Visualizing Object Oriented Software Using Virtual worlds

Satish.C.J Raghuveera.T

Department of Computer Science and Engineering,
College of Engineering, Anna University
Chennai, India.

Abstract

Software maintenance is the key issue in today's world. Lot of time and money is spent on making changes to existing versions of the software. Software maintenance engineers are forced to take up the extreme task of understanding large and complex software, which were not developed by them. Poor documentation can make understandability more complex and a mind-breaking task. Hence tools that can aid the software engineers to easily understand a given code is the need of the hour. The development of such a system that eases the understandability of software through visualizations forms a major part of this work.

Keywords: Virtual Reality Modeling language, Software visualization..

1. Introduction

Software maintenance is the key issue in today's world. Lot of time and money is spent on making changes to existing versions of the software. Software maintenance engineers are forced to take up the extreme task of understanding large and complex software, which were not developed by them. Poor documentation can make understandability more complex and a mind-breaking task. Hence tools that can aid the software engineers to easily understand a given code is the need of the hour. The development of such a system that eases the understandability of software through visualizations forms a major part of this thesis.

Visualizations shall be three dimensional in nature, and viewable using some type of inter-active browser. Object orientation is now a common principle, and 3D graphics are becoming more and more common. The description "Virtual Worlds" implies 3D graphics. Virtual Worlds are more than simply 3D diagrams; Virtual worlds must be interactive and navigable, like the real world. . Static visualization is chosen for two primary reasons. Firstly, this thesis has been written from the stance of making software more easily understood. If these visualizations are successful, they should be superior as learning tools to viewing system source code or other artifacts such as Javadoc. Source code is a representation of the static system, so the reasoning for using a static system is clear in the context of making software more understandable. Secondly, a static representation is more amenable to a Virtual World, implementation, than, for example, a program trace. Dynamic representation, generated from a program trace, is useful in other contexts, such as debugging and profiling. A three dimensional approach is taken to allow more information to be available to a user, without compromising clarity. The reasoning being that a traditional two-dimensional visualization can easily become cluttered with too much information. 3D also has the advantage of providing a more immersive environment for the user to explore.

2. Background

Much has been written on the subject of Software Visualization (SV) in general. It is a broad field, with a lot of room for interpretation. Some good references, for a broad overview, are: [5], [6], and [7].

From this general approach, focus has been on two sub-areas of SV, Object-oriented. Software visualization, and three dimensional software visualization. Substantial work has been done in both of these fields separately, and in combination, as in this study. Some

references for a good overview are: [8], [5], [9], and [10].

To deal with the OO approach to SV first, efforts in this area have been, broadly speaking, divided into two areas. The first being a run-time examination of systems. This type on visualization involves the generation of visualization from a program trace. The other area is static examination of OO systems. This thesis deals only with the latter type, static visualization. Static OO visualization deals with the details of program structure that can be discerned without ever actually executing the program. These must be derived from the program source.

A description of the attributes of an OO system that may be modeled is given in a later section. Work done in the field of static OO visualization is rather thinner on the ground than that on dynamic visualization. Languages such as UML (Unified Modeling Language) have symbols defined for visualizing an OO program's structure. There exist tools to generate a UML diagram, given a system. The software development tool, *Rational Rose*, allows OO systems to be designed in a graphical way, and turned into class skeletons. UML can also be output. A good overview of both UML and Rational Rose can be found in the book *Mastering UML with Rational Rose.*

For an overview of work done in the area of static OO visualization, see the aforementioned *Visualizations of Large Object Oriented Systems* [9].

Much work has been undertaken in the field of dynamic OO visualization. This type of visualization is usually done for debugging scenarios and algorithm visualization, rather than program understanding, so is not really of relevance to this work. The work focuses on tracing program execution through classes and method calls. A good reference to compare this form of visualization to static visualization is *Using Visualization to Foster Object-Oriented Program Understanding* [5]. The goal is to trace execution, not visualize program structure.

The three dimensional aspect of SV is covered in numerous papers. Some deal simply with the potential of having an added spatial dimension [8], whilst others become more involved, and discuss the advantages offered by having an immersive environment. None apparently use the 3rd dimension to its full potential, only using it as an extra dimension into which to move objects, rather than take advantage of its other possibilities. The approach here is not to use the 3rd dimension for any one variable, but to allow it to be used for any number of variables, or just to aid clarity in a model.

3.Advantages of 3D over 2D.

3.1. The average distance between entities in 3D space would be less than in an equivalent 2D space. This is due to the fact that 2D diagrams are forced to be spread out, whereas 3D has the potential to be more compact.

