JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

OBJECT ORIENTED SOFTWARE VISUALIZATION TOOL AND ITS COMPARISON

Ashok Kumar Behera
Associate Professor
Department of CSE
Bhilai Institute of Technology, Durg (CG)
ashokctc15@gmail.com

Himabindu Maringanti
Professor
Department of Comp. Applications
North Orissa University, Baripada
profhbnou2012@gmail.com

Abstract

Software Visualization is the visualization of computer programs or algorithms, which is very crucial and costly process. The visualization method can be derived into six categories as Scope, Content, Form, Method, Interaction and Effectiveness. To maintain visualized software used for industry or research, are required visualization tools for enhancement of the functionality of software. Here, I am here going to discuss the way of analyzing of the available software and its documentation as new Visualization program. The effectiveness of visualization tool upon the effectiveness of other relevant tools.

Keywords: Analyze, re-engineering, reverse engineering.

1. INTRODUCTION

Visualization of Software is the visualization of computer programs or algorithms. The visualization method can be derived into six categories as Scope, Content, Form, Method, Interaction and Efficiency. These categories are related to the people as per his contribution. The original programmer produces the code or algorithm which may or may not be able to be visualized according to Scope category. In the Software Visualization software developer produces a system with certain capabilities which determine what aspects of a program can be shown, as Content. The visualizer specifies the visualization resulting in the Method category. The output of the Software Visualization system takes one kind of Form and the user has some kind of interaction with it. Finally, the whole visualization will have some degree of Effectiveness in helping the user to understand the program or algorithm.

It is difficult to reuse a framework that doesn't have any method how the outline functions. Due to lack of available documents it may give motivation you to create a structure of your own, like to the present software. Lack of documentation having no meaning to the existing system, which is difficult to enhance or rebuild the system. These are the examples which could inspire of using reverse engineering methods and tools. Changing in the requirement or change in the stages of a software structure, will prominence to reform a software package. The existing packages are composite. So, it may be costly to re-build as alike package. So many complications have been resolved through structural development method of the current type of software or it may not be mandatory to make the full system from the start.
Sometimes the documents of a source code that forgotten from the developer, who created the source code many years ago. Re-developing [9, 10] system that are presently existing for analyzing the source code and creating documents.

1.1 Re Development

The aim of re developing of the source code is to study the present source code so that the source code is more understandable for better development i.e. for better understandable and reuse to enhance according to the requirement of the industry. The evolution [9] method contains maintainable, relocation and re developing. The terms re developing or round tour production are used when developing is used along with traditional software development processes. In re development the major part of source code, that used is not modified. That is only considered for analysis. It is often called Software re development. In figure 1.1 the steps of software growth method in traditional software development are described. In re development, the steps are taken back from execution to strategic planning. The scheme of a software method is stored in a software warehouse. Diverse kinds of study can be prepared built on scheme exhausting metrics and graphical representation. Document preparation is used to express desires of a software method.

Analysis

All the program writing languages have syntax. Based upon syntax it might be possible to build a parser as per to a program writing syntax. By means of a parser [2] it is likely to collect data from the existing program. Parser tokenizes the program and keeps it in stack for further study of the token. Parser is normally used in compilers.

Software Warehouse

Software warehouse includes data about data of a program. Data about the organization of the program that may store in dictionary. This includes cases, values and dependences. Software warehouse may also contain different types of data handling and records. Warehouse gives an chance to generate ideas of a program.

Static and Run Time Study

Static study is the method of reviewing software without doing execution. Static study [10] contains syntactic exploration, type of analysis and implication, control flow study, data flow study, cutting and sharing, reachability, difficulty of measures, fixed calls and fundamental study. Run time study is the method of studying the runtime performance of a software method.

Pattern Recognition

There are frequently used ways to implement some simple design of a package. These are called scheme outlines. The developer usages a scheme designed when he is aware with it. Some-times a developer develop a source code and usages structure that are similar to acquainted scheme. Recognizing these forms in an existing package is one way to understand the code.

