AN EMPIRICAL STUDY ASSESSING THE IMPACT OF SeeIT 3D ON COMPREHENSION

by

Grace Havila Jetty

Submitted in Partial Fulfillment of the Requirements

for the Degree of

Master of Computing and Information Systems

in the

Computer Science and Information Systems

Program

YOUNGSTOWN STATE UNIVERSITY

May, 2013

AN EMPIRICAL STUDY ASSESSING THE IMPACT OF SeeIT 3D ON COMPREHENSION

Grace Havila Jetty

Signature:

Grace H Jetty, Student Date

Approvals:

Bonita Sharif, Thesis Advisor Date

John Sullins, Committee Member Date

Alina Lazar, Committee Member Date

Bryan DePoy, Dean of School of Graduate Studies and Research Date

Abstract

Software visualizations are meant to help developers comprehend software systems. They are especially useful for large software systems with tens of thousands of classes. There have been many visualizations been proposed in the literature with relatively little empirical evidence showing their usefulness to developers. In this thesis, we conduct an empirical study to assess the impact a software visualization tool namely, SeeIT 3D (an Eclipse plug-in) has on performance of certain software tasks. Six different tasks in three different task categories, developed in the context of understanding an open-source system, GanttProject, written in Java. Ninety-seven subjects were recruited from three different universities and split into two groups; one group used the SeeIT 3D plug-in while the other did not use the plug-in. The main goal was to determine the impact and added benefit of SeeIT 3D while performing typical software tasks within the Eclipse IDE. Results indicate SeeIT 3D performs significantly better in one task category namely overview tasks. There is also a significant difference in the way experts and novices solve tasks. These results indicate when software visualization tools are useful for developers. They might not be useful for all tasks but are worthwhile for others.

Acknowledgements

Firstly, I would like to thank God for being with me all the time. I thank my family members for showering their endless love and care, without their love I would not have come to this stage.

I thank Dr. Bonita Sharif, my thesis advisor, who was the reason to step into this research work in my Master's; she has being my support throught out my thesis accomplishment. I was really amazed by her patience and encouragement through out my research work.

I thank all my friends for their support, encouragement and help to complete my thesis study by spending their valuable time. I thank committee members Dr. John Sullins and Dr. Alina Lazar.

Once again, I thank my parents and my sister for their prayers and encouragement. I thank all the members who directly or indirectly helped me to complete my research work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Visualization is the process of viewing abstract things. Software Visualization (Petre and Quincey 2006) (Price, Baecker et al. 1993) (Zhang 2003) is a type of visual representation of the software information in the form of measures such as metrics and granularity levels. It is claimed that software visualization tools improve the interaction between the user and the system, by providing better comprehension about the system to the user. While using software visualization tools, integration of the visualization tools within an Integrated Development Environment (IDE) is important. There have been many software visualization tools proposed in the literature (Bassil and Keller 2001, Teyseyre and Campo 2009). Most of these tools do not provide an empirical validation on the usefulness of the tool to the developer.

In this work we present a controlled experiment that gathers data via two different modes: online questionnaires and an eye tracker. Online questionnaires are traditional methods to collect data. Eye tracking is the process of measuring the movement of eye while a subject is solving certain software tasks. An eye tracker is used to measure the postion of the eye with respect to the movement. Eye tracking depends on the mental state of mind, thinking and cognitive process (Just and Carpenter 1980). Applications of eye tracking - used to characterize the comprehension of software programs (Bednarik and Tukiainen 2006) by using different measures, used to track the visual attention in pair programming (Pietinen, Bednarik et al. 2008), used in gaming, clinics, etc. In this thesis, we focus on comprehension tasks such as bug fixes, feature additions and general overview tasks. An eye tracker is used to gather data from subjects while they solve the tasks. The results will be provided as feedback to the tool developers, who may use it in further improving the tool. Kagdi et al. (Kagdi, Yusuf et al. 2007) state how eye tracking can help in validating and assessing software visualizations.

1.1 Motivation

In previous work (Gadapa 2012), an attempt was made to understand to usability of one such software visualization tool namely, SeeIT 3D (Ramírez 2010). This work was an observational study whose goal was to determine if SeeIT 3D was useful to developers while they performed the task. The results and observations got from (Gadapa 2012) was used by the SeeIT 3D tool developers to improve the SeeIT 3D plug-in. This study led us to prepare another experiment that tried to determine the impact SeeIT 3D has on performance of tasks.

This project extends previous work (Gadapa 2012) that tried to bridge the gap between the tools and empirical validations. Gadapa's study was an observational study that looked at the usability of SeeIT 3D on general comprehension tasks (mainly overview tasks on a large system, JFreeChart). The study presented in this thesis assesses the effectiveness of the SeeIT 3D plug-in with respect to bug finding tasks, new feature tasks, as well as overview tasks. The main motivation is to find whether the SeeIT 3D tool is helpful in debugging: finding the bugs and fixing them, find feature tasks to be implemented and in solving overview tasks. In order to do this two groups were formed, one that used SeeIT 3D to solve the tasks and the other that used only Eclipse with SeeIT not available to them.

1.2 Contributions

The main contribution of this thesis is an empirical study and in particular a controlled experiment that assessess the added benefit of one 3D software visualization tool, SeeIT 3D (Ramírez 2010) in the context of an open-source system namely, GanttProject. This is a direct extension of Gadapa's work (Gadapa 2012). Two methods of data collection were used and the study was conducted at three universities with a ninety-seven subjects.

Another contribution is to determine how experts differ from novices while they solve the tasks with and without the SeeIT 3D plug-in.

1.3 Research Questions

The following are our research questions that we seek to answer.

RQ1: Does the SeeIT 3D visualization tool help a developer in software comprehension tasks?

RQ2: Is there a difference between experts and novices with respect to the SeeIT 3D visualization tool?

1.4 Organization

This thesis is organized as follows. The next chapter gives a brief introduction to software visualization including the most popular software visualization tools and empirical studies done in software visualization tools. Chapter 3 gives an overview of

SeeIT 3D. Chapter 4 discusses details of our study setup and design. Chapter 5 presents observations and results. Chapter 6 concludes the thesis and presents future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter talks about an overview of software visualization and empirical studies conducted on software visualization tools including any eye tracking studies done in the software engineering domain.

2.1 Software Visualization and Metaphors

Software visualization is the process of graphical representation of the information, algorithms and program code; it reduces the complexity in understanding the software systems; software maintenance; reverse engineering and software evolution analysis. Software visualization can be used in animations, program executions, visualization of object-oriented programs, debugging, process analysis, requirement analysis etc. Properties are identified by the set of taxanomies of software visualization (Price, Small et al. 1992). Roman and Price (Roman and Cox 1993) explained the attributes of the software visualizations briefly, they are listed below

- Scope and Content: What program aspect is being visualized?
- Abstraction: What information is conveyed through visualization?
- From and Technique: How is the graphical data conveyed?
- Method: How is the specification of visualization done?
- Interaction: How can the user interact with the visualization?