3.2. Links are less likely to cross (or come close) than in 2D diagrams. In a complex 2D diagram, in which entities may be linked to other entities to show relationships, it is often impossible to avoid the crossing of links. If this happens often enough diagrams can become very confusing and virtually unreadable. 3D diagrams avoid this potential problem entirely. There is no single plane that links are restricted to; links can travel in any direction. It is relatively rare for links to need to cross in a 3D diagram.

3.3. The positions of objects in space may have relevance to properties. For example, objects below object X have property P, objects behind have property K, etc. In 2D diagrams, this is limited to above, below, left and right. 3D adds depth, allowing more information density. We can now connect meaning to an entity being behind another, or in front of it.

3.4. 3D also offers more interesting options as far as actual viewing is concerned. Whereas a user can scroll around a 2D diagram, they may wish to fly around a 3D view, or walk, or simply stay put and rotate the world around them. When a model is built, the most appropriate means of navigation will have to be considered. For small diagrams, the best solution may be to have the model surround the user, and allow them to study the entire world simply by spinning it around on some axes. For a large diagram there may be tiers of information, the top level may contain different information to the second level. Therefore it is feasible that the user may walk around the levels, and fly between them

4.The System Architecture

The code that is taken for visualization common is converted in to an intermediated language called Object Oriented Definition Language. The OODL file is then scanned for the information about the code and based on the

information different types of models are developed. These models are provided with the appropriate textual labels and they are easily navigable and user friendly. In this paper a new model for representing object oriented software in virtual worlds is brought out. As there are no standard notations for representing Object oriented software in a virtual world we hope that this will act as a foundation for the future work in this area. Six different types of views are developed for a given source code and they help in conveying a better and quick information to the user. The various types of views and are explained briefly below

4.1.The Class Centric View.

This model is developed to represent the information of the object-oriented software at a very abstract level. The user gets a feel about the several number of classes available in the code by viewing this sort of a view .The size of the program is also represented here, as the number of classes indirectly represent the size of the code.

The given code is first scanned for the number of classes and then the order of the classes are taken in to account and then the corresponding model is developed. This view is basically developed to give a very high level information to the user .The user also gets familiar with the various shapes available and the several colors available for generating a particular model. This view provides the facilities for the user to change the color and the shape of the model based on their own interest and thereby will help them to get accustomed to the future models that will consist of more information.

The figures shown demonstrate the model generated for class view. The user is able to generate the model with different shapes and colors of his choice as shown in the figure.

Figure 4.1.Class view

.The size of the program is also represented here, as the number of classes indirectly represent the size of the code.

The given code is first scanned for the number of classes and then the order of the classes are taken in to account and then the corresponding model is developed. This view is basically developed to give a very high level information to the user .The user also gets familiar with the various shapes available and the several colors available for generating a particular model. This view provides the facilities for the user to change the color and the shape of the model based on their own interest and thereby will help them to get accustomed to the future models that will consist of more information.

The figures shown demonstrate the model generated for class view. The user is able to generate the model with different shapes and colors of his choice as shown in the figures.

Thus the user gets a feel about the order of the classes in the code and their names and the user also feels the flexibility in the model generated by changing the shapes and colors of the model that is generated to represent the classes

4.2.Property Centric View

A central class is visualized together with representations of the properties that compose the class. This type of model focuses on a class, and its associations with its properties. The type of model generated by this framework could be useful in two cases. Firstly, by showing the types of public properties, some information is made available to the user about the interface of the class being focused on. Secondly, the combination of private and public properties gives the user a good indication of the composition of the class, the data that it encapsulates. The layout of such a model would need to be similar to that of a method-centric model, with the focused on class central, surrounded by representations of its properties. Alternative forms of layout are, of course, possible, however. One could imagine a situation where the central class is represented by some large transparent entity, and within this are representations of properties.

The several classes that are scanned by from the code are represented to the user and the users option is got and the particular class is scanned from the code taken for visualization. The information about the public, protected and private variables available in that class are obtained and the internal representation is built.

Based on the information available the model is generated for representing the details of the properties of the class using VRML.Here the central class is shown as a Sphere with the variables linked to it in the form of small spheres as shown in the figure.

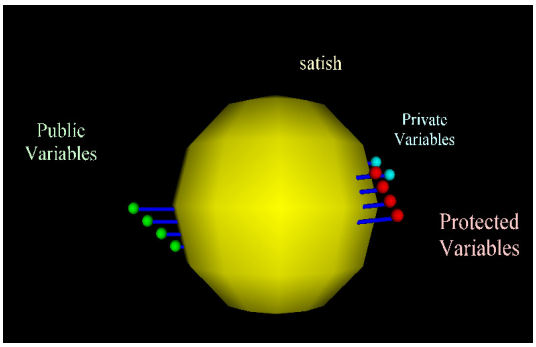

Figure 4.2.Property View

Thus the model generated represents the available public, private and protected variables in a class and thereby gives information about the interface of the class. The model is also flexible to the user, as the user can change the color and the shape of the model according to his interest.

