# An Efficient Software Engineering Ontology Tool for Knowledge Sharing

Polala Niranjan Reddy, Kukatlapalli Pradeep Kumar

HOD, Dept.of CSE**,** Kakatiya Institute of Technology and Science,
Warangal, A.P, India, 506015

Asst. Professor, Dept. of ECM, Narayana Engineering **C**ollege,
Nellore, A.P, India, 524004

## Abstract

Ontology is an important concept in Computer Science to formally represent knowledge in a way software can process the knowledge and reason about it. The software engineering ontology assists in defining information for the exchange of semantic project information framework. It intends to clear up the ambiguities that occur in the knowledge sharing between the software engineers. This paper presents the basic ontological representations for a given software project that which is well developed using the rudimentary software engineering principles. It draws a bead on an ontology model of software engineering to represent its knowledge. This paper also elicits about the analysis of SE Ontology and its advantages/applications with the example scenarios. Finally, a practical implementation of the in-depth ontological representation is elicited at the terminal of the paper with appropriate illustrations.

***Keywords:*** *Software Engineering, Ontology development, Multisite software development, Knowledge Sharing and Knowledge Engineering.*

## 1. Introduction

With the invent of the Internet, the development of the software in various fields, have become more cushy and comfortable. Realizing the pros of multisite software development, major MNC's and the corporate sectors have moved their business to the countries where the employees work for curtail and pare salaries. Software development has increasingly focused on the Internet, which enables a multisite environment that allows multiple teams residing across cities, regions, or countries to work together in a networked distributed fashion to develop the software. However, the effective communication and coordination across multiple sites is extremely important for the global software development. Team members, team leaders and the managers who carry out, control, manage different tasks and activities respectively may not be located at the same site in a multisite environment. Consider a scenario of a software development process where the team members work in a particular site and the person who manage them, their team leader is at different site who controls them and collects, integrates the completed modules for further enhancement of the software project. As the team completes their respective module and send the same to the team leader. They draft in their own form of representations for conclusions on the completed module with respect to their culture, customs and tradition which they follow in their day to day life. It is obvious that they might have not come face to face and never met as they work online. So, strict software engineering principles should be followed, to have a better communication among the teams and the team members. The incongruity in analysis, design, documentation, presentation, and diagrams could prevent proper access by other stake holders in a particular software project. Seldom issues of this kind are kept enigmatic. In reference to the above discussed problems, the software engineering has a commonly understood body of knowledge and is an easily learnt subject that includes some of the latest technology and methodology that is easily adopted. As the teams at different sites refer to various texts in the same software engineering domain, each individual have a personal guide and when they communicate with each other their terminology could be quite startling and unusual. This leads to inconsistency and equivocation among the teams. Communication is the real challenge that everyone face in their daily life and the affective communication is an important part of a successful business. Ontology is an important part of developing a shared understanding across a project to lessen the problems.

This paper is organized in five sections. The next Section describes software engineering and knowledge engineering as a part of related work. Section 3 presents proposed work. Section 4 deals with experimentation and results of a case study. Finally section 5 presents conclusions and future scope of the proposed work.

## 2. Related Work

Software engineering is the "application of a systematic, disciplined, and quantifiable approach to the development, operation, and maintenance of software". Although the claim of software development being an engineering discipline is subject to ongoing discussions, there is no doubt that it has undergone fundamental changes during the last three decades. This assertion holds true both for emergence of new technology and sophistication of methodology.

In order to cope up with the complexity in the software, there has been a constant drive to raise the level of abstraction through modeling and higher-level programming languages. For example, the paradigm of model-driven development proposes that the modeling artifacts are "executable", i.e. through automated validation and code generation as being addressed by the OMG Model Driven Architecture (MDA). However, many problems have only partially been solved including component reuse, composition, validation, information and application integration, software testing and quality. Such fundamental issues are the motivation for new approaches affecting every single aspect in Software Engineering.

The engineering of knowledge-based systems is a discipline which is closely related with Software Engineering. The term Knowledge Engineering is often associated with the development of expert-systems, involving methodologies as well as knowledge representation techniques. Since its early days the notion of "ontology" in computer science has emerged from that discipline, giving rise to Ontology Engineering, which we focus on in this paper. In computer science, the concept "ontology" is interpreted in many different ways and concrete ontologies can vary in several dimensions, such as degree of formality, authoritativeness or quality. As proposed by Oberle, different kinds of ontologies can be classified according to purpose, specificity and expressiveness. The first dimension ranges from application ontologies to reference ontologies that are primarily used to reduce terminological ambiguity among members of a community. In the specificity dimension, Oberle distinguishes generic (upper level), core and domain ontologies. Domain ontologies are specific to a universe of discourse, whereas generic and core ontologies meet a higher level of generality.