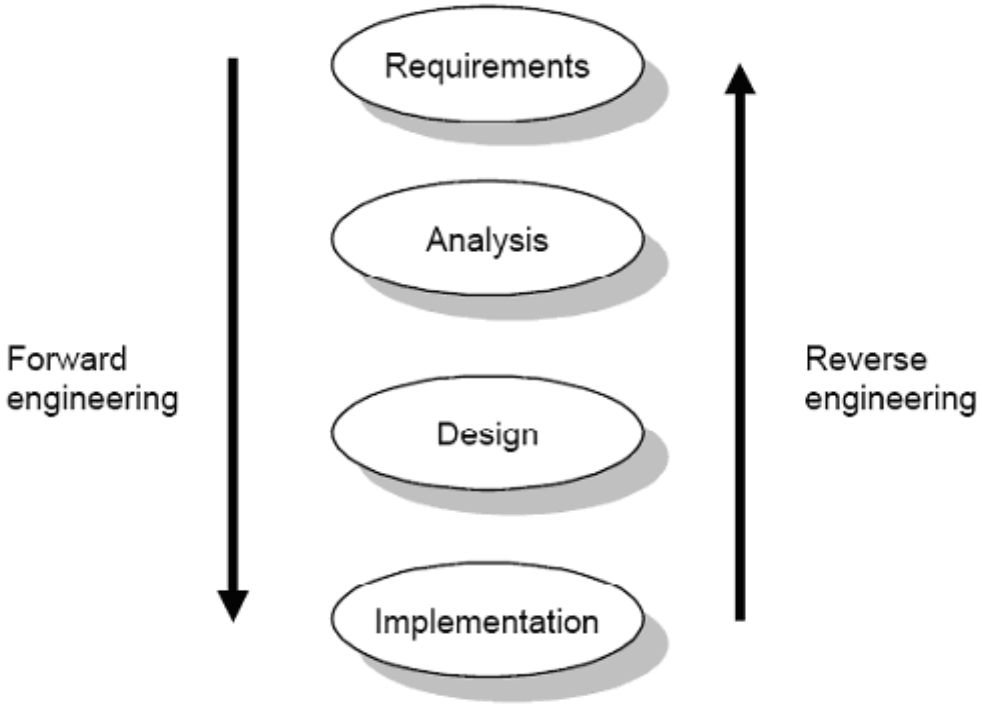

Figure 1.1: Round Trip Engineering

1.2 Software Visualization

Re-development of software gives an chance to generate diverse types of opinions on the developing of a source code or its application. Software Visualization helps to identify a package as Code Crawler [1]. In forward engineering figures are used to visualize a program. In reverse engineering, it is likely to make substitute opinions in adding to present ones as class blueprint [2]. By using these it is likely to correct present documents, and know the performance of program in better way. To maintain a program, it is probably vital that the source code maintenance engineer see some vital part of a program in another way than the program.

2. DEVELOPMENT OF SV TOOL

The SV tool offers the consecutive structure of the tool as shown in figure 2.1, which is developed in two stages. Stage1 is of modeling of metadata and 2nd stage is of representing in in visual form. Data can be communicated over association.

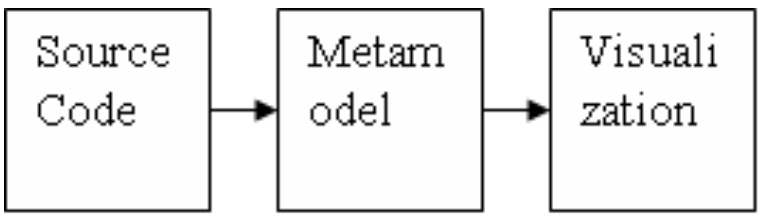

Figure: 2.1 Structure of the SV Tool

Source Code is the code written in C++ language, that have no documents available and essential for modification of code. This will help to represent the source code. This passes towards metamodeling for tokenize.

Metamodel is the core model that extract the program in the form of token and retain in a folder. All information related to the extraction of source code are retain in the given class with properties. All the data

retrieved through metamodeling are deposited in organizational way for well depiction. The main purpose of using this model is for static analysis. The file, that is stored passes for pictorial demonstration.

