A Survey on Ontology-Based Visualization Techniques towards Program Comprehension Application

Rozita Kadar[1*], Jamal Othman[2], Naemah Abdul Wahab[3]
[1,2,3] Department of Computer and Mathematical Sciences, Universiti Teknologi MARA Cawangan Pulau Pinang, Malaysia

*Corresponding author: *rozita231@ppinang.uitm.edu.my*
Received Date: 3 April 2018
Accepted Date: 15 August 2018

ABSTRACT

A key challenge to software maintainers during performing the software maintenance is to comprehend a source code. Source code plays an important role in aiding software maintainers to get conversant with a software system since some system documentations are often unavailable or outdated. Although there are many researches that have discussed the different strategies and techniques to overcome the program comprehension problem, there exists only a shallow knowledge about the challenges software maintainers face when trying to understand a software system through reading source code. In order to make source code more comprehensible, it needs to make an enhancement regarding to the source code presentation by transforming it into a different view. From the literature review, we found that there are lacks of study that consider the use of knowledge representation to represent source code. Hence, the aim of this work is to review the existing techniques for source code representation based on knowledge representation that could help developers to reduce the amount of their effort needed to understand a source code.

Keywords: *Program Comprehension, Knowledge Based, Ontology*

INTRODUCTION

Program comprehension is “the process of taking source code and understanding it” (Deimel & Naveda, 1990) or the process of using the existing knowledge to acquire new knowledge (Aljunid, Zin, & Shukur, 2012). The study of program comprehension can be explained as the process of taking place in the software engineer's mind when they understand the program (Feigenspan & Siegmund, 2012). This section discusses the issues arise in program comprehension followed by the existing approaches and models on program comprehension.

The activity in reviewing and understanding a source code is not the same as reviewing ordinary documents. Many problems in program comprehension arise due to the use of textual representation as the primary source of information. A program can be understood if users manage to comprehend a program, including its syntax and semantics. Although many methods and tools are proposed to represent the source code, experience have shown that the textual presentation is the superior (Krinke, 2004).

Current software systems are difficult to comprehend because of the size and complexity of the program leads to difficulties in understanding and maintaining the system. Most of the problems of understanding source code are the tricky code, different programming style, poor naming conversion, program representation, insufficient comments, and many more. Another issue is on the human activities where, software maintainers need to be able to understand the developers' code. Program comprehension activity may takes up to 50% the total lifetime of the software maintenance phase (Normantas & Vasilecas, 2013;

Roongruangsuwan & Daengdej, 2010). In addition, sometimes the source code documentation was never written, out-of-date or is lost. The implication of this is that the software maintainers have to give more effort in the maintenance phase.

Despite many researchers had worked on finding the different strategies and techniques to improve program comprehension, there is still room to discuss at great length on how to overcome the problems in program comprehension. The next section discuss about program comprehension in the cognitive point of view.

THE ONTOLOGY-BASED REPRESENTATION FOR KNOWLEDGE MANAGEMENT

At present, ontology is use in many fields to represent knowledge and provides a formal way to define the concept of certain domain. In software engineering, ontology is defined as a conceptual model that represents domain knowledge of software application. It is in a form of a set of concepts within the domain, the properties of the concepts and interrelations between those concepts (Dubielewicz, Hnatkowska, Huzar, & Tuzinkiewicz, 2015), which is considered as among the techniques that are able to support the understanding of a program (Wilson, 2010).

The Roles of Ontology as Knowledge-Based Representation

In philosophy, ontology is the study of the nature of being, the basic categories of being and their relations. An ontology is a powerful way of representing knowledge for multiple purposes and provides a framework for conceptualization and knowledge modeling in a multitude of area (Eriksson, 2007). Ontologies are used to share a common understanding of the structure of information and examine domain knowledge. Ontology defines the basic terms and their relationships comprising the vocabulary of an application domain and the axioms for constraining the relationships among terms (Ganapathy & Sagayaraj, 2011).

The basic idea of developing ontology for software system is to provide an artifact consisting of both; code knowledge and domain knowledge, with which software maintainer can understand the features of source code. Ontology includes the concepts, relations, attributes and instances that functioning to describe the specific domain of concern. Concepts refer to a category and also called a class. Series of concepts represent the topics or characters in domain ontology and the relations in ontology show the connection between concepts. The attribute uses to describe the association between concepts when considering a specific concept. Instances in ontology are the values of the attributes of the class that describes the necessary properties. The instance will inherit all the attributes or relationship of their class.