The different dimensions of software visualzations (Maletic, Marcus et al. 2002) are listed below.

- Tasks: Why is the visualization needed?
- Audience: Who will use the visualization?
- Target: What aspects of the software are to be represented?
- Representation: How will it be represented?
- Medium: Where will it be represented?

Metaphors are the graphical representations that show the affect of the software visualization. They are can be geometrical shapes like we see in SeeIT 3D or can be real world entities. Things to taken into considerations are characteristics of the metaphors for the SV are effectiveness which means the medium used for visualization and expressiveness which means how efficient the visualization for the comprehension of the software system, which can be examined by considering scope of the visualization, medium, consistency of the metaphors, semantics, ease of navigation.

2.2 Examples of Software Visualization Tools

SHrimP (Simple Hierarchical Multi-Perspective) (Storey, Best et al. 2001) is a visualization technique to enhace how people understand complex software. It supports multiple views (graphical and textual) in the form of a nested graph.

Another fairly recent visualization to help developers with working space is Code Bubbles (A. Bragdon 2010). Collection of light weight editable code fragments are called "Bubbles". Grouping up of these bubbles form the working set. The front end is implemented using Microsoft Windows Presentation and its back end is an Eclipse plug-in. Bubbles differ from Visual Studio or Eclipse. It uses reflow and elision which are applied only to the view. Bubbles are not overlapped, they use recursive algorithm to move away the overlapped bubbles. To refer a variable in the package it uses 'Bubble Stack', this is used to compare the fragments side-by-side. Program instances which are debugged are stored in channels that can be used for the comparisons.

CodeCrawler (Lanza 2003, Lanza and Ducasse 2005) is a language independent visualization tool that supports various types of software visualization views such as a complexity view, a class blueprint view, hotspot view and an evolution matrix view among others.

Code city (Wettel and Lanza 2008) is an integrated language independent 3D visualization tool written in Small talk, which is used for analysis of large object-oriented software system. City metaphor depicts software system as city, classes as buildings and packages as districts, which is shown in the figure below. To represent classes as buildings, polymetric view is used which help to understand the structure and detect problems of a software system in the initial phases of a reverse engineering process. Class methods are represented by the height of the buildings. Attributes are represented with both the height and width of the buildings. The visualization is interactive and navigable using the keyboard, i.e., it is easy to zoom in on details of the city or to focus on one specific district by spawning separate windows.

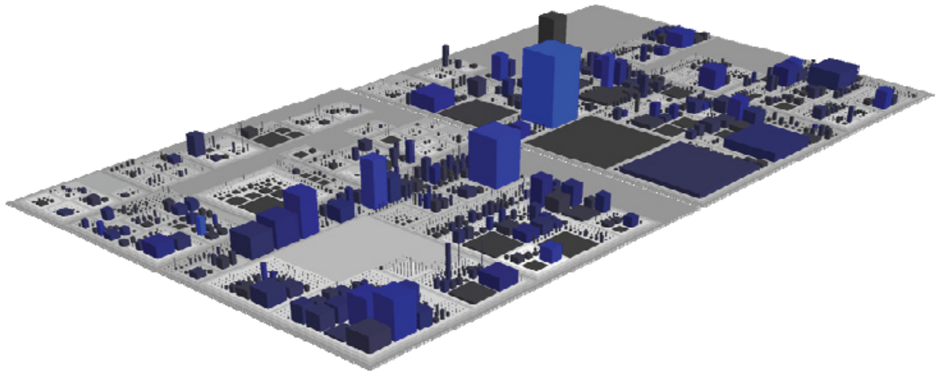


Figure 1. Code city visualization.

2.3 Empirical Studies on Software Visualization Tools

A brief description of empirical studies on software visualization tools is given below.

2.3.1 SeeIT 3D Observational Study by Gadapa (Gadapa 2012)

Gadapa conducted an observational study assessing the usefulness of SeeIT 3D, a software visualization tool on an open source Java system, Jfreechart (504 classes, 37 packages, ~ 73 LOC). Observations and results recorded were used by the authors of SeeIT 3D for tool improvement. The goal was to determine the usability of SeeIT 3D. Ten subjects participated in the study. There were 16 comprehension questions and 19 preference questions to rate the tool with respect to some criteria. They measured accuracy, time, difficulty and confidence levels for each question. The data was collected via paper based questionnaires and video recordings.

The subjects thought that the visualization response time was extremely slow while they did the study. 50% of participants thought it needed improvement.

Participants immediately noticed a bug in the drawing of arcs between related containers. They saw how it is useful for large projects and 57% said they would use it in the future. Based on the results of this study, the empirical study that forms this thesis emerged. The details of our study is presented next.

2.3.2 sv3D

Marcus et al. (Marcus, Comorski et al. 2005) conducted a usability study to evaluate sv3D, an earlier version of SeeIT 3D. sv3D is not part of an IDE like SeeIT 3D. The participants were divided into two groups for the purpose of comparing the performances obtained by using sv3D to the ones obtained by using tabular data with metrics values and the source code in an IDE. The questions are of two types: one type allowed the authors to obtain objective measures of the accuracy and completion time of the participants, while the other type provided subjective information about some of the sv3D features. The results showed that sv3D users requires more time to answers questions than the participants using tabular data and the IDE. Likewise, the average number of correct answers per user will not differ significantly between the two groups.

2.3.3 Extravis

In (Cornelissen, Zaidman et al. 2011) Cornelissen et al. reports on a controlled experiment for the quantitative evaluation of Extravis, a tool for the visualization of large traces. The authors defined a series of comprehension tasks that were addressed by two groups of participants that included graduate students, postdocs, professors, and subjects from industry. One group used the Eclipse IDE while the the other one had access to both

Extravis and Eclipse. The time needed and the correctness of the solutions given by the participants were measured. In this case, the results were statistically significant in both regards, showing a decrease in time requirements and an increase in correctness for the group using trace visualization.

2.3.4 Code City

Richard Wettel, Michele Lanza and Romain Robbes (Wettel, Lanza et al. 2011) conducted an experiment on CodeCity which shows significant statistical increase in terms of the correctness of tasks and reduce the task completion time, in which subjects are from both industrial and academia. Correctness and time were the main dependent variables in the study. Two subjects systems were used one to find the bugs and other for client sharing the information. Based on the background they grouped subjects as industry and academia; based on experience level they grouped beginners and advanced, they have assigned source code for control group and system model for the experimental group. From the data analysis of the study one of the task task4 was solved by most of the control group rather than experimental group the reason is this task requires deep knowledge in comprehension the task as this was programmed using the SmallTalk language. Overall Code city increased the correctness of the solutions of the program comprehension, and the completion time is reduced when compared with ECL + Exl users.

