Journal of King Saud University – Computer and Information Sciences 31 (2019) 403–414

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

The effect of 3D visualization on mainframe application maintenance: A controlled experiment

C.J. Satish *, Anand Mahendran [1]

School of Computer Science and Engineering, VIT University Vellore, Tamilnadu, India

ARTICLE INFO

Article history:
Received 8 September 2016
Revised 18 January 2017
Accepted 12 March 2017
Available online 15 March 2017

Keywords:
Software maintenance
Software engineering
3D software visualization
Software design extraction

ABSTRACT

Code written in 1960's and 80's are still being maintained by large software organizations. Such legacy systems play a significant role in supporting various data immense applications in banking, manufacturing, retail marketing, health care domains etc. The software maintenance engineer for such large systems has to undergo not trivial tasks of identifying and understanding relevant parts of system related to the developer/maintainer goals. As these systems were written before the evolution of software engineering principles, there is no standard documentation available for such systems and even best coding practices were not followed during the development of these systems. Hence, maintenance of such systems is a challenging task for young software engineers who need to spend a lot of time in comprehending the system before fixing the errors. The lack of tools that support legacy code comprehension makes the maintenance process to be time consuming and tedious. Our research is focused on understanding the impact of 3D software visualization for change impact analysis tasks on the mainframe during the maintenance phase. To conduct this research we have constructed a 3D visualization tool which generates a three dimensional call graph for a system written in COBOL using non immersive virtual reality (VRML). We have conducted a controlled experiment to test the effectiveness of the tool on subjects performing change impact analysis tasks. Our results show that subjects who used the tool performed more the tasks more accurately and faster than the group that did not use the tool.

1. Introduction

Mainframes play a significant role in today's enterprise systems because of their data processing capabilities (Van Geet and Demeyer, 2010). Studies show that 93% of Finance and Insurance data and 83% of transactions worldwide is processed using COBOL on the mainframe (Van Geet and Demeyer, 2008). Mainframes not only support thousands of concurrent users they are also very reliable and secure. They transfer huge quantities of data to other systems and also manage data in large amounts. Mainframes retain their continuity largely because they maintain strict backward compatibility for old software. The vast majority of mainframe programs are still those written in COBOL, with a significant use of PL/1, Assembler, C and C++ (ATOS, 2007). One the most concerning factors that impact the mainframe systems are its huge maintenance costs. The legacy systems were developed when software documentation was not rigorously followed and over the years of evolution the documentation that is available has become obsolete without timely updates.

This has led to the evolution of a highly complex system with inadequate information about the evolution. Software engineers face the daunting task of understanding the system by analyzing the code before making the actual changes. The documents are either out dated or missing and this leads to non availability of critical information on the architectural evolution for the engineers. The loss of information there by contributes to the increase in time with respect to impact analysis (Alaranta and Betz, 2012).

Change control attributes to almost 70% of the project life cycle costs (Hunt et al., 2008). Software maintenance is turning out to be a costly task due to poor documentation, lack of user understanding and inadequate user training (Palvia et al., 1995). Understanding the legacy code is one of the major challenges in the field of

* Corresponding author at: School of Computer Science and Engineering, VIT University, SJT 116 A07, Vellore, Tamilnadu, India.
E-mail addresses: Satish.cj@vit.ac.in, satish.cj13@gmail.com (C.J. Satish), manand@vit.ac.in (A. Mahendran)
[1] Address: School of Computer Science and Engineering, SJT 411 A15, VIT University, Vellore, Tamilnadu, India.

Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.jksuci.2017.03.003
1319-1578/

software maintenance. There is a huge need for reducing the time and effort towards maintenance of systems. Clients can be provided better services by IT firms at reasonable costs, without draining a huge portion of their budget on fixing software systems. Almost half of the time spent on fixing a system is taken up for program comprehension in software maintenance (Fjeldstad and Hamlen, 1979). Prior literature on program comprehension states that engineers spend 95% of their time trying to understand the impact of the changes made to the system and only 5% of the time is spent on support activities(Alaranta and Betz, 2012).

The time taken to comprehend a program can be reduced by following stringent documentation practices and using tools that aid maintenance engineers easily understand the architecture of a given system. Studies have found that development decisions, effective processes and tools can reduce the maintenance costs (Hunt et al., 2008). The use of tools like NDepend, JArchitect,SeeSoft,sv3D,SHriMP etc. for static code analysis can help in reduction of the costs by supporting decision management during the software maintenance phase. The comprehension of legacy system architectures can be successfully achieved by a combination of program analysis techniques and visualization tools (Zhang, 2003). Pierre Caserta and Olivier Zendra have summarized the various methods available for visualizing software (Caserta and Zendra, 2011).

Though, there has been a lot of research happening on visualization of code, very less amount of work has been done towards visualization of legacy systems using 3D space. There is a significant need for tools that support visualization of COBOL code since mainframe-savvy people have become a dying breed and typical university graduates aren't familiar with mainframes. These graduates are trained and are interested in the newer object oriented programming languages like Java and C++ instead of the older languages (such as COBOL and PL/I) (Van Geet and Demeyer, 2010). These graduates when recruited for maintenance tasks on the mainframe are left with the difficult task of learning a new platform and also the comprehension of COBOL code which has evolved over several years without any adequate documentation. Hence, maintenance of legacy systems is posing a great challenge in terms of finding the people with the right skillset and reducing the costs towards fixing these systems.

In this paper, we propose a tool for generation of a 3D call graph from given cobol source files. We have elaborated the research conducted as three parts in this paper. In the first part we have elaborated on the importance of software visualization and its role in software maintenance. In the second part we explain the architecture of our proposed tool and the algorithm behind its implementation. In the last part we detail the controlled experiment that we have conducted to analyze the impact of the tool on change impact analysis tasks during software maintenance. We conclude with our analysis of the results and future work.

2. Software visualization

Software visualization is a field in computer science that is used for displaying a software system graphically (Wang et al., 2009). The impact of UML (Unified Modeling Language) towards comprehension of object oriented systems across the globe stresses the importance of software visualization to a greater extent (Caserta and Zendra, 2011).

Software visualization is also used in the fields of reverse engineering, reengineering and software maintenance to understand the complexity of systems. Effective graphical representations can improve user's perception capabilities over textual representations (Card et al., 1998)

Visualization is mostly achieved by mapping the contents of the source code to a visual model that can be either in 2D or 3D. The visual models can be in the form of graphs with nodes and edges representing function calls in the form of a tree structure. They can be close to real world metaphors like cities, solar system etc. (Caserta and Zendra, 2011).

The idea behind such models is to give the end user an immediate representation of the system from the required perspective and there by acquire knowledge within a reduced time frame.

2.1. Visualization in software maintenance

Software visualization is widely applied in software maintenance for program comprehension, metrics and change history visualization (Caserta and Zendra, 2011). Software Visual Models can help engineers easily understand the architecture of a given system and the impact of changes requested on the system.