Due to the emergence of the "semantic web" vision ontologies have been attracting much attention recently. Along with this vision, new technologies and tools have been developed for ontology representation, machine-processing, and ontology sharing. This makes their adoption in real-world applications much easier, while ontologies are about to enter mainstream. Hence, we therefore try to alleviate some of the confusion by providing a framework for categorizing potential uses of ontologies in Software Engineering.

### 2.1 Ontology in Software Engineering

Ontology is the philosophical study of the nature of being, existence or reality in general, as well as the basic categories of being and their relations. Traditionally listed as a part of the major branch of philosophy known as metaphysics, ontology deals with questions concerning what entities exist or can be said to exist, and how such entities can be grouped, related within a hierarchy, and subdivided according to similarities and differences. In computer science and information science the ontology has a key role to play with the formal representation of the knowledge by a set of concepts within a domain and the relationships between those concepts.

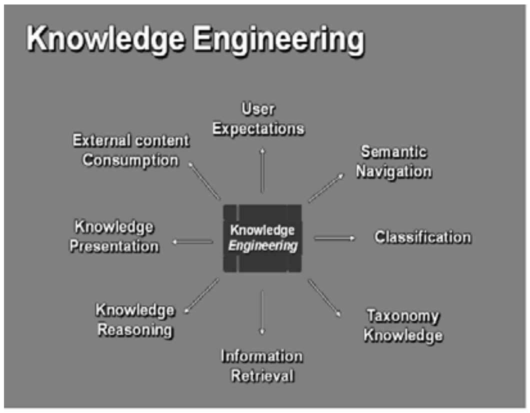


Figure1: The abstract view of Knowledge Engineering

It is used to reason about the properties of that domain, and may be used to describe the domain. Ontology provides a shared vocabulary, which can be used to model a domain — that is, the type of objects and/or concepts that exist, and their properties and relations. An Ontology in the field of Artificial Intelligence (AI) is an "Explicit Specification of a Conceptualization" [1][2]. Ontologies are used in artificial intelligence, the Semantic Web, Systems engineering, Software engineering, Biomedical informatics, Library science, Enterprise bookmarking, and Information architecture as a form of the knowledge representation about the world or some part of it. The basic abstract view of the knowledge engineering with all its outcomes is shown in fig 1.

The creation of domain ontologies is also fundamental to the definition and use of an enterprise architecture framework. The actual content and the domain are represented in the fig 2 with Semantic and the Pragmatic representations respectively. The content in the semantics (Actual meanings) area can be Stuff, Things, and Relationships.

The Domains in the pragmatic (Dealing or concerned with facts or actual occurrences) area can be Knowledge domain, Applications domain, and Functional domain. Combining both the Content and the Domain knowledge forms the basis for the Ontology. A simple and very regular ontological representation can be a standard library in a programming language environment which has all the methods, attributes, classes and packages that gives the answer for the preliminary question of "What Exists" in a programming language. However, some Representations may be poor due lack quality in design, implementation and so forth. So, a more specialized schema must be created to make the information useful, and for this, we utilize ontology.

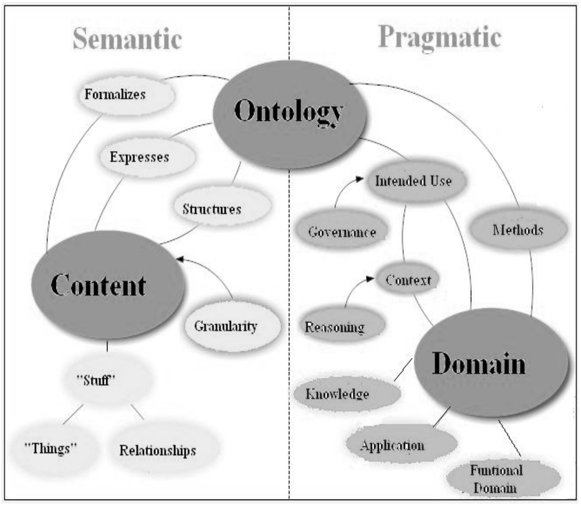


Figure2: Ontology with its 'Content' and the 'Domain' Concepts

An abstract view of representing the software engineering knowledge is shown in fig.3. The whole set of software engineering concepts representing software engineering domain knowledge is captured in ontology. Based on a particular problem domain, a project or a particular software development probably uses only part of the whole set of software engineering concepts. The specific software engineering concepts used for the particular software development project representing software engineering sub domain knowledge are captured in ontology.