2.3.5 Code Bubbles

Code Bubbles has some analysis done to determine the usefulness of the metaphor (Bragdon 2010). The results show that by using Code Bubbles one can see functions side-by-side. Qualitative Evaluation is performed by taking feedback from professional developers. Participants are asked to perform 6 tasks- code comparison, understanding, interruptions, debugging, sharing, debugging session comparison. Ratings from the developers for system convience-5.0, learning system – 3.0, reading and editing (side-by-side)- 5.0, The overall rating of Code Bubbles is 4.0. Developers liked most of the features of code bubbles – vertical elision, continuous nature of workspace, labeling the areas of the workspace bar, debugging, open data structures values in bubbles, debug sessions able to compare information, saved debug sessions, channel interface, adding notes/flags to debug session, offloading information from their limited memory, multi-tasking, Etc. Most of the developers suggested for the integrating instant messaging, enabling workspace as real-time collaboration and coordination.

2.3.6 MetricViewEvolution

Lange et al. (Lange and Chaudron 2007, Lange, Wijns et al. 2007) conduct an experiment to validate different views for UML class diagrams: MetaView, ContextView, MetricView, and UMLCityView. This is a true experiment that is similar to the study conducted in this thesis. Unified Modeling language (UML) is the modeling language for the object-oriented systems. UML has some elements that represent the software program and the relations between them. Framework has tasks, views and properties. Tasks represent a unit of work to be done by the software engineer to accomplish a purpose.

Properties means characteristics of a model element. View is the exposure of the properties to perform the task. Types of Proposed views and what for they are designed are listed below-

- ***Content view***: Designed for program comprehension tasks
- ***Quality Tree view***: Designed to know the quality of the tasks
- ***Meta view***: Program Comprehension, maintenance and task completeness
- ***Clustering view***: it's a special instance of Meta view
- ***Metric view***: Quality evaluation and maturity/ completeness
- ***UML-city view***: its as combination of Meta and Metric view
- ***Search and highlight***: same as Meta view.
- ***Evolution view***: quality evaluation, prediction and monitoring

Implementation of these proposed views by MetricView Evolution tool. For the study there were 13 subjects who are PhD and MSc students from the Technische Universiteit Eindhoven. Most of the tasks were answered correctly, but 2 of the tasks took much time for the completion. Reasons for this to happen were: a) large degree of expertise needed. b) Usage of different versions of model. Overall correctness and efficiency was estimated. Understandability of the system was easily done by *Quality tree view; context view* has scored low. Scalability is the most common problem in visual sections. Metric view and UML-city view involves in the scalability problem.

2.4 An Overview of Eye Tracking

Since we use an eye tracker to collect data for our study, a briefi description of eye tracking and the apparatus used is given below.

An eye tracker is able to detect where a person is looking at on the screen. Visual attention refers to the focus on a particular location on the screen. It is known that visual attention triggers mental processes in order to comprehend and solve a given task (Just and Carpenter 1980). Visual effort is directly linked to the cognitive effort (Just and Carpenter 1980). A set of eye-tracking measures representing visual effort are derived from the eye gaze data. *Fixations* and *saccades* are two main types of eye gaze data. A fixation is the stabilization of the eyes on a particular location for a particular duration. Saccades are quick movements between eye fixations. A *scan path* is a directed path formed by saccades between fixations. According to eye tracking literature, processing of visual information occurs during fixations whereas no such processing occurs during saccades (Rayner 1998, Duchowski 2003).

2.5 Eye-tracking Studies in Software Engineering

Sharif et al. (Sharif and Maletic 2010, Binkley, Davis et al. 2013) study the impact of identifier style (i.e., camel case or underscore) on code reading and comprehension using an eye-tracker. They find camel case to be an overall better choice for comprehension. Sharafi et al. (Sharafi, Soh et al. 2012) conduct an eye tracking study to determine if gender impacts the effort, time, and ability to recall identifiers. Guehénéuc (Guéhéneuc 2006) investigated the comprehension of UML class diagrams. Jeanmart et al. (Jeanmart, Guéhéneuc et al. 2009) conducted a study on the effect of the *Visitor* design pattern on comprehension using an eye tracker. Sharif et al. (Sharif and Maletic 2010) also conducted an eye tracking study assessing the role layouts have in the comprehension of design pattern roles. Yusuf et al. (Yusuf, Kagdi et al. 2007) used eye-tracking equipment to assess how well a subject comprehends UML diagrams.

In this thesis, we use an eye tracker to determine the visual effort that a subject experiences when using the SeeIT 3D tool. These are termed as *white box* measures (Kagdi, Yusuf et al. 2007) because they are collected as the subject is performing the tasks versus after the fact. In this case, fixations are the white box measures used to determin visual effort in the evaluation of SeeIT 3D.