Software visualization plays a crucial role with respect to reverse engineering and re engineering large amounts of complex data in software maintenance(Koschke, 2002). The aim of software visualization is not to create impressive images but models that can aid software engineers in program comprehension and reduction of impact analysis time (Maletic et al., 2001a,b). Engineers can get an perception of how software is structured and they can easily understand the program logic and will be able to explain and communicate the development (Diehl, 2007). A visual representation of information will help users to comprehend large amounts of data (Colin, 2012)

Though many visualization techniques were proposed by reseachers, the induction of such techniques in the industry has been very slow. The reason for that may be the attention given to software maintenance is relatively low when compared to other phases in software development (Storey et al., 2008)

2.1.1. Significance of call graph visualization for legacy systems

Call graphs reveal the functional dependencies between programs. The number of incoming calls (NIC) to a program measure the number of other programs referencing the program. The number of out going calls(NOC) from a program measure the number of calls made within the program to any other program. These measures are very important for estimating the impact of the change in software maintenance. A program with high measure of incoming and outgoing calls will be taking more time to fix as the change made to that program may impact other programs.

Van Geet and Demeyer (2008) constructed a tool for visualizing functional dependencies in a COBOL program using a graph. This knowledge was used for understanding the complexity of COBOL programs while migrating towards a service oriented architecture.

Khadka et al. (2013a)) identifies visualizing source code and their dependencies to be one of the techniques in legacy system understanding. Khadka and his team has also used a call dependency diagram to understand the computationally intensive cobol modules during the migration process of a legacy application to Service Oriented Architecture (Khadka et al., 2013b)

Control flow information in a system is visualized using call graphs. Koschke has identified call graphs to be one the most visualized artifact in software maintenance, reverse engineering and re-engineering through a research survey. The graphs in software maintenance are usually large and tackling large graphs is a research problem. The survey also reveals that many of the participants have requested animation and 3D visualization for maintenance, reverse engineering and re-engineering (Koschke, 2002).

Lei and his team has constructed a routine interaction diagram to identify routines that always interact together to perform a system functionality. Such a diagram helped the team towards decomposing a legacy system for migration (Lei et al., 2005). Call graph visualization thus plays an important role in understanding legacy systems during the maintenance and migration process.

2.2. 2D versus 3D visualization

2D Graphical representation is a traditional and simple way for visualization. It is presented as a simple xy plot. Treemaps, Seesoft, Shrimp views UML are some examples of 2D visualization techniques (Caserta and Zendra, 2011). 2D visualization does not require higher hardware requirements as that of a 3D visualization. The learning curve for users in 2D Visualization is lesser than that of a 3D Visualization as 3D Visualization has much complex interfaces and interactions. Navigation in the 2D space is much simpler than 3D as it is difficult for the users to navigate without being lost in a 3D space.2D visualizations are cheaper to build than 3D visualizations which are complex, complicated and involve high computation (Xiaohua, 2003). The important drawback of 2D visualization is that for large systems the picture gets highly complex with lot of nodes and interconnections. Such a cluttered image may be difficult and very confusing for an engineer to understand (Caserta and Zendra, 2011).

In the last decade with the advancement of technology a lot of research has been done towards the utilization of 3D models in software visualization. 3D technique is used to give a real world representation of the software artifacts. The information density is greater in 3D presentations when compared to the 2D ones (Robertson et al., 1993). In the case of a 3D visualization an extra dimension is given in terms of depth which improves the space usage. It gives greater information density and it facilitates perception of the human visual system. It has the capability of user interactions like (zoom, rotate, displays in any angle). Thus, 3D visualization makes it easier to view large applications where in the intersection of edges of the nodes will be lesser when compared to that of a 2D version of the model

2.3. Advantages of 3D visualization

Metaphors that are closer to the real world can be created using 3D visualizations (Alfredo et al., 2009). With the advancement of hardware and software the trend is to experiment new 3D visualizations. The interactions with a 3D model can performed with little effort (Caserta and Zendra, 2011). The human visual systems perception is facilitated by the 3D presentations (Diehl, 2007). The density of information provided in a 3D representation is higher than a 2D representations (Alfredo et al., 2009). It is basically a composition of multiple 2D views into a single 3D view. The 3D tools focus on different levels of abstractions and this helps engineers to understand how the software works from different perspectives (Alfredo et al., 2009).

2.4. VRML in 3D visualization

3D Visualization of COBOL code has been achieved using VRML (Virtual Reality Modeling Language). Non-immersive virtual reality and animated 3-dimensional graphics on the Internet is achieved by using VRML. Integration of 3D geometric modeling functions into Web-browsers is enabled by VRML (Wang et al., 2009). In the early 1990's Internet was comprised of 2-dimensional web sites; however, with the introduction of VRML 1.0 and now VRML 2.0, virtual reality and animated 3-dimensional web sites are now possible.

A wider range of a user's perceptual senses and their ability to easily navigate through the visual representation is possible using VRML (Maletic et al., 2001a,b). It is not only the navigation flexibility, additional attributes like shape, size, color add more information to the users perception once when the user get used to the conventions of a given visualization (Igor and Cottam, 2002). The advantage of VRML is that the models can be viewed using a browser and hence all the models generated are internet ready and can help in collaborative maintenance and knowledge transition for geographically distributed teams

3. Proposed 3d visualization tool

The proposed system aims at establishing a static 3D call graph model for COBOL programs using virtual reality modeling language. It receives the file having the function source code from the end user and establishes all functions and sub functions called by that function as a 3D model using VRML. The detailed architecture is given in Fig. 1.

3.1. Proposed tool architecture

A batch job scheduled on the mainframe transfers all the source code files from the mainframe server to an extraction folder as. txt files on the windows system using a file transfer protocol. The function name for which the call graph should be constructed is given as an input to the code parser. The code parser extracts all the function calls by scanning the CALL keyword from the text files by starting the scan from the main function file.

It establishes the levels of the functions in the hierarchy and writes the details to a function list file. The model builder extracts the function hierarchy details from the function list file and builds the model .wrl file which contains the code for the three dimensional call graph model.

3.2. Code parser

The code parser will scan the files recursively starting from the main file given as input, to establish the levels of the functions in the call hierarchy and writes the function names and hierarchies to a text file (function list file). The code parser will ignore recursive functions calls. The establishment of function levels is shown in Fig. 2.

The code parser writes the function level and function name to the function list text file as shown in Table 1 below.

3.3. VRML model builder

The VRML model builder module is responsible for writing out the .wrl file for the model which is later visualized using a VRML enabled browser. The generated .wrl file can be viewed using Cortona 3D viewer 6.0 on the Internet Explorer

3.3.1. Algorithm – VRML model builder

The model builder retrieves all the records from the function list file and loads them to a function details structure. The structure contains the following data members.

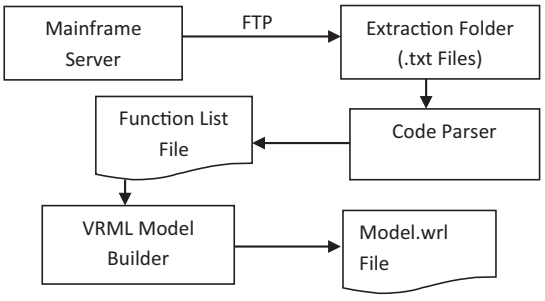

Fig. 1. Proposed Tool Architecture.