Ontology in the area of computer science represents the effort to formulate an exhaustive and rigorous conceptual schema within a given domain [3]. The generic software engineering knowledge represents all software engineering concepts, while specific software engineering knowledge represents some concepts of software engineering for the particular problem domain. If a project uses purely object-oriented methodology, then the concept of a data flow diagram may not necessarily be included in specific concepts. Instead, it includes concepts like class diagram, activity diagram, and so on.

However, for each project in the developmental domain, there exists project information or actual data including project agreements and project understanding. The project information especially meets a particular project need and is needed with the software engineering knowledge to define instance knowledge in ontology.

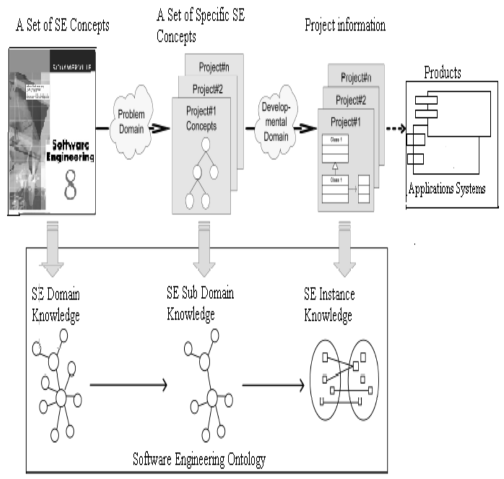


Figure 3: Schematic overview of the Software Engineering Ontology

Note that the domain knowledge is separate from instance knowledge. The domain knowledge is quite definite, while the instance knowledge is particular to the problem domain and developmental domain in a project. Once all domain knowledge, sub domain knowledge, and instance knowledge are captured in ontology, it is available for sharing among software engineers through the Internet.
The main purpose of the software engineering ontology is to enable communication between computer systems or software engineers in order to understand common software engineering knowledge and to perform certain types of computations; it also enables knowledge sharing and reuse.

The key ingredients that make up the software engineering ontology are a vocabulary of basic software engineering terms and a precise specification of what those terms mean. For software engineers or computer systems, different interpretations in different contexts can make the meaning of terms confusing and ambiguous, but a coherent terminology adds clarity and facilitates a better understanding. Software engineering ontology has specific instances for the corresponding software engineering concepts.

### 2.2 Developing Ontology

In the domain of knowledge engineering methodology for developing ontology, there are some fundamental rules in ontology design. These rules may seem rather dogmatic. However, these rules can often help in making design decisions.
- There is no one correct way to model a domain
- There are always viable alternatives. The best solution almost always depends on the application that you have in mind and the extensions that you anticipate.
- Ontology development is necessarily an iterative process.
- Concepts in the ontology should be close to objects (physical or logical) and relationships in your domain of interest. These are most likely to be nouns (objects) or verbs (relationships) in sentences that describe your domain.
Deciding what we are going to use the ontology for, and how detailed or general the ontology is going to be, will guide many of the modeling decisions down the road.

Pros of developing Ontologies:

- Share common understanding of information among people or agents
- Reuse of domain knowledge
- Make domain assumptions explicit
- Separate domain knowledge from operational knowledge
- Analyze domain knowledge

After we define an initial version of the ontology, we can evaluate and debug it by using it in applications or problem-solving methods or by discussing it with experts in the field, or both. As a result, we will almost certainly need to revise the initial ontology.

Then we can create a knowledge base by defining individual instances of these classes filling in specific slot value information and additional slot restrictions. However, the concept of the ‘Ontology’ exists in each and every domain and about all the phases of the software development process. As ‘class’ represent a real world entity, everything explained with the classes and their relationships. Various software engineering ontology modeling are elicited in the next sections with a case study for deeper evaluations.