Visualization is the depiction of the scheme of the source code in graphical way, that may be simply appreciated by the programmer or the end-user. It contains relationship between class and its relationship with other class.

Visualization Process

The visualization Process of Software Visualization method is the user collaborating process. It suggests the actual effort of pictorial representation of OOPS, explicitly C++ code. The extraction of token from the source code, its analysis and place in proper stack is done by the visualization engine. The said methods are executed in C++.

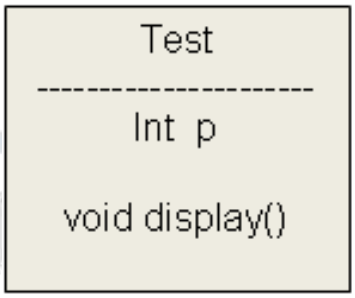

Figure.2.2 Class Diagram

The example of class and it's properties is shown in figure 2.2, where the name of class along with the properties shown in the figure 2.2 the name of class with attributes are shown. The attributes consist of the characteristics and procedures described in class. This method demonstrates all the present classes available in the code. These classes can be shown and kept in a different file which can be used in future. This method also delivers class diagram along with the methods derived using specific properties, with the characteristics & techniques stated in the class. The sample model is given in figure 2.3.

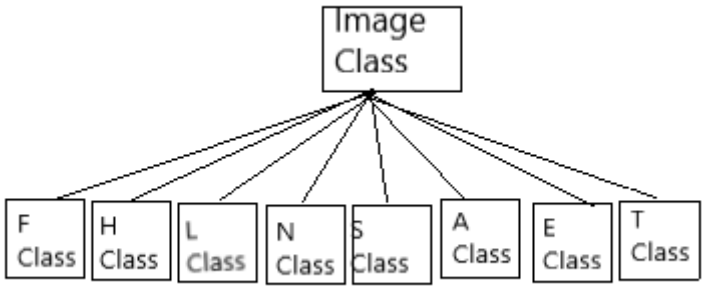

Figure.2.3 Inheritance

Software Visualization tool also delivers the number of child classes existing in the class that is NOC (Number of Children) of the stated class, along with the no of techniques available in the class. From the no of techniques WMC (Weighted Method of Class) can be calculated as shown in figure 2.4. From the number of child classes, it is easy to build the relationship between the classes, that will help better documentation of the source code.

No of Child Class : 2
Total No of Method : 1
WMC : 0.6

Figure.2.4 Class Count

3. OTHER VISUALIZATION TOOLS

3.1 SANVIZ Software Visualization

The architecture of SANVIZ is shown in Fig 3.1. The *compiler block* consists of three basic units namely the *parser*, *analyzer* and *trace[3]*. The parser block decomposes the input source code into tokens. The analyzer block conducts a comparison-based semantic analysis.

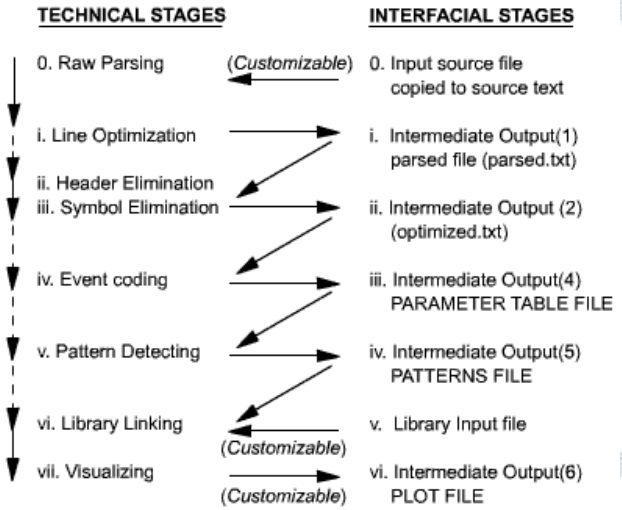

Fig 3.1 : SANVIZ Software architecture

The post-compilation information is then recorded in an event trace file, appropriately coded to signify individual object-oriented event. The *detector block* consists of *sequencer* and *matching* units. The event trace file undergoes necessary sequencing and annotation after which it is matched with various patterns available in the library. Matched patterns are recorded in pattern trace file. The *visualizer block* matches patterns entry from pattern trace file with standard graphical notations available in the library as a function of the *graphical interpreter* unit. Unified modeling language (UML) notations have been adopted as the representation standard. The *graphical actuator* unit finally displays the matched graphical notations in a fully composite window display. The technical and interfacial stages of SANVIZ is shown in Fig 3.2.