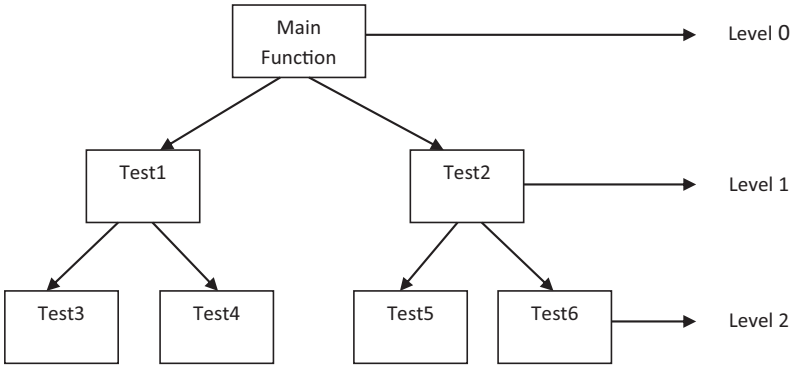

Fig. 2. Sample Function Call Hierarchy.

Table 1
Function List File Data.

Function Level	Function Name	
0	Main	
1	Test1	
2	Test3	Batch1
2	Test4	
1	Test2	
2	Test5	Batch2
2	Test6	

Table 2
First Batch Data.

Function Level	Function Name	
1	Test1	
2	Test3	Batch1
2	Test4	

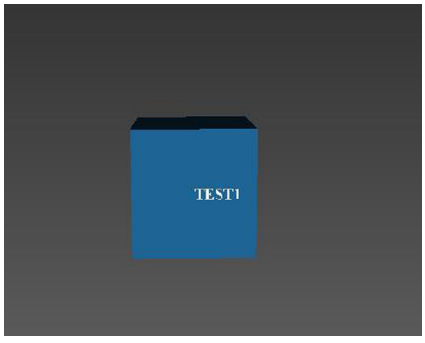

Fig. 3. Plotting Level 1 Node.

- Function level – A number to denote the level of the function in the call graph hierarchy
- Function name – Character array to store the function name
- X – A number to store the x coordinate value
- Y – A number to store the y coordinate value
- Z – A number to store the z coordinates value

It then reads the records in batches from the function details structure as shown in Table 1. The first batch read from the structure will have the following records as shown in Table 2.

For every function in the first batch the following is performed

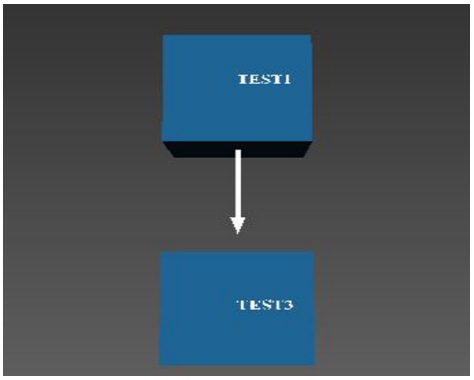

Fig. 4. Plotting Child Node.

- If the function level is 1 then the function is plotted as a node at default location as shown below
  X axis = 0, Y axis = 0 and Z axis = 0 i.e. (0, 0, 0)
- The function location (XYZ Coordinates) is written to the function details structure for the respective function
- The function name is also retrieved from the file and written at the location (0, 0, 1) such that the name is visible on the node as shown in Fig. 3.
- The location values for X Y and Z axis is written to the function details structure for the corresponding function name.

For each subsequent functions read from the first batch

- If current function level > previous function level. Then the previous function becomes the parent node to the current function.
- The parent node location is retrieved from the function details structure and the child node location is determined by subtracting a fixed distance (the distance between two nodes in the graph is taken as 3) from the Y-axis of the parent node.
- Function TEST3 has a function level 2 whereas function TEST1 has a function level 1 and hence TEST3 is the child node of TEST1.
- The location of TEST1 is (0, 0, 0) and the location of child node is derived by subtracting 3 from the y-axis that is (0,−3,0) and is plotted as shown in Fig. 4
- If previous function level is equal to current function level than the previous function is the sibling of the current function.
- The sibling node location is retrieved from the function details structure and the current function is plotted at a location by adding the fixed distance 3 to the x-axis of the sibling node.
- The location of TEST3 is (0,−3,3) and the location of TEST4 is (3,−3, 0). The node is plotted as shown in Fig. 5 below

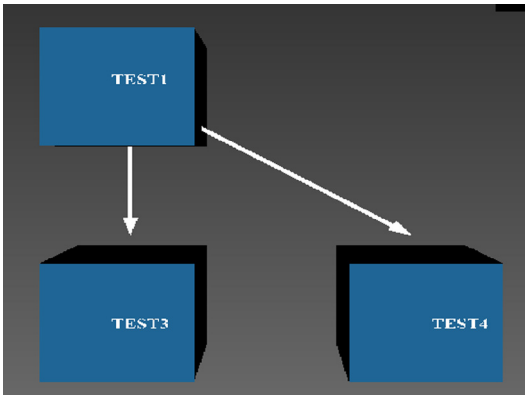

Fig. 5. Plotting Sibling Node.

Table 3
Function List File Data.

Function Level	Function Name	
0	Main	
1	TEST1	Batch1
2	TEST3	
3	TEST4	
2	TEST2	

If previous function level is greater than the current function level (as highlighted in red in Table 3) then the following steps are performed

- The corresponding sibling node of the current node is found by traversing the function details structure in the reverse until it reaches its sibling node which is at the same level. The function details structure is traversed in the reverse from function TEST2 until it reaches its sibling node i.e. TEST3. The location of TEST2 is computed as number of nodes between TEST2 and TEST3 $*$ 3 + x axis location of TEST3. The location is computed such that the child nodes of TEST2 does not overlap on the child nodes of TEST3. The graph plotted for the above scenario is shown in Fig. 6 below

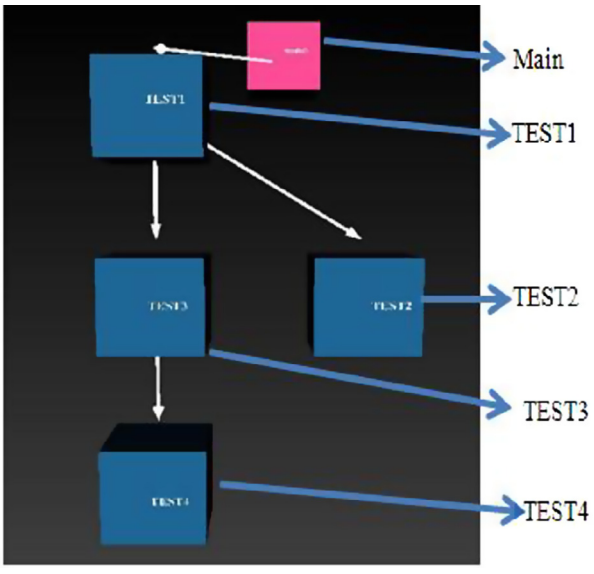

Fig. 6. Plotting Sibling Node.

Table 4
Function List File Data (repeating functions).

Function Level	Function Name	
0	Main	
1	TEST1	Batch1
2	TEST3	
2	*TEST4*	
1	TEST2	Batch2
2	TEST5	
2	TEST6	
3	*TEST4*	

Table 5
Tool Feedback Analysis Table.