## 3. Proposed Work

Many different modeling ontologies have been developed. Mostly used are the Knowledge Interchange Format (KIF) [4] and knowledge representation languages designed from KL-ONE

[5]. However, these representations have had little success outside. AI research laboratories and require a steep learning curve. KIF provides a Lisp-like syntax to express sentences of first order predicate logic and descendants of KL-ONE include description logics or terminological logics that provide a formal characterization of the representation Traditionally, AI knowledge representation has a linear syntax. The recent papers documented in the literature, to use the Unified Modeling Language for the ontology modeling [6][7][8]. In Unified Modeling Language ontology information is modeled in class diagrams and Object Constraint Language (OCL) [8]. However, there is controversy, regarding whether or not ontology goes beyond the standard UML modeling. However, the standard UML cannot express advanced ontology features such as constraints or restrictions. Therefore, additional notations need to be defined in order to leverage expressiveness in the ontology.
Note that the models underlying ontology should be distinguished from its use in software development to model the application domain model. This kind of agile modeling method for ontology design has some benefits derived from using the same paradigm for modeling ontology and knowledge. In this paper, graphical notations of modeling software engineering ontology are presented. The main aim is not only to create a graphical representation to make it easier to understand, but also, this model should be able to capture the semantic richness of the defined software engineering ontology.

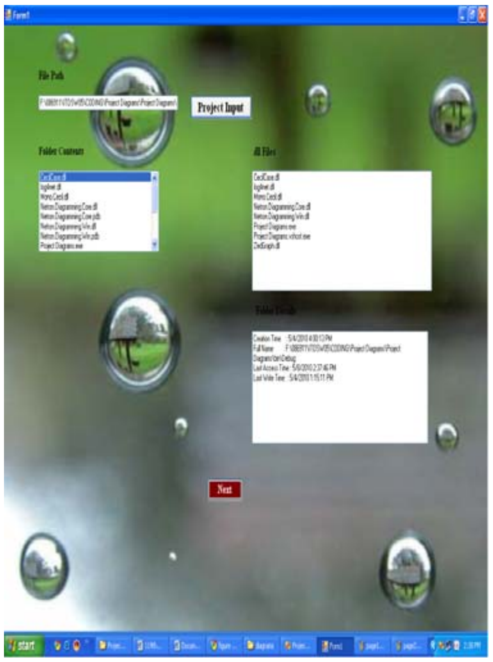
## 4. Experimentation and Results

This part of the section deals with a case study of the practical implementations with respect to the ontology basics. It is elicited with appropriate screen shots explaining the each and every module clearly. The main application works as follows.
This project is developed as a windows application on the Visual Studio framework version 3.5. This concentrates on the pictorial representation of the applications developed on the same domain. The application takes other software projects as input, i.e., by browsing from the current location. The appropriate compiled .exe file should only be selected as an input. After selecting the .exe file, the corresponding textboxes shows the selected folder details, .exe files etc. Loading the same; would provide the namespaces, number of classes, number of methods, number of parameters to it that are used in writing the code at the implementation phase of the project. The classes, methods, parameters are shown separately in the respective fields shown under.

The vivacious module that is developed is to draw the same concepts in a pictorial representation. It gives the hierarchical structure of the whole concepts that are used in the project. It also depicts the relation between the classes with the other classes. At last the schema representing the ontology can be saved as a JPEG, GIF, BMP or any other format.

This paves way for the basics of the Ontological representation in the software engineering. The main use of this application is that there is no need to walk through the entire thousands of lines code to analyze the project.

**Project input:**



Creation Time : 5/4/2010 4:30:13 PM
Full Name : [illegible]\Project Diagrams\Project Diagrams\bin\Debug
Last Access Time : 5/6/2010 2:37:46 PM
Last Write Time : 5/4/2010 1:15:11 PM

Figure 4: Project input module with the folder details and contents

However, it is always preferred to assay and explore the concepts which are in a diagrammatic representation. The same was elicited in the current project. The screen shots of the application are shown in the further sections.

Fig 4 is the project input module, the user is asked to provide the corresponding input to the application. The input may be any other application/project (the debug file). The project should posses a '.exe' file; which in turn mean that it should be a complete project or application that is in use. After giving the input, three text boxes are displayed, namely;

'Folder contents',
'.exe files' and
'Folder details'.

These are shown clearly within their respective text boxes. However, the 'folder details' field contains the creation time, full name, last access time, last write time of the folder where the actual application is installed.
This paves way for the basis of the ontology in the software projects. The first step of the ontological representations of the concepts of software engineering starts here. However, as mentioned earlier, the project/application selection process takes on, as shown in the figure 5.

