Analysis and Design of Software Visualization Tool for the Behavior of Object Oriented Programming

Ashok Kumar Behera
Bhilai Institute of Technology
Durg (CG)

Rajesh Tiwari
Sri shankaracharya College of
Engineering & Technology, Bhilai

ABSTRACT
The software visualization tools are highly required by the industry and research centers. In past, many specific tools have been developed, whose rate of development does not match the rate of requirement of industry. In this paper, we are going to discus the analysis of the similar tools available and design & development of our Software Visualization Tool, which focuses the class diagram with attributes and methods of class of object oriented programming. We design the common algorithm for visualization of object oriented program. The aim of our tool is to simplify the design of complex code, which can be easily maintainable and usable.

Keywords: Software visualization, re-engineering, reverse engineering.

1. INTRODUCTION
Software reverse engineering is most costly activity in software development. Different tools have been proposed to understand the software system. The application of different visualization techniques [3] has the potential to provide the solution for complex system. The success of the tool depends upon, how it fulfills the need of user.

The tools are widely necessary for the industry for maintenance of the software. The task of tool is to identify the design from the coding of the system. This assists the user of software to understand better about the system.

This paper presents the software visualization support[11] to simplify the complexity of program to lower level. This tool is designed in C++. This provides the information that is required to maintain the programs written in specifically in C++. This tool provides the design of the program with respect to class and lower up to attributes along with methods. This will help for re design and maintain the project.

2. PREVIOUS WORK
In past the tools of software visualization was developed according to the need of user. The tools are Code crawler[1,7], Goose[1, 7], Jaliot, CVSgrab, Code Visual to Flowchart, etc.

Code crawler is a language independent software visualization[3,5] tool written in Smalltalk[1]. Code crawler provides the class structure[2,7] and relation of class & functions. It works along the moose reengineering environment. It collects and stores the data and later on is visualized[3]. It picks classes, functions from the code directly and design flowchart. The other tools are designed to form the class diagram, object diagram. Presently most of the tools are no use as all are requirement specific.

Code Visual to Flowchart provides the flowchart of the program. The flowchart is language dependent, as the flowchart build directly from block of code, not in generalize form.

3. MOTIVATION
As discussed earlier, the past tools provide the flowchart, sequence diagram, class diagram and its relation with other classes. Apart from that, it provides inheritance.

It is necessary to find out, which attributes, the class contain and which attributes, it inherited from the parent classes[2]. How attributes transfer from one class to other, whether it is static or dynamic. How attribute type will be identified? All these are necessary to simplify the requirement of user. From this requirement we motivate to design the tool that is low level design like attributes along with class diagram. This will help developers for better and easier redesign of software.

4. VISUALIZATION TOOL
This tool is to analyze the object oriented programming[6], taking command line argument and provides menu based solutions. Then it scans the code word by word and store in different stack like class, attribute and method. The attributes and methods are divided according to their access-specifier. Whenever the class is inherited, it matches the type of inheritance to access-specifier for inheritance properties. The reference model[4] is shown in fig(1).

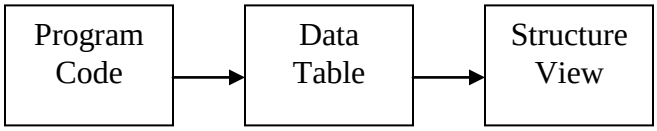

Fig 1: Reference model

Program Code: - The source code of input system. Here we are considering C++ programming as input, from which design would be extracted.

Data Table: - Retrieved data like attributes, methods, classes along with relations. Data table extracted from source code through SV-tool are stored in a file from.

Structure View: - Representation of relations. The relation is build in pictorial form, it is class diagram, inheritance and properties present in classes.

4.1 Design of the Tool

Our focus is to simplify the design of complex code, which can be easily maintainable and user-friendly. The main focus is

(a) Need :- The requirement of tool to user. It is designed for research and education purpose.
(b) User :- The users of this tool are research scholar, student and instructors of the institutes.
(c) Representation :- We represent the tool very simple and user friendly. It provides the information on menu based.

Firstly, the program is passed through analyzing tool, which provides the required data necessary for retrieval. The retrieved data is stored in a file for future enhancement. In the next step the information available in the file is represented in graphical format, which can be accessed by the user.

4.2 Algorithm of the Tool

The algorithm is designed in two parts; one is to extract class along with its attributes and methods. Second part is to design the extracted information in menu based format. Along with the algorithm we are given the complexity of it. Figure 2 shows the first part and figure 3 shows second part.

SV(file X)

1. For each char ch in X
2. if ch != ' ' or ch !='\n'
3. ch1[i] ← ch
4. i ← i+1
5. ch1[i] = '\0'
6. if ch1=class
7. then file1 ← class-name
8. While ch != end-of-class
9. file1 ← var
10. file1 ← method

Fig:2 Extraction of members.

This algorithm provides variables and methods of class. The worst case of time complexity is **O(n).** As the main loop will continue for N times, where N is no of character present in the program. The second loop will run for M times and M<N.

Inheritance (file x)

1. for each char ch in x
2. if ch = ':'
3. then f ← f+1
4. else ch1[i] ← ch
5. if b = 1
6. base ← ch1
7. else derv ← ch

Fig:3 Graphical representation

The worst case of time complexity is **O(n).** As the main loop will continue for N times, where N is no of character present in the program.