Quality Attributes	Survey Results –Positive Feedback
Product Usefulness	95%
Ease of Use	85%
Ease of Learning	85%
Satisfaction	90%

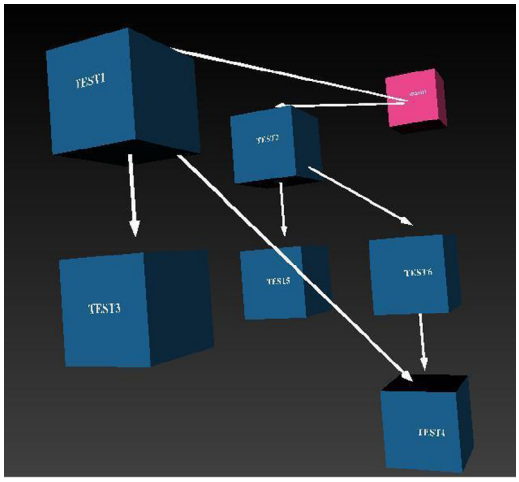

Fig. 7. Call Graph for repeating functions.

There can be functions that are repeatedly being called by functions from different batches. For the given batches in the table 4 below, function TEST4 (shown in italics) is being called more than once from Batch1 and Batch2 by two different functions

Before writing the node TEST4 the function details structure is scanned to check if the function is already written. If it is written then the location of the current function is retrieved and an arrow mark is drawn between the retrieved location and the previous function. The node is plotted as shown in Fig. 7 below

3.4. User interaction capabilities with the model

The model created by the VRML Model Builder is viewed using Cortona3D Viewer. The viewer offers users the walk, fly and examine navigation modes. Each navigation mode has several options like plan, pan, turn and roll. The camera motion and its orientation in the 3d space are determined by the combination of navigation modes and options. The viewer display window with its entire navigational buttons is shown in Fig. 8.

The walk navigation mode in combination with the plan option as shown in Fig. 9 is used for moving forward and backward in the 3d space. The walk navigation mode in combination with pan

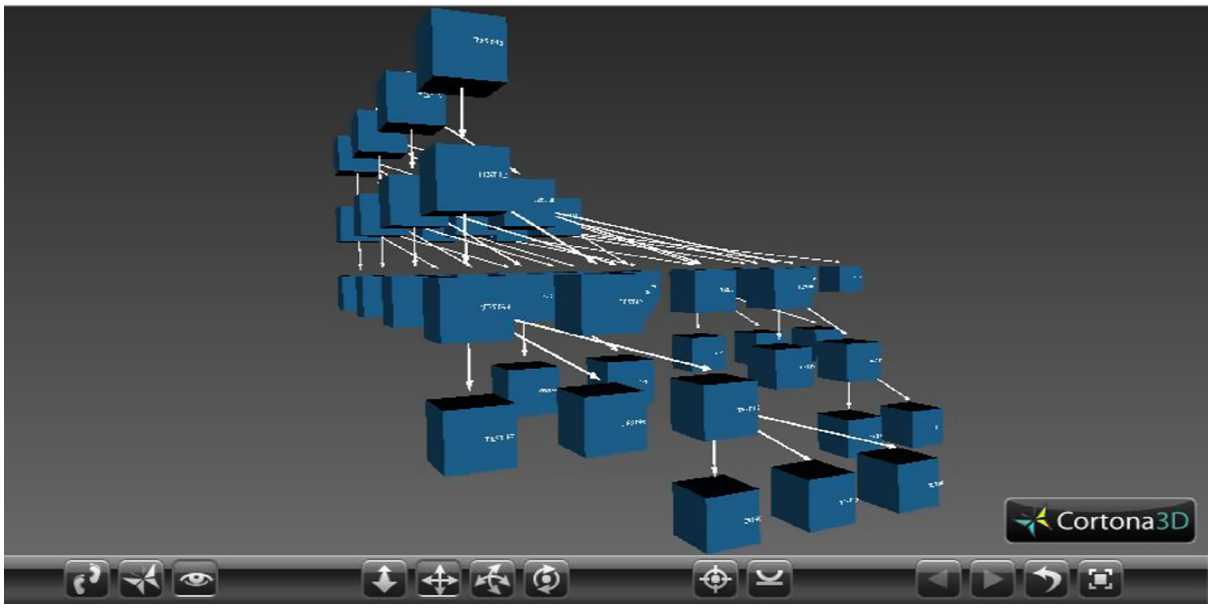

Fig. 8. VRML 3D Viewer with Navigation options.

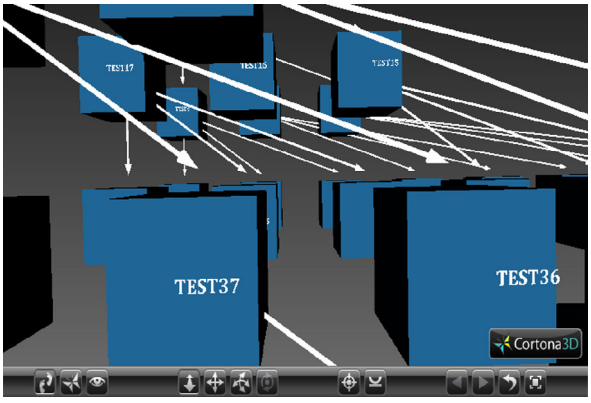

Fig. 9. Walk + Plan Navigation Mode.

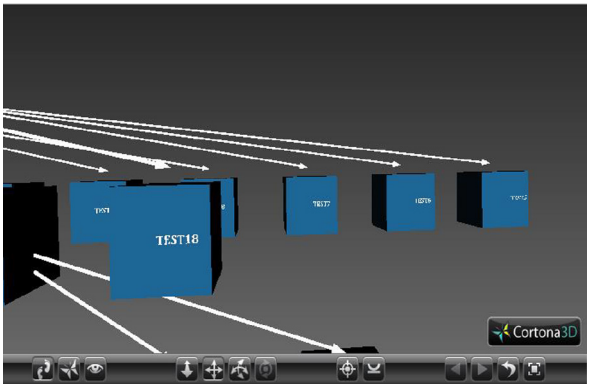

Fig. 10. Walk + Pan Navigation Mode.

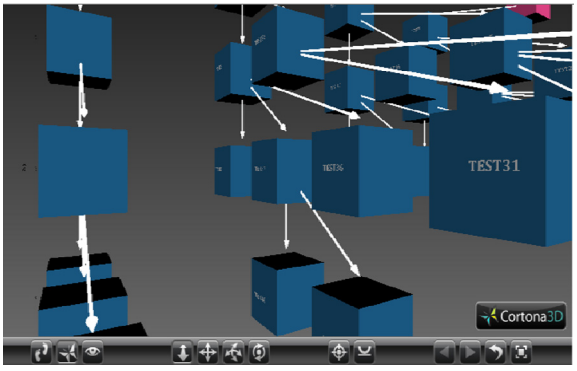

Fig. 11. Walk + Turn Navigation mode.