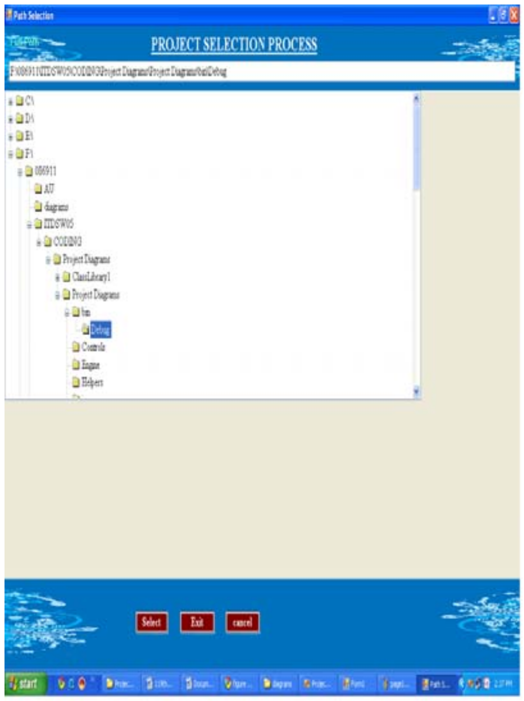


Figure 5: Application/ project selection process

It is developed with the help of the 'tree view control' in Visual Studio C# .Net. The Windows Forms Tree View Control helps to display the hierarchy of nodes that can be used to represent the organization structure, file system or any other system which includes hierarchical representation.

For each node added in the hierarchy, user can add a child node to it or a sibling node to it provided there is a parent node for the selected node present, as depicted in the fig5.

As this project is a windows application, the project/application selection is done by searching in the local drives. The three buttons namely; Select, Exit, Cancel can basically serve the user to navigate through the application. The same concept can be scaled to a web site application or can be inserted in a network (typical LAN).

**Project classification:**

Loading the corresponding application, the following are displayed with respect to the data available in the project, which is shown in the fig6 above.
Namespaces,
Class Names,
Methods,
Parameters.

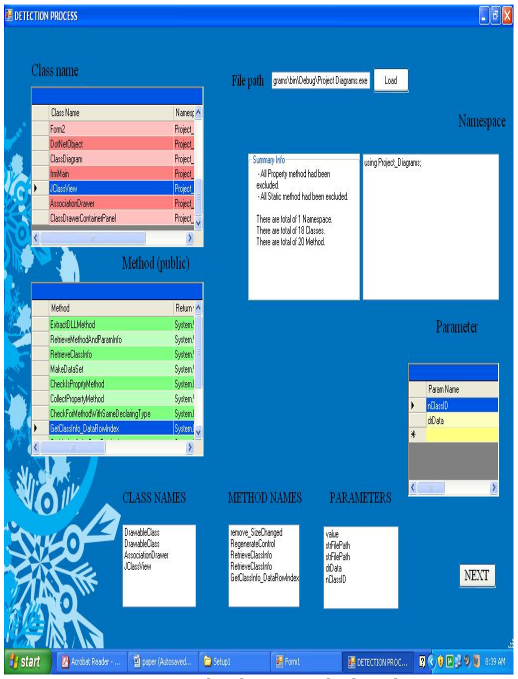
The Namespace that is used over here is the 'project_diagrams'.

Some of the Classes are
'Jclassview',
'form2',
'classdiagram',
'AssociationDrawer',
'ClassDrawercontainerpanel'.

Some of the Parameters that were used are 'nClassId', 'drData', 'value', 'strFilePath'.

Some of the Methods are

'ExtractDllMethod',
'RetriveMethodandParameterInfo',
'RetiveClassInfo',
'MakeDataSet',
'CheckIsProprtyMethod',
'CollectPropertyMethod',
'GetClassInfo_DataRowIndex'.



grams\bin\Debug\Project Diagrams.exe

| Class Name | Namesp |
|---|---|
| Form2 | Project_ |
| DotNetObject | Project_ |
| ClassDiagram | Project_ |
| frmMain | Project_ |
| JClassView | Project_ |
| AssociationDrawer | Project_ |
| ClassDrawerContainerPanel | Project_ |

Summary Info
- All Property method had been excluded.
- All Static method had been excluded.