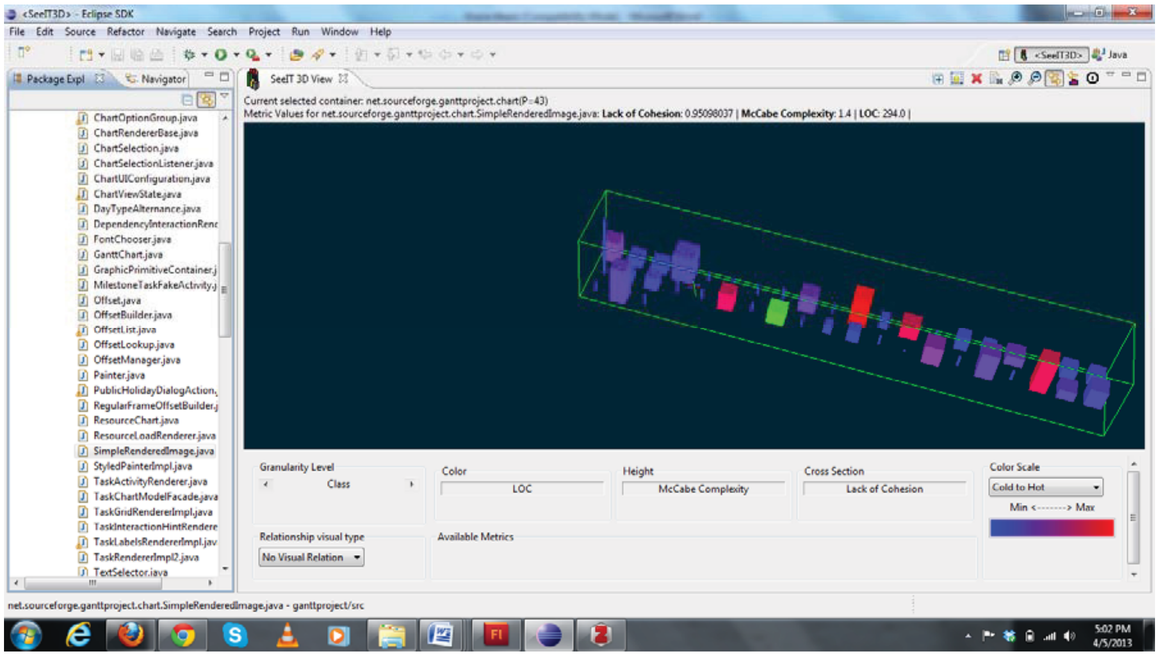
CHAPTER 3

Overview of SeeIT 3D

SeeIT 3D (Ramírez , Ramírez 2010) is an Eclipse plug-in. It analyzes source code, generates certain metric information and visualizes the system in containers. A more detailed description of SeeIT 3D is given in (Ramírez , Ramírez 2010). This chapter provides an overview of the features of SeeIT 3D from a user's perspective.

3.1 Metaphor

Metaphors are able to handle high amount of data from different source of information. It contains only the fixed number of visual properties. Polycylinders are the metaphors in this tool. SeeIT 3D has flexible metaphors which provides different visualization types. From the figure below, we can see the visualization of the GanntProject system package. In this figure, packages are depicted in the form of containers and classes in the form of polycylinders.



Current selected container: net.sourceforge.ganttproject.chart(P=43)
Metric Values for net.sourceforge.ganttproject.chart.SimpleRenderedImage.java: **Lack of Cohesion:** 0.95098037 | **McCabe Complexity:** 1.4 | **LOC:** 294.0 |

Granularity Level: Class
Color: LOC
Height: McCabe Complexity
Cross Section: Lack of Cohesion
Color Scale: Cold to Hot
Min <-------> Max
Relationship visual type: No Visual Relation
Available Metrics

net.sourceforge.ganttproject.chart.SimpleRenderedImage.java - ganttproject/src

5:02 PM
4/5/2013

Figure 2. SeeIT 3D visualization of a package in Ganttproject system

3.2 Metrics and Mappings

Metrics are the measure of software properties. SeeIT 3D has 4 metric values: *Lines of Code, Lack of Cohesion, Mc Cabe's complexity and control structures*, which are briefly discussed in (Gadapa 2012). A brief description is given below.

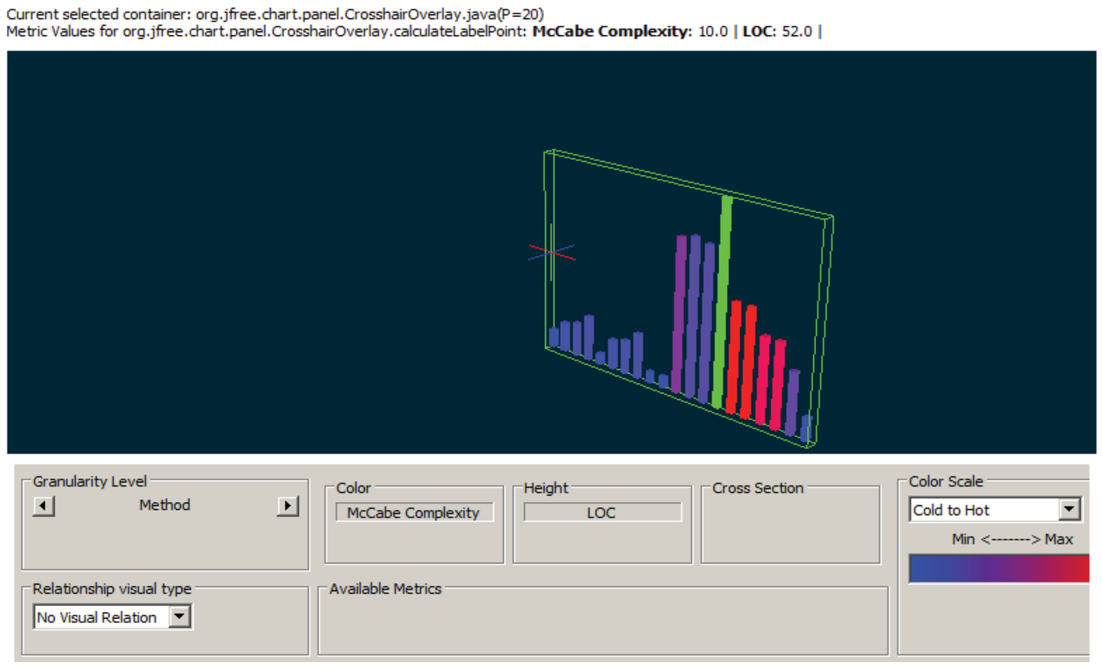
Lines of codes (LOC) is used to find out the size of the source code by counting the number of lines in the code. In SeeIT 3D, LOC metric counts the lines of codes with respect to polycylinders that were visualized in the visualization area.

Lack of Cohesion (LCOM) is the measure of the functionality of the module or method in a class. In SeeIT 3D, LCOM measures the lack of cohesiveness at the class level, this doesn't appear in method or line level.

Mc Cabe's complexity metric measures the number of independent paths in the source code and find the complexity. In SeeIT 3D, it provides the average of the linear independent paths in the source code for methods and classes in a package.

Control Structure measures how many control structures (If, while, for, else, none) are present in the code.