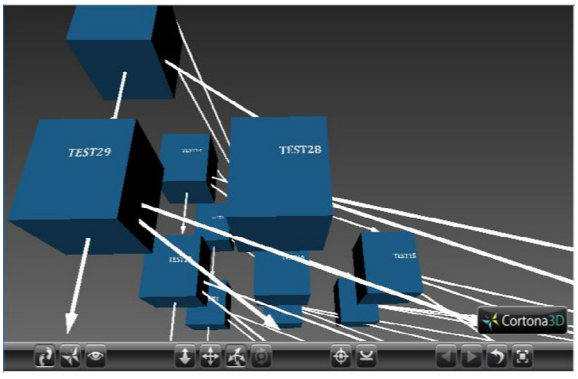

Fig. 12. Fly + Pan Navigation Mode.

option as shown in Fig. 10 is used for moving left and right in the 3d space.

The walk navigation mode in combination with turn option is used for changing the angle of the camera in the 3d world. The fly navigation mode in combination with plan option is used for moving the camera forward of downward towards its longitudinal axis. The navigational modes are shown in Figs. 11 and 12.

3.4.1. Editing the VRML model

The complexity of the model can be reduced by editing the function list file and rerunning the VRML model builder. The user can eliminate unwanted batches and focus only on the batches needed by editing the function list file. The function list file batches shown in Table 8 are edited to hold only the first two batches. The

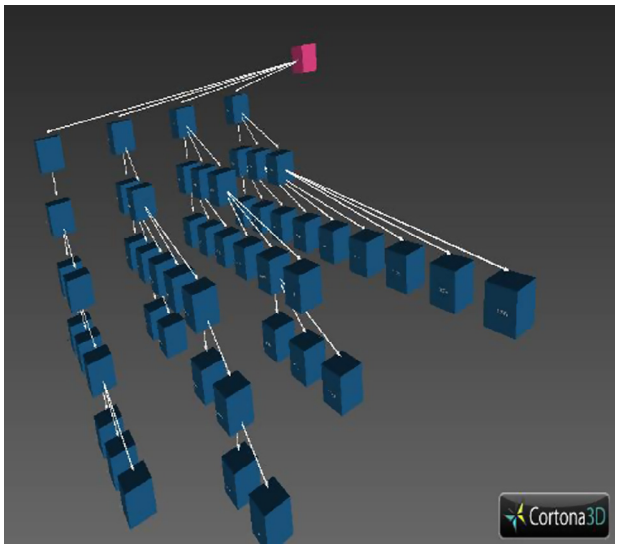

Fig. 13. Model with Four Batches.

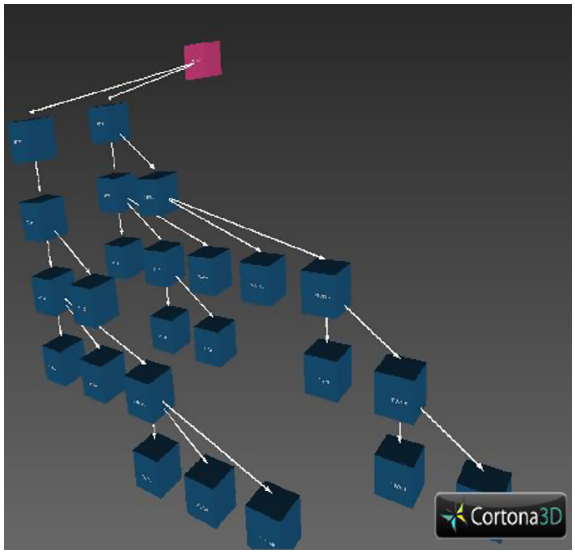

Fig. 14. Edited Model with two batches.

model generated before editing and after editing the function list file is shown in Fig. 13 and 14. The complexity of a generated model can be reduced by this approach and the users can analyze only the needed batches rather than analyzing the call graph for the entire system

4. Validation

In this section we detail the controlled experiment carried out to investigate the following research questions

4.1. Research questions and measures

4.1.1. Research question 1

Does the use of the COBOL visualization tool reduce the manual effort needed with respect to change impact analysis? Here we seek an evidence of reduction in the time taken for understanding the call hierarchy among COBOL modules and understanding the impact of change to a given module in the hierarchy.

4.1.2. Measure

The Manual effort here refers to the time taken for establishing the call graph for a given system. It also includes the time taken by the user to manually navigate through all the files and construct the call graph and identify the number of modules that can have an impact on the given module.

4.1.3. Research question 2

Is there a significant difference in task accuracy when performing maintenance tasks using the visualization tool?

4.1.4. Measure

Task accuracy for both the groups is calculated using precision. The precision for is calculated by dividing the number of students who completed the tasks correctly by the total number of students in each group. The precision measure helps in identification of the impact of the tool on the group's results.

4.1.5. Research question 3

How far the user perceptions are enhanced with the use of the 3d call graph model? Is the tool enhancing the user experience on the maintenance task?

4.1.6. Measure

Here the user experience with the usage of the tool is recorded. The users will be rating their experiences on the tool from different perspectives like ease of use, ease of learning, satisfaction with respect to model generation and edition.

4.2. Subjects

The subjects comprised of 40 computer science students who have completed their software engineering and software project management courses. All the students were in the final year of their graduation. The students had an in-depth knowledge on software maintenance and the change management process. The students were randomly assigned to two groups (Vz-Group and no-Vz group). Each group had 20 members in it. The Vz group will be using the visualization tool to complete the activities whereas the no-Vz group will be performing the task manually without using any tools.

None of the students were having any kind of training or experience on COBOL or Mainframes. We have chosen subjects who had no prior experience in COBOL or Mainframes so as to test the effectiveness of the tool in aiding an inexperienced user. The purpose of this visualization tool is for aiding graduates who are assigned such maintenance tasks in the industry without any prior experience.

4.3. Activity

The following activity was assigned to students in both the groups.

Construction of the call graph using the sample test files and to identify the following (The list of test functions used is listed in Table 8)

- Establishing the number of functions that are calling functions for function TEST28
- Establishing the number of functions that are called by the Function TEST50

4.4. Experimental set up

4.4.1. Hypothesis formulation

Our experiment has one dependent variable (the use of the visualization tool) and two treatments (Vz [use of visualization

tool], no-Vz [visualization tools are not used]). Our experiment has two dependent variables

TCA – Time taken to complete the activity
ΔP – Precision of the completed activity
Null Hypotheses:
H_0: $\mu_{[TCA]\ Vz} \geq \mu_{[TCA]\ no\text{-}Vz}$
H_0:$\Delta P_{Vz} = \Delta P_{no\text{-}Vz}$
Alternate Hypotheses:
H_1: $\mu_{[TCA]\ Vz} \leq \mu_{[TCA]\ no\text{-}Vz}$
H_1: $\Delta P_{Vz} \neq \Delta P_{no\text{-}Vz}$

The experiment was conducted using test files containing COBOL programs. Each Test file had one COBOL program. There were 50 Text files used in this experiment. The programs were chosen such that each program is either calling or being called by another program. All possible conditions were considered like a program calling multiple other programs and a program being called by multiple programs to simulate the complexity in real world applications

4.4.2. Experimental procedure

- All Subjects were given a brief introduction on how program calls are established in COBOL programs. They were shown a sample file with a call statement. The subjects were given an introduction on the significance of mainframe systems and the crucial role of maintenance engineers in maintaining such systems
- All subjects were given detailed instructions on the activity they need to complete and the questionnaires that they need to fill for the completed task.
- The subjects of Vz group were asked to install Cortona 3D viewer for viewing VRML Models. They were given sample VRML models two days before the experiment so that they are accustomed to the viewer and model analysis techniques.
- Vz Group subjects were given copies of the visualizer tool and were given a demo on the programs and generation of models using the visualizer before the start of the experiment.
- All subjects performed the assigned tasks and recorded the time using the questionnaires. The subjects were also briefly interviewed on problems they faced when they were trying the complete the activities. Vz group subjects were interviewed

Table 6
Vz Group Observations.