4.3.Method Centric View

A class representation together with representations of its methods is shown. The models generated from this framework could be useful for analyzing the interface of a class, that is, its methods. There are two facets to a method-centric framework. The first is a representation of method return types, the second is a representation of argument types.
By types, we mean the classes that are returned, or used as arguments.

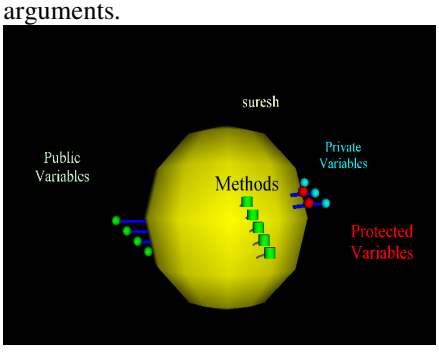

Figure 4.3 Method Centric view

There are several intuitive ways to lay out the method-centric model. One would be to have the class being focused on central to the model, surrounded by representations of its methods. When clicked upon these methods may expand to show the classes, which comprise their arguments and return types. Another model may involve having separate representations for the various classes, which are arguments and return types, and having links to these from the central class. It may be sensible in some situations (i.e. a large number of classes) A simple sketch of how a method-centric model appears is shown in Figure 4.3. The central method being visualized is central, with representations of its methods hanging off it.

4.4.Complete view

This view in short can be called as the combination of the three views mentioned above. The view depicts the details of the methods and properties of all the classes available. This view gives complete information about the inner details of the classes .As the model that is generated involves more information embedded, the user is given the option of generating the model from three different angle. This sort of user flexibility enables the user to easily navigate through the model and henceforth understand the code easily.

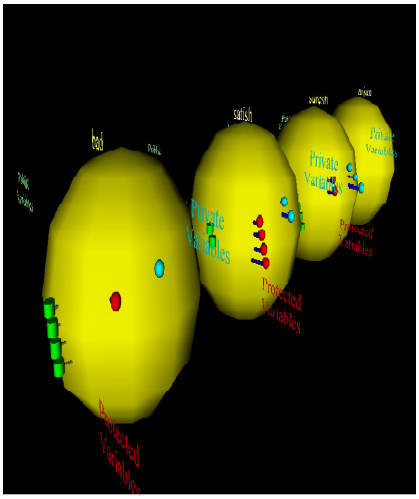

Figure 4.4 Complete View

Thus this sort of a model with different angles of view helps the user to understand and compare the information available inside the code.

4.5.Inheritance Centric view

This view allows a class to be viewed in the context of all classes from which it is derived. This framework allows the user to get an instant overview of a class and which methods and properties it has available from parent classes, without an excessive amount of navigation. Numerous variations are possible within this framework. Some may be more beneficial than others. The nature of systems being visualized may also dictate the best layout. For instance, a system with a very deep inheritance hierarchy (that is, a relatively large number of ancestors per class) may be better suited to a model in which all parent classes are shown in conglomerate form, so as less navigation is required. Those with perhaps only one or two generations behind them may, conversely, be more suited to a model showing each class individually, to emphasize which classes contribute which members. In the model shown in the figure4.5 the inheritance hierarchy is shown for any particular class that is selected by the user from the list of the available classes. The ancestors of that particular class are represented. The user is also provided with option of selecting various shapes and colors for the model.

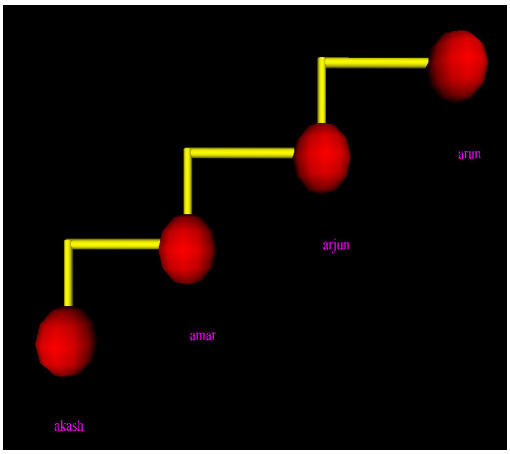

Figure 4.5.Inheritance Centric View

4.6.Metric Centric view

A lot of metrics information can be generated from a systems source code. Some of this may be useful as part of a visualization to aid a user in maintaining code. There now also exist, in addition to standard metrics for procedural languages, specialized metrics for OO systems. The types of metrics worth including are those such as \Lines of Code", and \Age of Code". These can be applied with various scopes to add meaning to visualization. For instance, they may be scoped to a branch of the inheritance hierarchy, a single class, or a method. There needs to some trade off between information quantity, and cluttering of the 3D world.
The advantage of including metrics information within visualizations is that they need not be intrusive. They can be supplementary to the main focus of a model. That is, there will be principally an inheritance hierarchy-centric model, with elements of metrics playing a supplementary role.