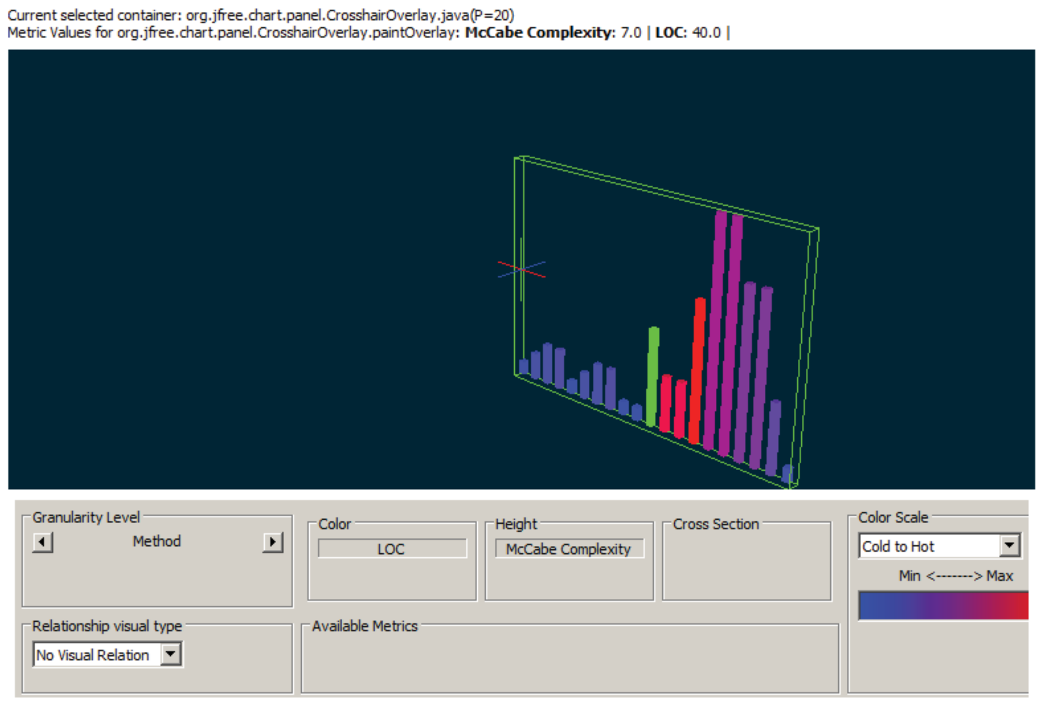
SeeIT 3D has 3 visual properties: *color, height and cross section.* Metric mapping can be done by mapping the metric values with the visual properties. This can be done by dragging the metric values into the visual property to get desired visualization in the visualization area. Screen shots of metric mapping are clearly shown in the figure below. In the figure below, Mc Cabe complexity is mapped with visual property color and LOC metric is mapped to height.


Current selected container: org.jfree.chart.panel.CrosshairOverlay.java(P=20)
Metric Values for org.jfree.chart.panel.CrosshairOverlay.calculateLabelPoint: **McCabe Complexity:** 10.0 | **LOC:** 52.0 |

Granularity Level: Method
Color: McCabe Complexity
Height: LOC
Cross Section:
Color Scale: Cold to Hot
Relationship visual type: No Visual Relation
Available Metrics

Figure 3. Metric mapping: Mc Cabe's complexity with Color and LOC with Height

After the drag and drop actions which cause a change in metric mapping is shown in the figure below which shows the Mc Cabe Complexity is mapped to height and LOC is mapped to color.

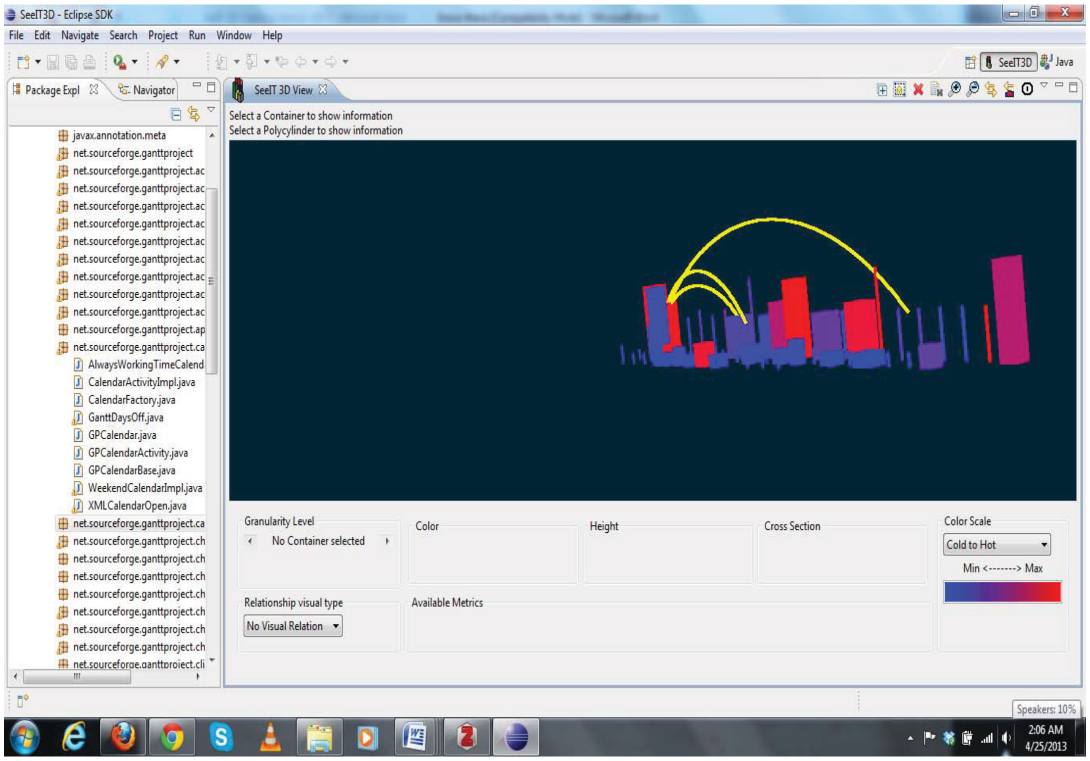

Current selected container: org.jfree.chart.panel.CrosshairOverlay.java(P=20)
Metric Values for org.jfree.chart.panel.CrosshairOverlay.paintOverlay: **McCabe Complexity:** 7.0 | **LOC:** 40.0 |

Granularity Level: Method
Color: LOC
Height: McCabe Complexity
Cross Section
Color Scale: Cold to Hot
Min <-------> Max
Relationship visual type: No Visual Relation
Available Metrics

Figure 4. Metric mapping change: Mc Cabe's complexity with Height and LOC with Color

3.3 Representing Relationships between Containers

The visualization also shows the relationships between the containers that are selected, these relationships are shown by using the visual relationship types. In SeeIT 3D there are four types of visual relationship types which are situated in the user customizable area. *Common base, lines, arcs and movements*. Default value of the visualization type is no visualization. These types are shown under the containers when they are selected. The figure below shows packages connected via arcs. A package is connected if there is an import to another package in the import clause of the program.



Select a Container to show information
Select a Polycylinder to show information

Granularity Level: No Container selected

Relationship visual type: No Visual Relation

Color Scale: Cold to Hot

Figure 5. Visual relationship type arcs.

3.4 User interactions

The figure below shows the newer version of the SeeIT 3D layout after incorporating changes following the observational study done by Gadapa in 2012 (Gadapa 2012).