Serial Number	Time Taken to Complete Activity (Min)
1	3
2	3
3	2
4	4
5	3
6	3
7	2
8	2
9	3
10	2
11	5
12	4
13	4
14	3
15	4
16	5
17	3
18	4
19	4

Table 7
Vz Group Observations.

Serial Number	Time Taken to Complete Activity (Min)
1	20
2	15
3	10
4	30
5	15
6	25
7	12
8	15
9	20
10	10
11	27
12	20
13	15
14	22
15	14
16	20

Table 8
Function List File Data for 50 Functions-Used for result validation in Section 4.

Function Level	Function Name
3	TEST5
3	TEST6
3	TEST7
2	TEST2
3	TEST8
3	TEST9
3	TEST10
2	TEST3
3	TEST11
3	TEST12
3	TEST13
2	TEST4
1	TEST1
3	TEST18
4	TEST21
4	TEST22
4	TEST23
3	TEST19
3	TEST20
2	TEST15
2	TEST16
3	TEST24
3	TEST25
3	TEST26
2	TEST17
1	TEST14
5	TEST34
5	TEST35
3	TEST30
3	TEST31
2	TEST28
3	TEST36
4	TEST39
4	TEST40
3	TEST37
3	TEST38
2	TEST29
1	TEST27
3	TEST43
5	TEST48
5	TEST49
5	TEST50
4	TEST45
4	TEST46
4	TEST47
3	TEST44
2	TEST42
1	TEST41
0	Main

on the usefulness of the tool and suggestions for further improvement of the tool

4.4.3. Data gathering

Data gathering was done using questionnaire. Subjects were asked to record the time after the completion of the activity. The Vz group was also asked to rate the experience of using the tool. The users gave a rating between 1 and 5 for the below mentioned categories.

- Product Usefulness
- Ease of Use
- Ease of Learning
- Satisfaction

Any other feedback was recorded in a brief interview with the users at the end of all activities.

5. Results

5.1. Research question 1

To understand whether the tool improves the performance of one group over the other we have used independent two samples t test. We had to remove the results of four students from no-Vz group and one student from Vz group because of the wrong outputs arrived by these students. The time recorded by these two groups is given in Tables 6 and 7.

$X_{Vz:}$ Mean of Vz group = 3.3
n_1: Number of observations in Vz group = 19
s_1: Standard Deviation in = 0.94
$X_{no\text{-}Vz:}$ Mean of no-Vz group = 18.1
$n_{2:}$ number of observations in no-Vz Group = 16
s_2: Standard Deviation = 5.89
$\alpha = 0.05$

$$t = \frac{X_{vz} - X_{no-vz}}{\sqrt{s_p^2\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

$$s_p^2 = \frac{(n_1 - 1)s_1^2 + (n_1 - 1)s_2^2}{n_1 + n_2 - 2}$$

$$t = \frac{3.3 - 18.6}{\sqrt{16.20\left(\frac{1}{19} + \frac{1}{16}\right)}} = -11.50$$

Degrees of freedom = $n_1 + n_2 - 2 = 33$
Rejection Region $t > 1.692$ or $t < -1.692$

Since the calculated t value falls inside the rejection region the Null hypotheses is rejected and we concluded the performance of the Vz group with respect to the completion of the activity is better than the no-Vz group. The T test led to a positive conclusion that the tool plays an important role in the reduction of time with respect to understanding the architecture of an existing system

5.1.1. Research question 2

The research was focused on assessing the quality of the results by the two groups. We have used precision to compare the results. Precision is calculated as shown below

5.1.2. Precision

ΔP = Number of students who completed the task correctly/total number of students

Precision for Vz group is 19/20 = 95%
Precision for no-Vz group is 16/20 = 80%

This concludes that the precision of the activity increases with the usage of the tool.

5.1.3. Research question 3

The survey on the product quality attributes received as a feedback from no-Vz group is given in Table 5. Majority of the

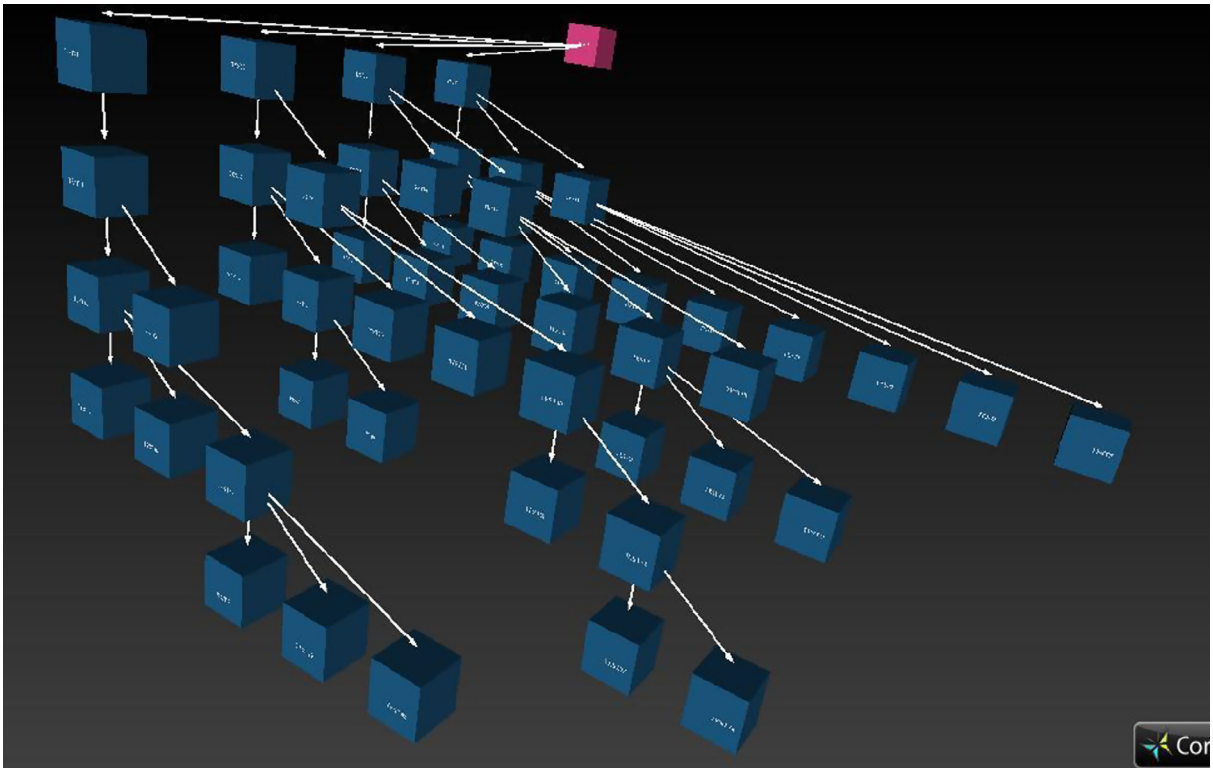

Fig. 15. Call Graph for 50 Functions.

