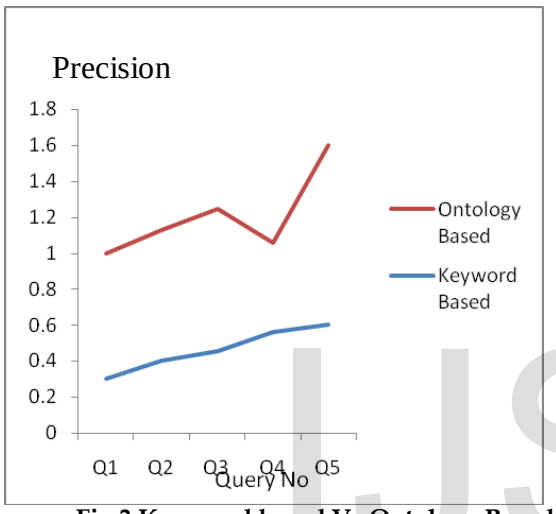

Fig 3 Key word based Vs Ontology Based

4 Conclusion

Software companies produce lots of reusable components or artifacts which is stored in the structured way. The unstructured storage leads to unsuccessful reuse, because of the unavailability of the needed knowledge. The proposed system solves the relevant problem by using the Ontology for the Semantic based retrieval and Storage.

The Domain and Folder Ontology helps in the finding the needed and available relevant reuse artifacts for the current project. In future the distributed based storage system is proposed in order to improve the reuse efficiency, even though the reusable knowledge are distributed geographically.

Acknowledgment

Our Sincere thanks to the management and Principal of the corresponding colleges for their support.

References

[1]. Aileen Cater-Steel Stephano Ah-Fock ," Reuse: Can it Deliver Competitive Performance?".
[2]. Anguswamy, R. and Frakes, W. B., An Exploratory Study of One-Use and Reusable Software Components, International Conference of Software Engineering and Knowledge Engineering, SEKE'12, San Francisco, USA, 1-3 July 2012.
[3]. Florinda Imeri1, Ljupcho Antovski,"An Analytical View on the Software Reuse", ICT Innovations 2012 Web Proceedings ISSN 1857-7288.
[4]. Hafedh Mili, Fatma Mili, and Ali Mili," Reusing Software: Issues and Research Directions", IEEE Transactions On Software Engineering, Vol 21, No. 6, June 1995
[5]. Mandava Kranthi K B M Konda Dr. K. Thammi Reddy B. Ravi Kiran,"A Systematic Mapping Study on Value of Reuse", International Journal of Computer Applications (0975 – 8887) Volume 34– No.4, November 2011 .
[6]. N. Nada, L.Luqi, M. Shing, D. Rine, E. Damiani, S. Tuwaim," Software Reuse Technology Practices and Assessment Tool-Kit".
[7]. Paul D. Witman Terry Ryan," Innovation in Large-Grained Software Reuse: A Case from Banking".
[8]. Rattikorn Hewett," Learning from Software Reuse Experience".
[9]. Vinicius Cardoso Garcia, Liana Barachisio Federal University of Pernambuco," Towards an Assessment Method for Software Reuse Capability", Eighth International Conference on Quality Software 2008.
[10]. Wang Hong," Architecture-Centric Software Process for Pattern Based Software Reuse", IEEE 2009.
[11]. William B. Frakes," Practical Software Reuse",IEEE 2000.
[12]. William B. Frakes and Kyo Kang,"Software Reuse Research: Status and Future", IEEE Transactions On Software Engineering, Vol. 31, No. 7, July 2005.