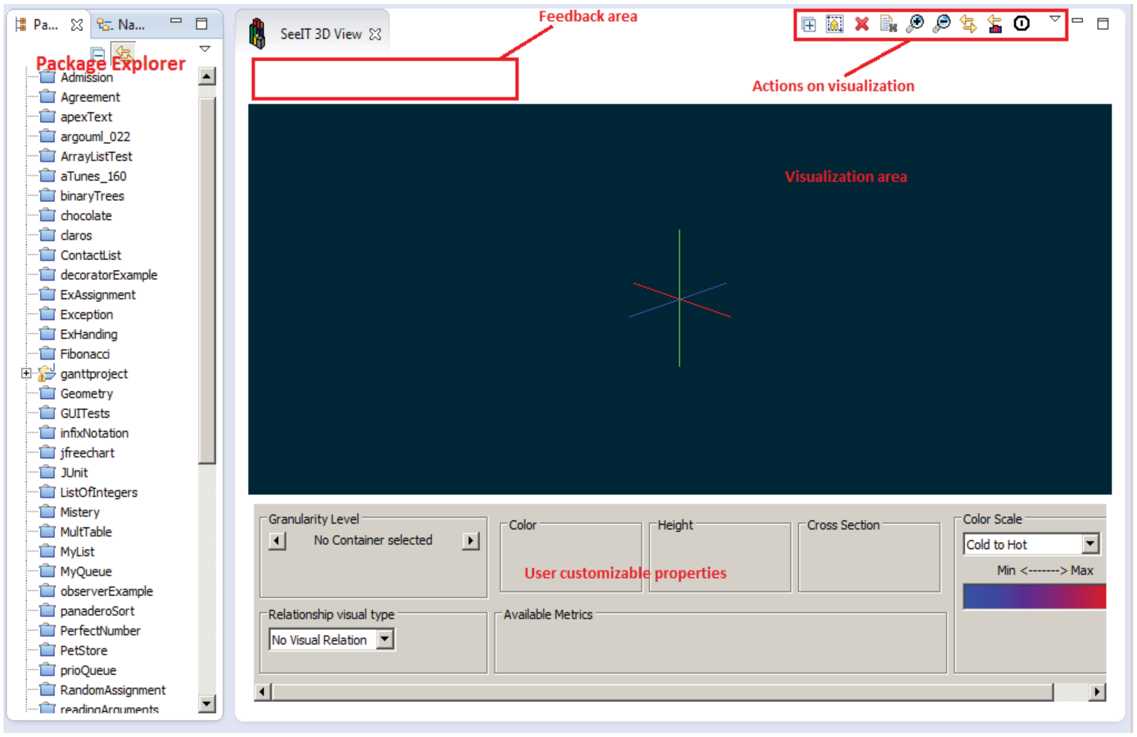
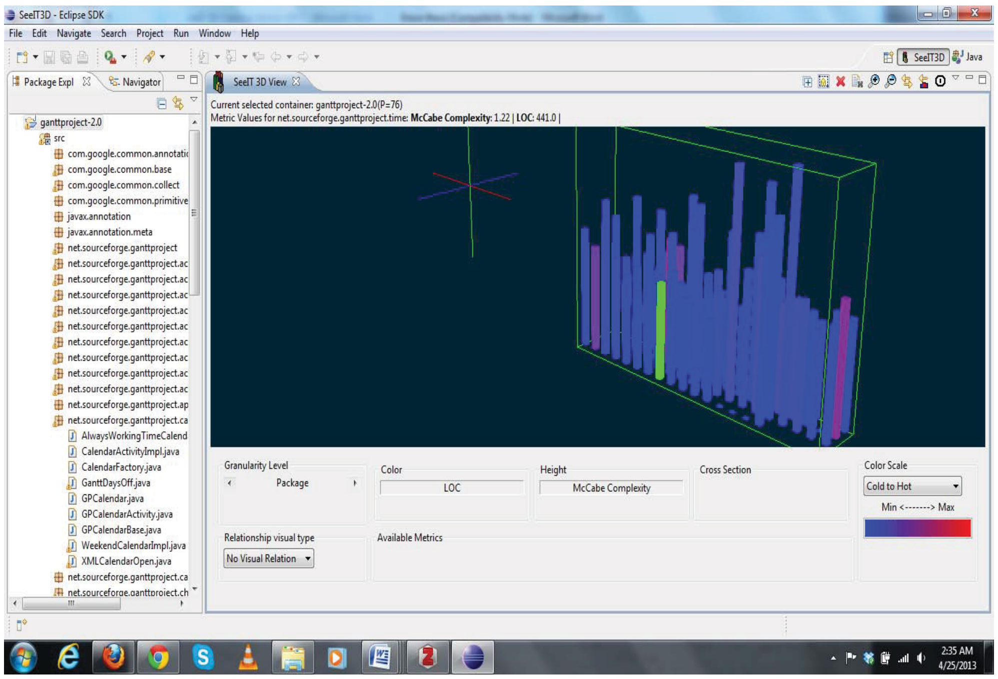


Figure 6. General Layout Pattern of SeeIT 3D

The package explorer contains the project systems. Visualization of project files are shown in the visualization area and for the polycylinders that were selected in the visualization area, information is shown in the feedback area. The feedback area shows the package/class/method that is selected as well as the number of package/class/method artifacts in the visualization area including metric values. The action icons on the top right allows changes to be made on the visualization like zooming, scaling, translating, resetting the visualization, deleting the containers. Refer to Appendix A.4 for a detailed description of the plug-in. It is a tutorial that was used by participants in our study to get familiar with the tool.

3.5 Visualizing GanttProject in SeeIT 3D

GanttProject is used as the subject system in our empirical study. This section shows some visualization on GanttProject using SeeIT 3D. The project can be visualized at the package level, class level, method level and line level. At the package level the entire system is visualized.