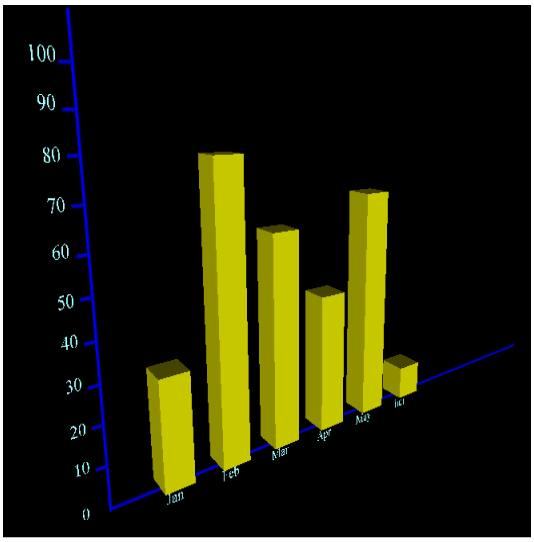

Figure 4.6.Metric centric view

size of the code taken for visualization. The view considered here represents the number of defects reported in the functionalities of the code during the several months. This will help the user to identify the more error prone areas in the code and pay more attention to it. The lines of code are also visualized for the several classes and modules in a code. This is a metric, which gives the age of the code and the number of times the code has been modified can also be visualized as part of the metric centric view

5. CONCLUSION

This project has shown that we can find new visualizations and representations for the structure of a software system. These move away from the conventional visualizations of directed graphs and expand into a more flexible and information rich environment. The new visualizations and representations make use of virtual worlds rather than the more familiar and more limited flat visualizations. User studies are needed in order to clarify which visualizations are most useful to software engineers.
Virtual and augmented reality environments encourage and support the collaborative analysis of large complex systems and their increased adoption as part of the software engineering tool set is anticipated.

6. Future Enhancements

Features to support collaborative problems solving within the VE will be of great benefit to large-scale software development. Multiple developers can enter the VE from the same or remote sites to address problems of design, maintenance, or error correction. This type of environment will also prove useful for explaining the complexities of a software system to new team members. The future version of this system will be further integrated into the software development process. The representation of the software system will be updated as each line of code is written or changed and saved.
The system can thus be extended to meet the collaborative work requirements of software maintenance engineers. As further enhancements are reported in distributed VE these sort of multi-user VE systems for maintenance can be made possible in the near future.

References

1. N. Churcher, W. Irwin, and R. Kriz. "*Visualising class cohesion with virtual worlds"*. In T. Pattison and B. Thomas, editors, Australasian Symposium on Information Visualisation, volume 24 of Conferences in Research and Practice in Information Technology, pages 89--98, Adelaide, Australia, Feb. 2003.

2. Maletic, J.I.; Leigh, J.; Marcus, A.; Dunlap, G. "*Visualizing object-oriented software in virtual Reality*" Program Comprehension, 2001. IWPC 2001. Proceedings. 9th International Workshop on , 12-13 May 2001 Pages:26 – 35

3. Young, P.; Munro, M. "*Visualising software in virtual reality"* Program Comprehension, 1998. IWPC '98. Proceedings., 6th International Workshop on , 24-26 June 1998 Pages:19 – 26.

4. Knight, C.; Munro, M. *"Virtual but visible software"* Information Visualization, 2000. Proceedings. IEEE International Conference on , 19-21 July 2000 Pages:198 - 205

5.Dean F Jerding and John T Stasko. *"Using visualization to foster object-oriented program understanding"*. Technical report, Georgia Institute of Technology, July 1994.

6. G Ruia-Catalin Roman and Kenneth C Cox. *"Program visualization: The art of mapping programs to pictures"*. In Proceedings of the 14th International Conference on Software Engineering, May 1992.

7.Blaine A Price, Ronald M Baecker, and Ian S Small. *"A principled taxonomy of software visualization"*. Journal of Visual Languages and Computing, 4(3):211{266, September 1993.

8. Hideki Koike. *"The role of another spatial dimension in software visualization"*. *ACM Transactions on Information Systems*, 11(3): 266{286, July 1993.

9 .P Haynes, T J Menzies, and R F Cohen. "*Visualizations of large object oriented systems"*. Technical report, Monash University, 1996.

10 .John T Stasko. "*Three-dimensional computation visualization"*. Technical report, Georgia Institute of Technology, 1992.