Application of Ontology in Program Comprehension

Ontology becomes popular in several fields of information technologies like software engineering. Many works analyze the role of ontologies in software engineering processes and shows that ontologies can be a basis for domain analysis and simulation in software development. A work that supports the software development process based on ontologies are proposed by Dubielewicz et al. (Dubielewicz et al., 2015). The work applies the domain ontologies to conceptual model development. It is believe that ontologies are an important source of knowledge in the conceptualization phase and propose the ontology life cycle as the background for software development.

Ontology fragment is one of the approaches that can be applied to improve program comprehension (Wilson, 2010). Petrenko et al. (Petrenko, Rajlich, & Vanciu, 2008) introduced the use of ontology fragment by combining with a tool named *Grep*. The work uses the ontology fragment to store partial domain knowledge about features. The objective of the work is to increase the effectiveness of the feature location. The work developed a tool called Protégé used to support the management of the ontology fragment as well as assists developers to formulate queries and guide them in investigation of source code. Another work that focuses on formulating queries based on ontology fragment is by Wilson (2010). The author enhanced Petrenko et al.'s (Petrenko et al., 2008) approach by introducing more systematic approach to represent partial knowledge about system. The approach provides a query as an input and allows developers to formulate a query based on the terms that are present in the ontology fragment. The result of the evaluation shows that only a small and partial knowledge about the system is sufficient for successfully locating a concept in the code.

Abebe and Tonella (Abebe & Tonella, 2010) created ontology in their work to capture the concepts and the relations from the source code. The work applied natural language process (NLP) techniques to extract concepts from source code. Identifiers from program elements are extracted and candidate sentences that use those identifiers are formed. Some of the sentences that do not follow certain rules are eliminated, and the remaining sentences are used as an input for creating ontology. The aim of the study is to allow developers to formulate more precise queries and to reduce the search space. The preliminary evaluation shows that enhancing the queries using information from the ontology increases the precision of concept location. The main difference between these approaches with the previous works is the approach is capable of generating the ontology automatically.

Although many researchers have suggested techniques and tools to implement knowledge management through this technique but researchers still have yet to discuss at great length on how to help the developers to program comprehension.

ONTOLOGY-BASED VISUALIZATION TECHNIQUES

This subsection discusses on the techniques of ontology visualization that are grouped in the following categories, representing their visualization type: *Indented list, Node–link and tree, Zoomable Visualization, Space-filling, Focus and context*; and *Information landscapes*. The categories is based on the work of (Katifori, Halatsis, Lepouras, Vassilakis, & Giannopoulou, 2007). Details of the methods are discusses below. At the end of this subsection, we provide the summary of the ontology visualization methods as shows in Table 1 that compares the strengths and weakness of the methods.

Indented List

The *Indented List* method is used to represent elements in the form of a tree-like view (as shown in Figure 1), often used to describe a visualization system, for example to illustrate the file system explorer-tree view. The main advantage of the indented list visualization is its simplicity and familiarity of implementation. The method provides a clear view to the hierarchy of concepts in the ontology. It is able to represent the taxonomy of the ontology, which represent clearly the inheritance is-a relationship through the list paradigm.

Furthermore, the arrangement of the subclass and superclass as the top-down layout of a tree browser with the subclasses appeared under the superclass and indented to the right allows for a systematic exploration of the whole ontology. User can navigate in the hierarchy of classes and expand or retract branches to view the element in the list. Its retraction and expansion of nodes are useful features specific

for focusing on explicit parts of the hierarchy, especially for large hierarchies. Although the method may reflect a clearer hierarchy in a form of tree and can display the inheritance relation but it has a few limitation where the method is not suitable to display in the form of graphs as well as it cannot support the representative role of the relationship.