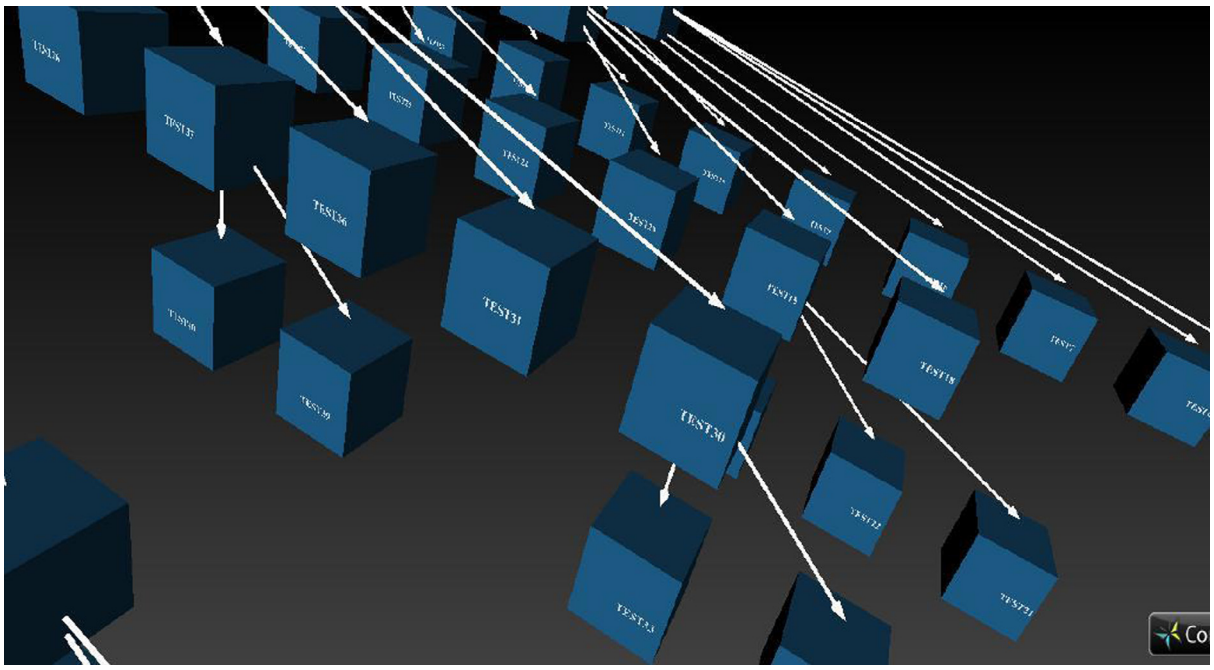

Fig. 16. Call Graph – Closer View.

users in no-Vz group were satisfied with the support of the tool. Many students felt that the tool drastically reduced the time taken for construction of the call graph and identification of all the modules that a module impacts. The students were also impressed with the amount of navigational flexibility they achieve in a virtual reality environment when compared to that of a 2D environment.

The concept of program visualization in virtual reality made the concept of change impact analysis an interesting task and aided the Vz group towards the completion of the task with a lesser time and greater accuracy. However a few students felt that the navigation to be a little harder with so many plotted nodes and they had to repeat the process of searching a node from the beginning as they sometimes felt lost and confused in the 3D space.

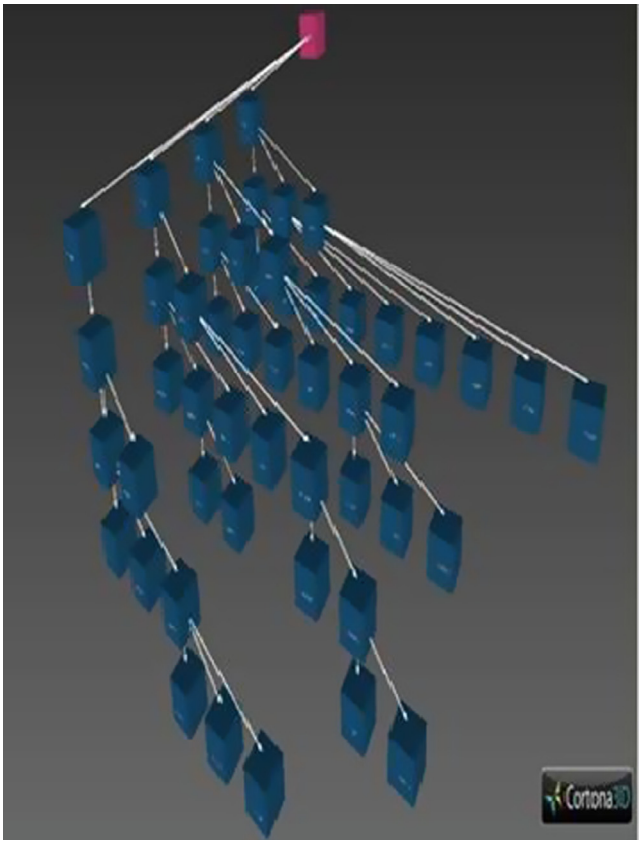

Fig. 17. Call Graph – Rotated View.

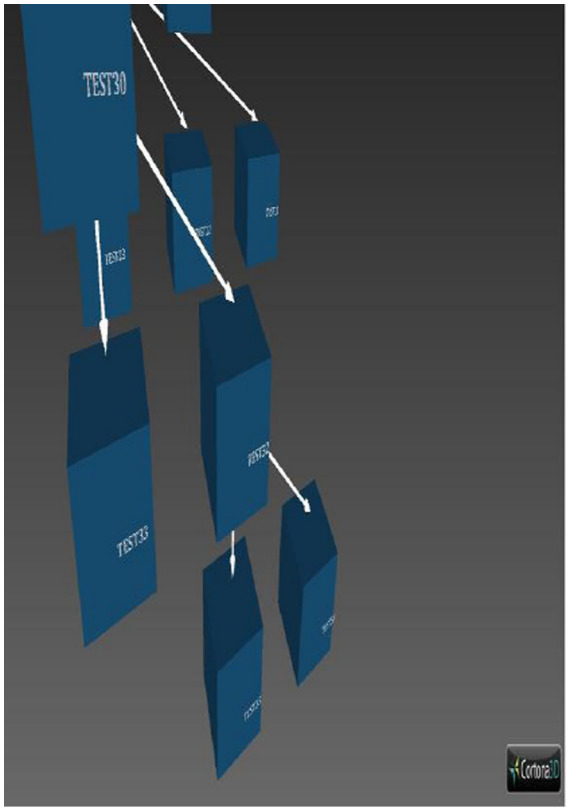

Fig. 18. Call Graph – Node View.

The function list file (Table 8) on given as an input to VRML Model builder generated the 3D call graph model as shown in Fig. 15. The call graph closer view, rotated view and node view is shown in Figs. 16–18.