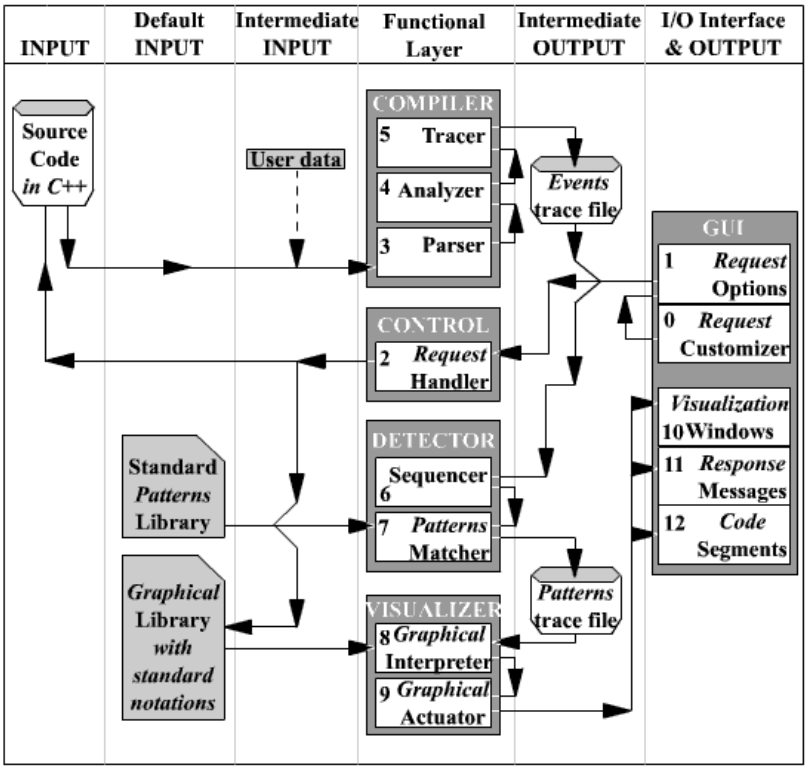

Figure 3.2 : technical and interfacial stages of SANVIZ

3.2. RIGI Software Visualization

Rigi provides a reverse engineering environment and facilitates software visualization via its editor or viewer called RigiEdit. It is the most cited and compared tool within the scope of reverse engineering and software visualization. It is quite a comprehensive prototype tool of its kind, consequently it is considered as the most representative tool among its counterparts. Although commercial tools like SNiFF+ might have a competitive edge compared to this prototype tool, they do not provide intermediate files of parsed software artifacts to researchers or practitioners. This causes the difficulties particularly to researchers who want to manipulate the intermediate files to fit their research needs. Likewise, the intermediate files generated by Rigi parsers are in ***Rigi Standard Format (RSF)*** 3-tuples that is quite simple, easy to understand and customize.

The software artifacts extracted by Rigi parser[11] is also up to the lowest unit of source codes that is a local variable. The research of Rigi tool is still active and its web sites are always updated with the latest version of Rigi to be downloaded. The research group of the tool is also quite supportive and helpful as experienced by the researcher. These reasons instigate Rigi to be selected as the reverse engineering environment in which DocLike[12] Viewer prototype tool of this research should reside and as a control tool in the experiment conducted to evaluate the proposed method and tool.