There are total of 1 Namespace.
There are total of 18 Classes.
There are total of 20 Method.

using Project_Diagrams;

| Method | Return |
|---|---|
| ExtractDLLMethod | System.' |
| RetrieveMethodAndParamInfo | System.' |
| RetrieveClassInfo | System.' |
| MakeDataSet | System.' |
| CheckIsProprtyMethod | System.I |
| CollectPropertyMethod | System.' |
| CheckForMethodWithSameDeclaringType | System.I |
| GetClassInfo_DataRowIndex | System. |

| Param Name |
|---|
| nClassID |
| drData |

DrawableClass
DrawableClass
AssociationDrawer
JClassView

remove_SizeChanged
RegenerateControl
RetrieveClassInfo
RetrieveClassInfo
GetClassInfo_DataRowIndex

value
strFilePath
strFilePath
drData
nClassID

Figure 6: Detecting the classes, methods and parameters

The respective count i.e., number of Namespaces, Class names, Methods, Parameters are also depicted in the summery information text box.
This module is developed using the ‘Grid View control’ in the Visual Studio in with c# language.
These concepts refer directly to the ontology definition where as ontology is an “Explicit Specification of a Conceptualization”. However, the concepts that are in the application are explicitly specified over here without referring or going again to the implementation phase (coding) in the software development process.

## Ontology Type 1:

Figure 7 is a window which has the menu options such as the ‘File’, ‘Settings’, this in turn contains the ‘open file’, ‘draw diagram’, ‘select the root node’, ‘exit’. This developed using the ‘Tree View Control’, shows all methods, classes, parameters in a hierarchical representation. It depicts the hierarchical representation in a pictorial enactment of the concepts. It also shows the relationship, mainly the inheritance between the classes which reside in the same. The concepts in the class are disclosed and are shown when they are desired by the user to view the whole concepts.
The same diagram can be saved as JPEG, GIF, BMP, TIFF as desired by the user for reference.

The diagram fig 7 shows the clear picture of the concepts that are used to develop the project and is very easy to analyze the things; this refers to the basics of the ontology in the multisite software development process. This is aimed to canvass or dissect the conceptions of the project before it is delivered to the customer/client.

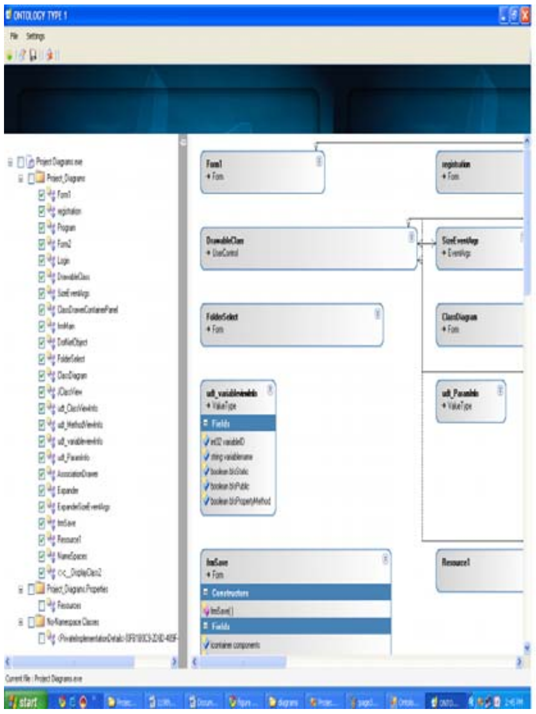


Figure 7: Pictorial contents of the concepts used in developing the actual application

However, fig 8 shows the actual ontological representations of the various concepts used in the developing the application. Their relation and the hierarchy are also shown with clear representations. The same can be viewed, saved for further enhancements and information processing of that particular software application in the same domain.

## Ontology Type 2:

The following depiction shown in the figure 9 is another type of the ontology representations, as explained above, this module also takes the .dll file or the .exe file as the input. All the contents in the application are shown in a tree format as like a super class and sub class format, aside of the window. After selecting the root node, the dependency relation like diagram is shown with respect to the namespaces, classes, and methods.

If the namespace root node is selected, the corresponding classes contained in it are depicted.