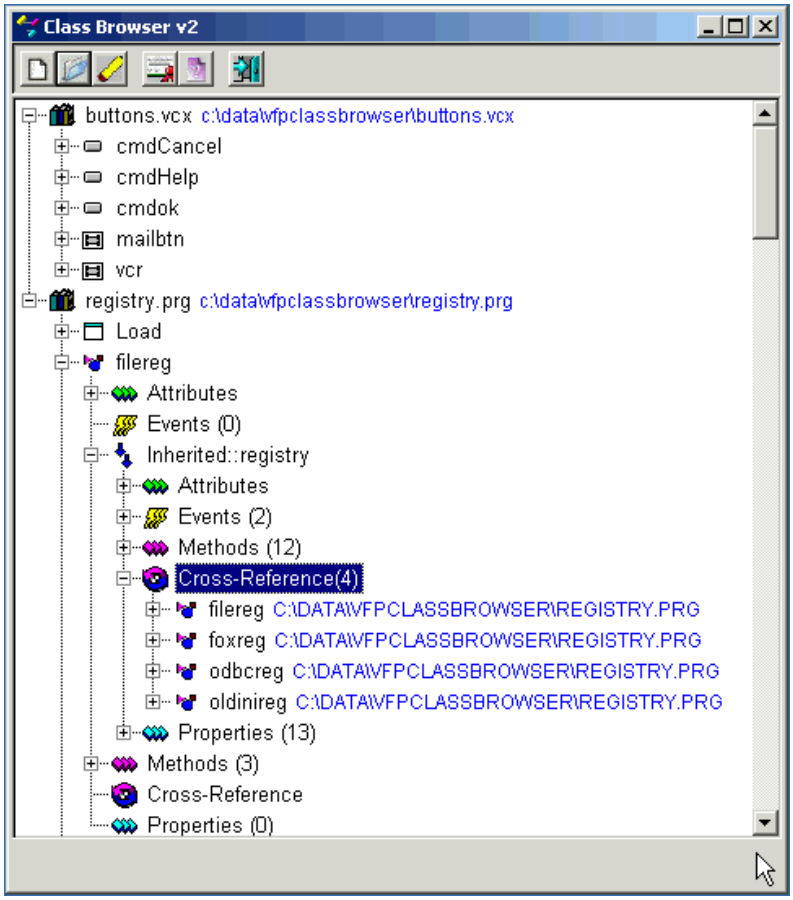

Figure 1:An example of Indented List method

Node-link and Tree

The *Node-Link and Tree* method represent the ontology as a set of interconnected nodes (see Figure 2). It provides an intuitive overview of hierarchy for hierarchy and connections representation. As nodes are displayed in a top down (or left to right) positioning, a good overview of hierarchical structures is offered, as different levels and features such as hierarchy depth or width are easily distinguishable. For an ontology presentation, it has the capability of each class to present, apart from the name, its properties and inheritance and also relations by expanding or retracting nodes for more or less details. To sum up, tree-like node link diagrams seem to be effective for representing an overview of the hierarchy. The limitation of the method is it may produce cluttered displays when more than a couple of hundred elements have to be visualized.

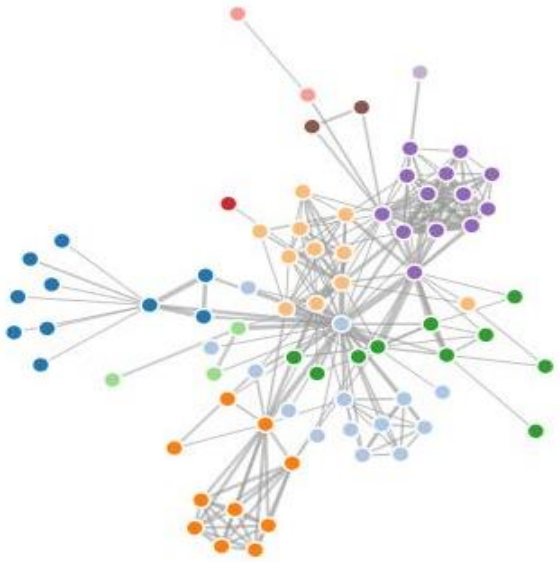

Figure 2: An example of Node-Link and Tree method

Zoomable Visualization

Figure 3 depicted an example of the *Zoomable Visualization* method. It present the nodes in the lower levels of the hierarchy nested where it appears as nested node, the child with smaller size node placed inside their parents. Nested Node is also used for instance-of relationships, thus a class node may contains both its subclasses and instances. The user may *zoom-in* into nodes for enlarging and viewing lower level nodes. These methods seem to be successful for browsing to locate specific nodes and comprehensive view of parts of the hierarchy. The lack of this method is it may result in overlapping labels and able to display only one hierarchical level at a time. Therefore, it is good for tasks that need only small parts of hierarchy to be displayed.