6. Threats to validity

A description on the several threats that can have an impact on the validity of our results

6.1. Internal validity

Internal validity is the process of establishing a relationship between the uses of the visualizer with the reduction in change impact analysis time. All subjects were randomly assigned to the two groups and the test materials distributed to the two groups were the same. Both the groups had same level of expertise and exposure to software maintenance concepts. The briefing to all the subjects was done using the same facilitator in order to reduce the impact of the training on the outcomes. The subjects who were not able to complete the tasks correctly were removed from the t test analysis to have comparable results.

6.2. External validity

External validity relates to the generalization of results. Our experiment was limited to test files that were linked to each other by the CALL keyword for function call in COBOL programs. As this is in sync with the real world mainframe applications, the reduction of time is possible as per the results achieved in our experiment. The usage of the tool and the change impact analysis time was tested using students who are new to mainframes and COBOL programming. Therefore the results on the reduction of call graph comprehension will be better for experienced engineers in the industry. Thus our choice of test files and subjects reduces the threat to external validity of our study.

7. Conclusion

One of the major motivations of this research was to improve on visualizations provided for legacy systems. Moreover, this is an attempt towards understanding the impact of 3D models for legacy system understanding. The research has yielded positive results and we have found that the subjects using the tool were 15% more accurate than then subjects not using the tool. The user experience feedback on the tool was also encouraging and it clearly establishes the fact that the presence of such a tool will enhance the productivity of maintenance engineers.

The call graph is generated for the function that is supplied as input to the code parser program and this helps the end user to have a control over the functions for which the model needs to be generated. The end user can also simplify the visualization by directly deleting the number of batches from the function list file and rerunning the VRML model builder program. This enables the user to reduce the complex nature of the views and enable further exploration of batches independently.

The model is achieved by running two C++ programs and viewed on the Internet Explorer using Cortona 3D Viewer. This is a major advantage as there is no need for installation of licensed packages or tools for visualization. The time taken for the system to generate the model is less than a few seconds and moreover the model is around 11 KB for a system that has 50 functions to be part of it. These models can be viewed using a browser and hence they can be viewed and distributed by geographically distributed teams without any additional effort.

The performance of the system has been stable for around 50 functions. The given research and experimental approach has proved that analyzing a sample system that has 50 or more functions on the mainframe and generating a call graph will take about 15 min to half an hour manually. This time will further increase for industry standard applications as we have only used test functions in our experiments. Moreover engineers may not be motivated to do this manual tedious work during the impact analysis phase and there are a lot of chances that results of the impact analysis can be error prone.

The proposed model helps with the reduction of time and error towards understanding the architecture and impact of changes needed on the system. This approach will definitely help engineers towards understanding complex legacy systems. The reduction in time taken to understand a system during impact analysis will lead to reduction in cost of the maintenance.

8. Future work

This system is applied only for COBOL programs. The builder will not be able to map recursive functions in the system. Future work can be with respect to extending the model builder for inclusion of metrics such as nodes with higher fan-in and fan-out can be denoted with a specific color. The model can be further revised to show the parameters being passed between function calls. The dependency of functions based on JCL's is not resolved here and a model with such a perspective can be plotted to help maintenance engineers get a better understanding of the legacy systems.

References

Alaranta, M., Betz, S., 2012. Knowledge problems in corrective software maintenance–A case study. In: Proc. 45th Hawaii International Conference on System Science (HICSS), pp. 3746–3755.

Teyseyre, Alfredo R., Campo, Marcelo R., 2009. Overview of 3D software visualization. IEEE Transactions on visualization and Computer Graphics, 15.

ATOS – White paper on Mainframes in Perspective a classic going strong.

Card, S., MacKinlay, J., Shneiderman, B. (Eds.), 1998. Readings in Information Visualization: Using Vision to Think. Morgan Kaufmann.

Caserta, P., Zendra, O., 2011. Visualization of the Static Aspects of Software: A Survey. IEEE Trans. Visualization Comput. Graphics 17, 913–933.

Colin, Ware, 2012. Information Visualization: Perception for Design. Elsevier.

Diehl, S., 2007. Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software. Springer.

Fjeldstad, R., Hamlen, W., 1979. Application program maintenance study: Report to our respondents. In: GUIDE 48, Philadelphia, PA.

Hunt, B., Turner, B., McRitchie, K., 2008. Software Maintenance Implications on Cost and Schedule. In: Aerospace Conference, pp. 1–6.

Igor, Rojdestvenski, Cottam, Michael, 2002. Visualizing metabolic networks in VRML. Proceedings of the Sixth International Conference, USA, on Information Visualization (IV'02).

Khadka, Ravi et al., 2013a. A structured legacy to SOA migration process and its evaluation in practice. 2013 IEEE 7th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA). IEEE.

Khadka, Ravi et al., 2013b. Migrating a large scale legacy application to SOA: challenges and lessons learned. 2013 20th Working Conference on Reverse Engineering (WCRE). IEEE.

Koschke, Rainer, 2002. Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey. J. Software Evol. Res.

Lei, Wu, Sahraoui, Houari, Valtchev, Petko, 2005. Coping with legacy system migration complexity. 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05). IEEE.

Maletic, J.I., Leigh, J., Marcus, A., 2001a. Visualizing Software in an Immersive Virtual Reality Environment. Proceedings of ICSE'01 Workshop on Software Visualization, Toronto, Ontario, Canada, pp. 49–54.

Maletic, J.I., Leigh, J., Marcus, A., Dunlap, G., 2001b. Visualizing Object Oriented Software in Virtual Reality. Proceedings of International Workshop on Program Comprehension (IWPC01), Toronto, Canada, pp. 26–35.

Palvia, P., Patula, A., Nosek, J., 1995. Problems and issues in application software maintenance management. J. Inf. Technol. Manage. V I (3), 17–28.

Robertson, G., Card, S.K., Mackinlay, J.D., 1993. Information visualization using 3D interactive animation. Comm. ACM 36 (4), 57–71.

Storey, M.A., Bennett, C., Bull, R.I., German, D.M., 2008. Remixing Visualization to Support Collaboration in Software Maintenance. Proceedings Frontiers of Software Maintenance(FoSM 2008), Beijing, China, pp. 139–148.

Van Geet, J., Demeyer, S., 2008. Lightweight visualizations of cobol code for supporting migration to SOA. Electron. Comm. Eur. Assoc. Software Sci. Technol. 8. http://journal.ub.tu-berlin.de/index.php/eceasst/issue/view/17.

Van Geet, Joris, Demeyer, Serge, 2010. Reverse Engineering on the Mainframe: Lessons Learned from "In Vivo" Research. IEEE Computer Society.

Wang, Haopeng, Zhao, Kai, Liu, Bing, Binru, Chen, 2009. Research on VRML Modeling for Web-Based Virtual Product Development. Networking and Digital Society, 2009. ICNDS '09. International Conference on, 1, pp. 7–12.

Xiaohua, Xian, 2003. 2D & 3D UML-Based Software Visualization for Object-Oriented Programs Diss. Concordia University.

Zhang, K., 2003. Software Visualization: From Theory to Practice. Springer Inc.