3.3. CODE CRAWLER – Software Visualization

It is not a language specific Software Visualization method developed in Smalltalk. It delivers the structural representation and relationship with other classes & methods. It performs the task beside the moose atmosphere. It gathers information from the program and after words is processed in pictorial form. It extracts classes & attributes from the program and shows them in the relation of flowchart.

Code Crawler is focused on visualizing static data about software, *i.e.*, thus working mostly at a structural level. The internal architecture of Code Crawler is the metamodel, that can be divided into four parts: the core model, the polymeric views [2] subsystem, the layout engine and the user interface and service classes. The model of Code Crawler is shown in figure 3.3.

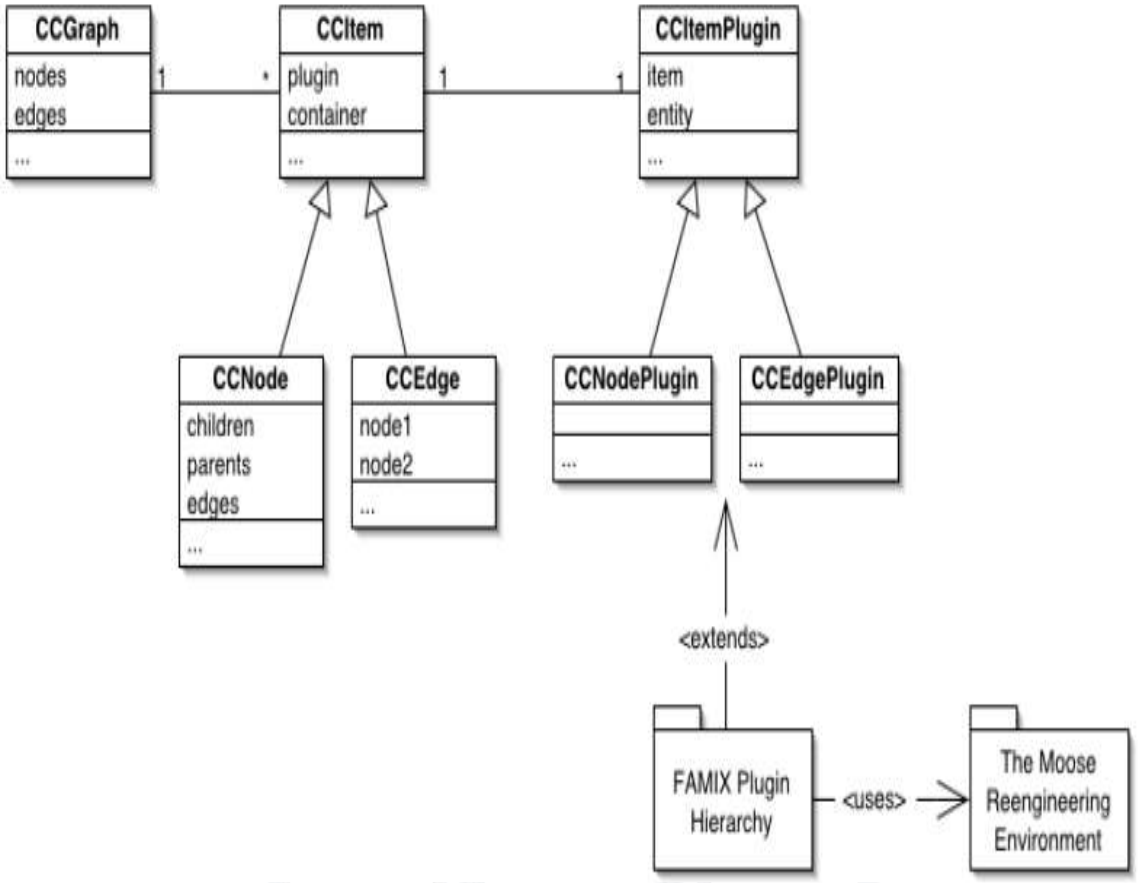

Figure 3.3: The Model of Code Crawler

4. COMPARISON

The visualization tools used so far provides flow chart, sequence diagram along with the number of classes used in the program. The Software Visualization tool designed and developed by us provides detail about the program. It provides the name of classes, its base class, member variables and member functions. It also provides all types of inheritance of classes in the form of multiple, single and multilevel inheritances. For this Software Visualization tool has different appearance than the other tools. Since it provides detail about the class along with the relation towards the other classes, it is easier to retrace the program and to find the algorithm of the existing program. This provides excellent features about the visualization of object oriented programming specifically C++, by which it is easier to extract the documentation and to redesign the code for remake of the software