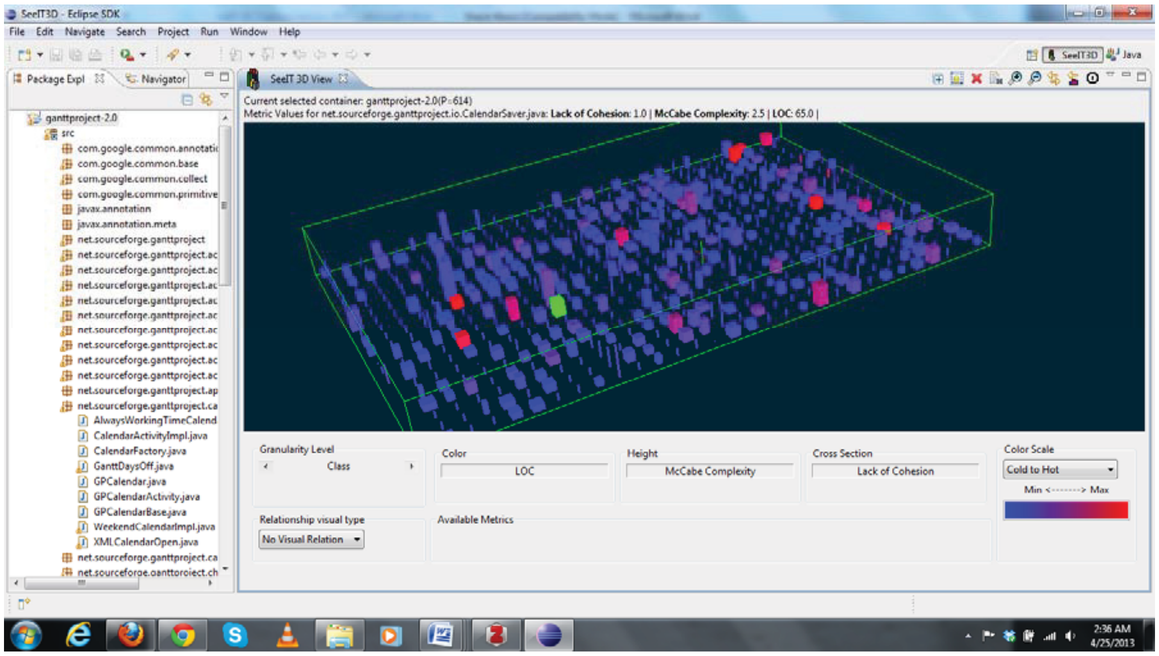


Current selected container: ganttproject-2.0(P=76)
Metric Values for net.sourceforge.ganttproject.time: McCabe Complexity: 1.22 | LOC: 441.0 |

Granularity Level: Package
Color: LOC
Height: McCabe Complexity
Cross Section
Color Scale: Cold to Hot
Min <-------> Max
Relationship visual type: No Visual Relation
Available Metrics

Figure 7. Visualization of GanttProject with granularity level as Package

From the figure above we can see the visualization of the subject system GanttProject which shows all the packages in the project folder. By the changing the granularity level to class level visualization we get the visualization shown in the figure below.

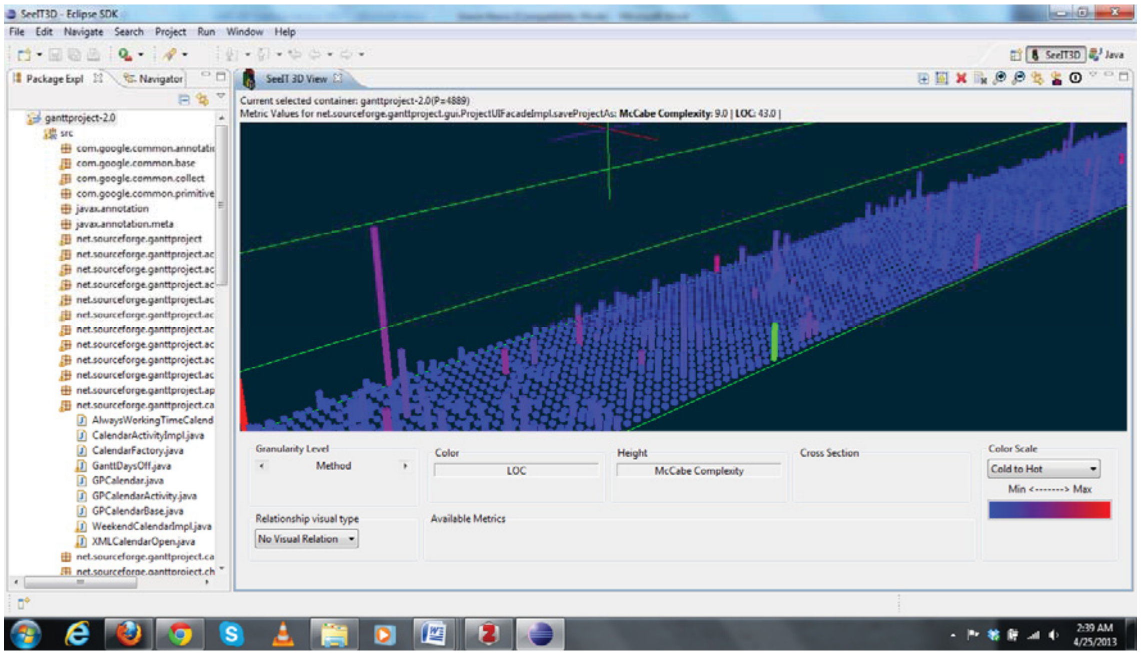


Current selected container: ganttproject-2.0(P=614)
Metric Values for net.sourceforge.ganttproject.io.CalendarSaver.java: **Lack of Cohesion:** 1.0 | **McCabe Complexity:** 2.5 | **LOC:** 65.0 |

Granularity Level: Class
Color: LOC
Height: McCabe Complexity
Cross Section: Lack of Cohesion
Color Scale: Cold to Hot
Min <-------> Max
Relationship visual type: No Visual Relation
Available Metrics

Figure 8. Visualization of GanttProject with granularity level as class.

If we change the level to method we get the visualization below.



Current selected container: ganttproject-2.0(P=4889)
Metric Values for net.sourceforge.ganttproject.gui.ProjectUIFacadeImpl.saveProjectAs: **McCabe Complexity:** 9.0 | **LOC:** 43.0 |

Granularity Level: Method
Color: LOC
Height: McCabe Complexity
Cross Section
Color Scale: Cold to Hot
Min <-------> Max
Relationship visual type: No Visual Relation
Available Metrics

Figure 9. Visualization of GanttProject with granularity level as Method

CHAPTER 4

The Empirical Study

This chapter presents the details of the empirical study conducted as part of this thesis. It gives details on the experiment design, hypotheses, data collection, tasks, and participiants and how the study was instrumented.

4.1 Experiment Design

The goal definition template by Wohlin et al. (Wohlin, Runeson et al. 1999) is used to describe the experiment. The experiment seeks to *analyze* a software visualization tool (SeeIT 3D) *for the purpose of* evaluating it's impact on solving overview, new feature and bug fixing tasks *with respect to* effectiveness (accuracy), efficiency (time), and visual effort *from the point of view of* the researcher *in the context of* students at three different universities.

An overview of the experiment is shown below. The main factor being analyzed is the usage of the used. A between-subjects design was used, where each subject was tested on either SeeIT 3D or No SeeIT 3D but not both. There are three dependent variables: accuracy, time, and visual effort. The data collection was done via online quetsionnaires and for one of the university in particular Youngstown State University, an eye tracker was used to track 20 participant's eye movements while they did the tasks.

Table 1. Experiment overview

Goal	Study the effect of the SeeIT 3D tool in the context of six tasks: overview, new feature, and bug finding tasks
Main Factor	Tool (SeeIT 3D, No SeeIT 3D)
Dependent variables	Accuracy, time, visual effort
Secondary factor	Expertise (novices and experts)

4.2 Hypotheses

Based on the research questions presented above in Section 1.3, four detailed null hypotheses based on each of the three dependent variables are given below.