Figure 8: Class diagram with the relations among the classes

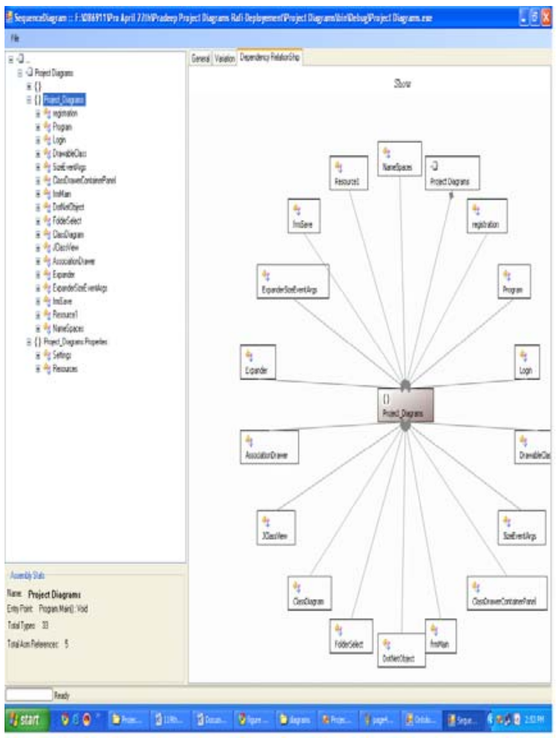


Figure 9: Class diagram with the relations among the classes

When a Class is selected as root node, then the appropriate methods contained in it are represented as a dependency diagram.

Thus the ontologies concept in the software engineering domain can be illustrated. However, this application can act as an efficient software engineering ontology tool for common knowledge sharing especially in the multisite software development.

## 5. Conclusions

In this paper, we have analyzed the characteristics of software engineering ontology. The alternative formalism have been defined i.e., graphical notations of modeling software engineering ontology. The modeling notations are used to design software engineering ontology. When the knowledge of the software engineering domain is represented in a declarative formalism, the set of software engineering concepts, their relations, and their constraints are reflected in the representation that represents knowledge. Thus, the software engineering ontology can be defined by using a set of software engineering representational terms. The software engineering ontology is organized by concepts, not words. This is in order to recognize and avoid potential logical ambiguities. A New different software engineering ontology has been developed for communication purposes. A case study with the practical implementations were implemented and deployed.

## References

[1] T.R. Gruber, "A Translation Approach to Portable Ontology Specification," Knowledge Acquisition, 1993.

[2] T.R. Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing," Proc. Int'l Workshop Formal Ontology in Conceptual Analysis and Knowledge Representation,
1993

[3] Wikipedia, "Ontology (Computer Science) from Wikipedia, the Free Encyclopedia,"
http://en.wikipedia.org/wiki/Ontology_
%28computer_science%29, June 2006.

[4] M.R. Genesereth, "Knowledge Interchange Format—Draft Proposed American National Standard," http://logic.stanford.edu/kif/dpans.html, 1998.

[5] R.J. Brachman and J.G. Schmolze, "An Overview of the KL-ONE Knowledge Representation System," Cognitive Science, pp. 171-216, 1985.

[6] D. Duric, "MDA-Based Ontology Infrastructure," Computer Science and Information Systems, vol. 1, no. 1, 2004.

[7] P. Kogut et al., "UML for Ontology Development," The Knowledge Eng. Rev., vol. 17, no. 1, pp. 61-64, 2002.

[8] J. Evermann, "A UML and OWL Description of Bunge's Upper-Level Ontology Model," Software and Systems Modeling, vol. 8, no. 2, pp. 235-249, Apr. 2009.

**P.Niranjan Reddy** received B.E. (Computer Technology) from Nagpur University in 1992 and M.Tech (Computer Science and Engineering) from NIT ,Warangal in 2001. He has been working as a faculty member in the department of CSE of KITS, Warangal, Since 1996. Presently he is heading dept of CSE. He also a research scholar pursuing his research in CSE in K.U., Warangal. He authored two text books, Theory of computation and Computer Graphics in the field of Computer Science. He published 5 papers in international journals and 6 papers international conferences. He is member of the **ISTE** and **CSI**.

**Pradeep Kumar**, born in India 1985, obtained his M.Tech in Software Engineering in 2010 from Kakatiya Institute of Technology and Sciences, Warangal. He received his B.Tech degree in Electronics and Computer Engineering in 2007 from Narayana Engineering Collage, Nellore. He was one of the toppers in his university in M.Tech and is currently working as an Assistant Professor at Narayana Engineering Collage, Nellore. His research interests include Software Engineering Ontology, Knowledge Sharing, Knowledge Management, and Genetic Algorithms.