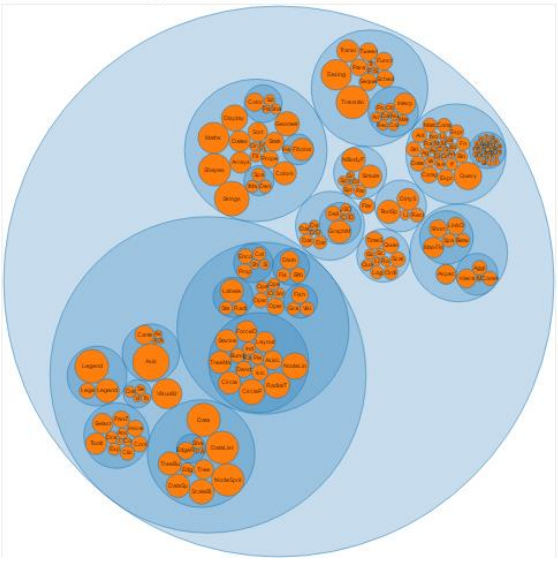

Figure 3: An example of Zoomable Visualizations method

Space Filling

Space filing method has been successful at visualizing trees that have property values at the leaf (instance) node level, which is the case in ontological structures. The reason for this is that these techniques allow colour and size coding of properties at instance level. They are effective when the user concern mostly about leaf nodes and their properties but does not need to focus on the topology of the tree.

The method is implemented into ontology visualization by dividing the ontology class in a form of area (as depicted in Figure 4). The subclass will be formed by dividing the class into subarea where this part rely on the number of subclasses of the class. Each area act as parent (superclass) node and the children (subclass) is subdivision of the node. The number of subareas indicates the number of children belonging to the parent node. Each area can be distinguished by the use of different colour and the size of the area is dependent on the properties of a class. This features used to assist users in determining the complexity of a class. The lack of this method is that it provides a limited space to place the internal nodes as well as it create difficulties to reconstruct the hierarchical of the ontology.

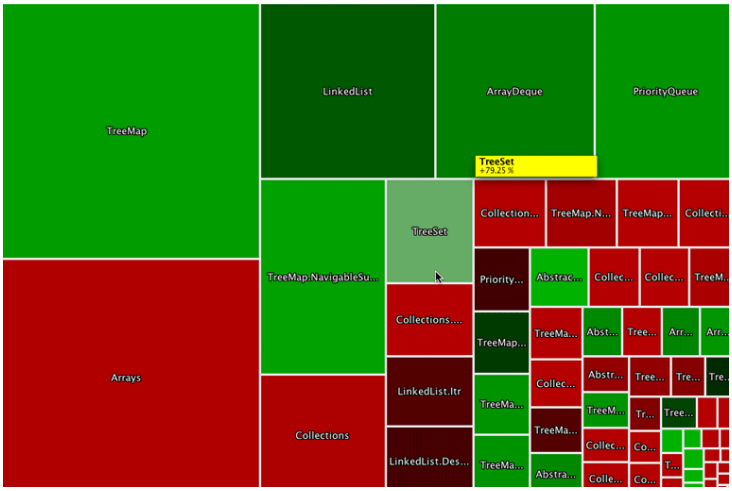

Figure 4: An example of Space Filling method

Context and Focus

The *Context and Focus* method present the node on the focus at the centre and the connected nodes around it, usually reduced in size. The techniques seem to be very effective at providing global overviews and displaying many nodes at once (as shown in Figure 5). They can be used for focusing on certain nodes and viewing their related nodes, and for quick browsing of the ontology to locate specific classes or instances. According to this technique, nodes (classes) repel one another, whereas the edges (links) attract them, thus nodes that are semantically similar are placed closed to one another.