4.3 Architecture of the Tool

This tool provides the sequential architecture, which is of two stages. Stage1 is of metamodel and stage 2 is of visualization. Information will pass through bridge.

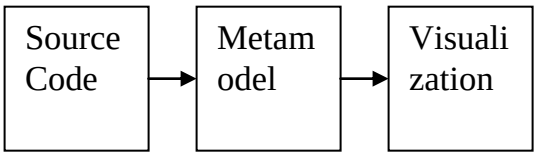

Fig.2 Architecture of the Tool

Source Code is the program written in C++, which have no documentation and required for maintenance. This required visualization of the source code. This pass to the metamodel for tokenize.

Metamodel[7] uses the algorithm shown in figure 2. This is the base model that tokenizes the source code, i.e. the C++ program and stored in a file. All information regarding the visualization of source code are stores in the specified class having attributes as variables and methods. All the information extracted through metamodel are stored in structural way for better representation. This model used for static code. The stored file is then passing for graphical representation.

Visualization is the representation of the design of the code in pictorial form, which can be easily understood by the developer or the user. It includes class diagram along with the behavior of the attributes and methods through inheritance.

4.4 Visualization Engine

The visualization engine[6,7] of SV tool is the user interactive tool itself. It provides the real work of visualization of object oriented program, specifically C++ program. The algorithms stated in this program are implemented in C++.

The sample class diagram of the program is shown in fig 3.

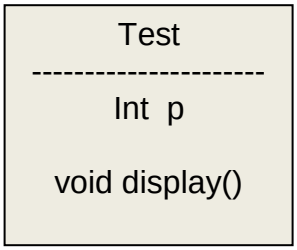

Fig.3 Class diagram

Here in figure 3 the class name along with the properties are given. The properties include the attributes and methods used in class. This tool extracts all classes available in the program. These classes will be displayed and stored in a separate file which can be utilized later. This tool also provides relation between the classes. How the properties are inherited through access-specifier, along with the attributes and methods specified in the class. It shows all types of inheritance. From

the inheritance we can extract the properties used by the classes. The sample example is given in figure 4.

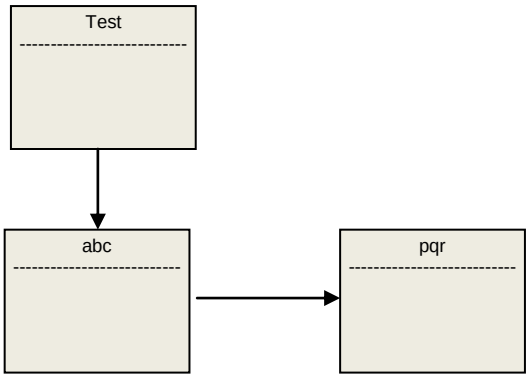

Fig.4 Inheritance of classes

5. CONCLUSION

This paper presents a visualization design of object oriented software with focus on the inheritance of properties. This provides actual design of program, flow of data and methods from one class to other.

This paper describes the concept of the tool and the techniques used in it. The target of these systems is to provide multiple object-oriented environments. This tool is already implemented in our research lab. This is mostly helpful for research students and industries used C++ programs. As per the testing concern, it can extract small and medium scale programs and provide the required output. We will further enhance this tool for large scale programs.

6. REFERENCES

[1] Michele Lanza: Code Crawler- Lessons learned in building a software visualization tool. IEEE Conf. on Software Maintenance & Reengineering, 2003. pp 409 – 418.

[2] Stephane Ducasse, Michele Lanza: The class blueprint: Visually Supporting the Understanding of Classes. IEEE Journal 2005, Vol-31, Issue 1. pp 75 – 90.

[3] Thorsten Schater, Mira Mezini: Towards more flexibility in software visualization Tools. IEEE Conf.on Visualizing Software for Understanding & Analysis, 2005. pp 1 – 6.

[4] R. Ian Bull and Margaret-Anne Storey, Jean-Marie Favre: An architecture to support model driven software visualization. IEEE 2006

[5] Jonathan I Maletic, Aridrian Marrcus, Michael L. Collard: A Task Oriented view of software visualization. IEEE Conf. on Visualizing Software for Understanding & Analysis, 2002. pp 32 – 40.

[6] Michael P. Smith and Malcolm Munro: Runtime visualization of object oriented software. IEEE Conf. on Visualizing Software for Understanding & Analysis, 2002. pp 81 – 89.

[7] Anslow C., Noble J., Marshall S., Tempero E.: Towards End-User Web Software Visualization. IEEE Symposium on Visual Languages and Human-Centric Computing. 2008. pp 256 – 257.

[8] Blaine Price, Ronald Backer, Ian Small: An introduction to Software Visualization.

[9] Timo Raitalaakso: Dynamic Visualization of C++ Programs with UML sequence diagrams.

[10] Mariam Sensalire and Patrick Ogao: Visualizing object oriented software: Towards a point of reference for developing tools for industry. IEEE 2008

[11] Craig Anslow, Stuart Marshall, James Noble, Robert Biddle: Software visualization tool for component reuse.

[12] Wim De Pauw, David Lorenz, John Vlissides, Mark Wegman: USENIX – Execution patterns in object oriented visualization:URL– http://www.usenix.org