5. CONCLUSION

Reverse engineering authorities us to expand comprehend and scope with the difficulty of the code. In large source code, it is mandatory to develop high-level ideas to make the code more comprehensible. It is likely to make another opinions of the source code by using reverse engineering methods. In this method it is likely to generate documents, which might not have been even produced in many cases. The data which are lost or not ever documented can be recovered. The documents which have defined previous versions of a code or consist of features that are not applied. With reverse engineering methods a developer may generate a better scheme. Later reforming & executing the novel design with portion of previous program, or later accumulation novel structures into a current method, undesirable performance may appear. The previous method, we want to implement prepares really above it was supposed to fix or the efficient task does not achieve to understand all the earlier responsibilities. The output of re-development enables to use and they can also be implement to inhabit a software warehouse.

Our software visualization tool provides the better design for object oriented programming specifically C++ program. This will enhance the performance of reengineering. This provides relation of class/object, along with its count and inheritance.

6. FURTHER SCOPE

For further study, this tool is proposed to be enhanced for other object oriented programming like JAVA along with the existing C++ program. Again it is proposed to recognize object and its relation with other objects either through inheritance or friend function.

REFERENCES

[1] Biggerstaff, T.J., Mitbander, B.G. and Webster, D.E., "Program Understanding and the Concept Assignment Problem", *Communications of the ACM*, Vol. 37, No. 5, pp.72-83, May 1994.

[2] D. Auber. Tulip : A huge graph visualisation framework. In *Graph Drawing Software – Mathematics and Visualization*, pages 105–126. Springer, 2003.

[3] D. H. R. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics (Proc. InfoVis 2006)*, 12(5):741–748, 2006.

[4] D. Auber, M. Delest, J.-P. Domenger, and S. Dulucq. Efficient drawing of rna secondary structure. *J. Graph Algorithms and Applications*, 10(2):329–351, 2006.

[5] T. Barlow and P. Neville. A comparison of 2D visualizations of hierarchies. In *Proc. IEEE InfoVis*, pages 131–139, 2001.

[6] F. Chevalier, M. Delest, and J. P. Domenger. A heuristic for the retrieval of objects in low resolution video. In *Proc. Intl. Workshop on Content-Based Multimedia Indexing (CBMI)*. IEEE Press, 2007.

[7] S. Ducasse and O. Nierstrasz. On the effectiveness of clone detection by string matching. *Intl. J. on Soft. Maint. And Evolution: Research & Practice*, 18(1):37–58, 2006.

[8] R. Koschke, R. Falke, and P. Frenzel. Clone detection using abstract syntax suffix trees. In *Proc. WCRE*, pages 253–262, 2006.

[9] L. Voinea and A. Telea. Visual data mining and analysis of software repositories. *Computers & Graphics, DOI 10.1016/j.cag.2007.01.031*, 2007.

[10] L. Voinea, A. Telea, and J. J. van Wijk. Cvsscan: Visualization of code evolution. In *Proc. ACM SoftVis*, pages 47–56, 2005

[11] Mariam Sensalire and Patrick Ogao: Visualizing object oriented software: Towards a point of reference for developing tools for industry. IEEE conference on visualizing software for understand and analyze 2007 (1-4244-0600-5) pp 26 – 29.

[12] Scholtz,J., Plaisant,C., Whiting,M., Grinstein,G.: Evaluation of Visual Analytics environment: The road to the Visual Analytics Science and Technology Challenge Evaluation Methodology. SAGE Journal. 2013.

[13] Eden,A.H., Gasparis,E., Nicholson,J., Razman, R.: Modeling and visualizing object-oriented program with Codecharts. Springer. 2013