This method is effective to provide global overviews, to focus on specific nodes, quick browsing and locating of specific nodes. It performs well and easy to display for huge ontologies. The method has a limitation where it is not suited for tasks on hierarchy as it does not offer a very obvious representation of the hierarchy structure. The user has to see the link label in order to distinguish parent from child nodes. Also, if the role relations are visible, the display seems to clutter even for ontology of a few hundred nodes. Label clutter seems to be a problem and the constant redrawing of the graph does not help the creation of a mental model of the ontology.

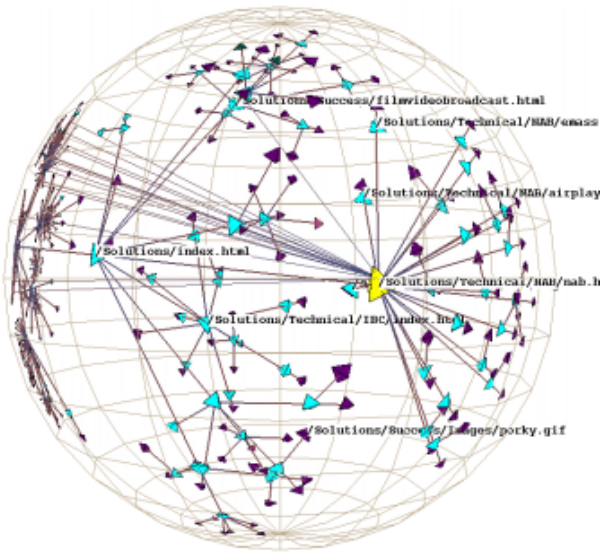

Figure 5: An example of Context and Focus method

Information Landscapes

Information landscapes (as illustrated in Figure 6) attempt to present hierarchies using a landscape metaphor. This method is useful in providing an extra dimension where node properties can be coded and relation links are presented. The large nodes can act as landmarks, so the user easily knows which part of the hierarchy should be focused on. The user's view is focused on the selected node and its subnodes but for a good structural overview, a separate overview window is needed. The limitation of this method is that the data set which has many children node will create a wide layout that cannot be seen all at once. Other than that, text labels also tend to overlap or occlude other objects. To sum up, it is not yet very clear if the information landscapes are useful in the context of ontologies. Information Landscapes could probably be effective for hierarchy overview related tasks, if coupled with appropriate searching and filtering tools as well as using intuitive, simple and effective navigation mechanisms.

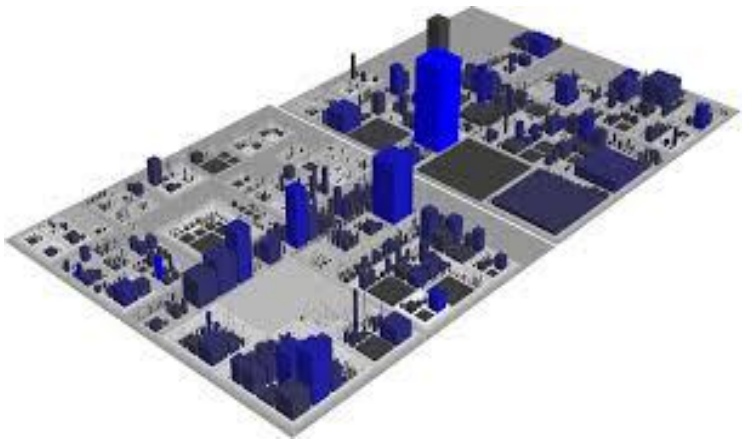

Figure 6: An example of Information Landscapes method

Table 1 Summary of Ontology Visualization Techniques

Method	• Description	Strengths	Weaknesses
Indented List	• Tree-like view. • Offered by most visualization systems.	• Simple and familiar representation. • Easy and quick-browsing. • Good overview through clear view of hierarchy.	• Ontology is graph-like but indented is tree-like. • No role relations or multiple inheritances. • Siblings are not easily seen.
Node-link and Tree	• Set of interconnected nodes. • Top-down or left to right layout. • Expand or retract nodes for more or less details	• Good overview of hierarchy and intuitive.	• Inefficient use of screen space. • Root side of tree empty. • Opposite site overfilled.
Zoomable Visualizations	• Zoom-in into nodes for enlarging, viewing lower level nodes. • Child nodes inside parents with smaller size.	• Useful for locating specific nodes. • Comprehensive view on parts of the hierarchy	• Overlapping labels. • No overview of the whole hierarchy. • Only one hierarchical level at a time.
Space Filling	• Divides whole screen among nodes • Size of subarea depends on the property of node. • Child nodes subdivide area of parent node.	• Mapping of properties to different color and size at leaf node level. • Good for tasks on leaf nodes.	• Minimum space for internal nodes. • Difficult to reconstruct the hierarchy.
Context and Focus	• Graph of nodes in a distorted manner. • Biggest node is the one in focus, others presented around it.	• Easy movement of nodes. • Quick browsing and locating of specific nodes. • Huge ontologies are easy to display.	• Confusing because of dynamic positions of nodes. • Not suited for tasks on hierarchy.
Information Landscapes	• Nodes arranged in landscapes as 3D objects. • Different color and size depending on property.	• Nodes have unique look that make it easier to recognize nodes. • Very intuitive.	• Difficulties in displaying whole tree bad overview of whole ontology.

CONCLUSION

Program comprehension is one of the problems faced by developers during the software maintenance phase. This article discusses the importance of ontology-based technique that able to overcome the problems in program comprehension. In addition, this article also revealed on the significance and advantages of ontology in representing knowledge information as well as exploring the benefit of ontology-based technique to represent knowledge towards program comprehension.

REFERENCES

Abebe, S. L., & Tonella, P. (2010). Natural language parsing of program element names for concept extraction. In *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on* (pp. 156–159). IEEE. http://doi.org/10.1109/ICPC.2010.29

Aljunid, S. A., Zin, A. M., & Shukur, Z. (2012). A Study on the Program Comprehension and Debugging Processes of Novice Programmers. *Journal of Software Engineering*, *6*(1).

Deimel, L. E., & Naveda, J. F. (1990). *Reading computer programs: Instructor's guide to exercises*. DTIC Document.

Dubielewicz, I., Hnatkowska, B., Huzar, Z., & Tuzinkiewicz, L. (2015). Domain Modeling in the Context of Ontology. *Foundations of Computing and Decision Sciences*, *40*(1), 3–15. http://doi.org/10.1515/fcds-2015-0001

Eriksson, H. (2007). The semantic-document approach to combining documents and ontologies. *International Journal of Human-Computer Studies*, *65*(7), 624–639. http://doi.org/10.1016/j.ijhcs.2007.03.008

Feigenspan, J., & Siegmund, N. (2012). Supporting comprehension experiments with human subjects. *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, (6), 244–246. http://doi.org/10.1109/ICPC.2012.6240494

Ganapathy, G., & Sagayaraj, S. (2011). To Generate the Ontology from Java Source Code, *2*(2), 111–116.

Greevy, O., & Zaidman, A. (2005). Workshop on Program COmprehension through Dynamic Analysis (PCODA ' 05), *28*(2), 2005.

Katifori, A., Halatsis, C., Lepouras, G., Vassilakis, C., & Giannopoulou, E. (2007). Ontology visualization methods—a survey. *ACM Computing Surveys (CSUR)*, *39*(4), 10.

Krinke, J. (2004). Visualization of program dependence and slices. *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, 168–177. http://doi.org/10.1109/ICSM.2004.1357801

Normantas, K., & Vasilecas, O. (2013). A Systematic Review of Methods for Business Knowledge Extraction from Existing Software Systems, *1*(1), 29–51.

Petrenko, M., Rajlich, V., & Vanciu, R. (2008). Partial domain comprehension in software evolution and maintenance. *IEEE International Conference on Program Comprehension*, 13–22. http://doi.org/10.1109/ICPC.2008.14

Roongruangsuwan, S., & Daengdej, J. (2010). A test case prioritization method with practical weight factors. *J. Software Eng*, *4*, 193–214.

Wilson, L. A. (2010). Using ontology fragments in concept location. *Software Maintenance (ICSM), 2010 IEEE International Conference on*, 1–2. http://doi.org/10.1109/ICSM.2010.5609555

Xu, S. (2005). A cognitive model for program comprehension. *Third ACIS Int'l Conference on Software Engineering Research, Management and Applications (SERA'05)*, 392–398. http://doi.org/10.1109/SERA.2005.2