H_a: There is no significant difference in *accuracy* between SeeIT 3D and No SeeIT 3D for overview, new feature, and bug finding tasks.

H_t: There is no significant difference in *time* between SeeIT 3D and No SeeIT 3D for overview, new feature, and bug finding tasks.

H_{ve}: There is no significant difference in *visual effort* between SeeIT 3D and No SeeIT 3D for overview, new feature, and bug finding tasks.

H_e: There is no significant difference between novices and experts in terms of accuracy, time, or visual effort, for SeeIT 3D and No SeeIT 3D users for overview, new feature, and bug finding tasks.

Alternative Hypotheses: There is a significant difference in accuracy, time, and, visual effort when SeeIT 3D is used for overview, new feature, and bug finding tasks.

4.3 Tasks

The experiment involves the comparison of using SeeIT 3D vs. not using SeeIT 3D with respect to six software tasks. The tasks fell into one of three task categories: overview, new feature and find bugs. The tasks were randomized when shown to the subject to avoid any learning effects that might occur. It is important to note that the tasks chosen were directly from the google code issue tracker for GanttProject. We chose bugs that were already fixed in a newer version so we were able to accurately score these tasks based on the patch files submitted for these tasks.

See **Table 2** for an overview of the tasks used in the study. The complete set of study questions including all background questions and post questionnaires can be found in Appendix A.2.

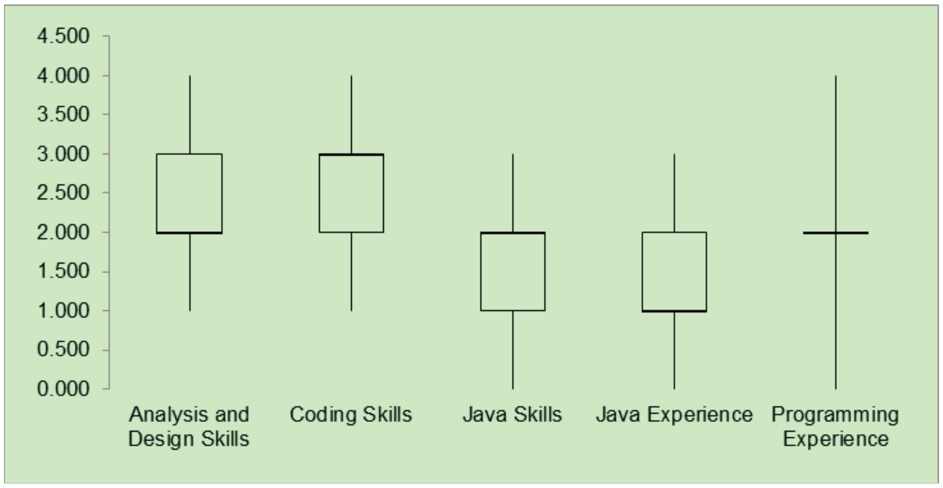
This study has two bug finding tasks, one new feature task and three overview tasks. The approximate time needed was 4, 3 and 12 minutes respectively. This was based a trial run by one volunteer.

Table 2. Overview of tasks and used in the study.

ID	Type	Brief Description	Approx. Time (with plug-in)	Difficulty
B3	Bug Find (issue 364)	Poor Print Quality	3 minutes	Easy
B4	Bug Find (issue 4)	Wrong update of dependencies on subtask duration change	4 minutes	Difficult
N6	New Feature (issue 27)	To be able to import data from basically any source, we need to have the ability of importing a CSV file with some more details than only tasks name.	2 minutes	Average
O8	Overview	Find the largest method in terms of lines of code (LOC) in package	3 minutes	Easy
O9	Overview	How many packages are connected, Which of these packages has the largest number of classes?	3 minutes	Difficult
O10	Overview	Number of methods in the largest class (in terms of LOC)	2 minutes	Average

4.4 Participants

There were ninety-seven volunteers from three different universities that participated in the study. Each participant was asked to fill out a background questionnaire before they took the study. See the Appendix for the entire background questionnaire used. Figure 10 shows demographics of participants. They were asked to rate their analysis and design skills, coding skills, java skills, java experience, and programming skills.



Analysis and Design Skills	Coding Skills	Java Skills	Java Experience
below average = 1	below average = 1	I don't know = 0	None = 0
average = 2	average = 2	Beginner = 1	Between 1 and 2 = 1
above average = 3	above average = 3	Intermediate = 2	Between 3 and 5 = 2
excellent = 4	excellent = 4	Advanced = 3	Between 6 and 10 = 3
			More than 10 = 4

Figure 10. Descriptive statistics on background questionnaire

Based on their rating, we classified them as experts or novices. The table below shows the split of participants into groups. Participants were randomly placed into the SeeIT 3D or No SeeIT 3D group. We did maintain a balance between the groups with respect to expertise as much as possible. The subjects were not aware of the experiment's hypotheses. The subjects in the No SeeIT 3D group only used Eclipse's Java code editor and other Eclipse features (no plug-ins) to solve the tasks.

Table 3. Groups Used in the Experiment.

	Experts	**Novices**	**Total**
SeeIT 3D	25	23	**48**
No SeeIT 3D	26	23	**49**
Total	**51**	**46**	**97**

Twenty students (10 experts and 10 novices) participated from YSU. Sixty-six students (34 experts and 32 novices) participated from University A and eleven students (7 experts and 4 novices) participated from University B. Majority of the subjects were in the Computer Science program. Some were in the Math and Electrical Engineering program.

4.5 Data collection

All subjects answered the six tasks via an online questionnaire. We used qualtrics.com to create the entire study online. We used the randomized blocks feature in qualtrics to randomly shuffle the questions to avoid learning effects. Each question was timed online. The subjects had to type the answer in the space provided in the online forms after they finished each task.

In addition to the online questionnaires, we collected eye tracking data and audio/video recordings of subjects that did the study at Youngstown State University because we have access to an eye tracker at this location. Twenty subjects fell into this category. The other two locations did not use this method of data collection. We did obtain IRB approval and training before we began this study.

4.6 Eye-Tracking Apparatus

The Tobii X60 eye tracker (www.tobii.com) was used in this study at one location primarily at YSU. It is a 60Hz video-based binocular remote eye tracker that does not require the user to wear any head gear. It generates 60 samples of eye data per second. The average accuracy for the Tobii eye tracker is 0.5 degrees which averages to about 15 pixels. The eye tracker compensates for head movement during the study. The study was conducted on a 24 inch monitor with screen resolution set at 1920 * 1080. The study was configured to use a dual monitor extended desktop setting. The first monitor was used by the experimenter to setup and initiate the study. The eye tracker records eye-gaze data and audio/video recordings of the entire study session on the second monitor. The eye gaze data includes timestamps, gaze positions, fixations and their durations, pupil sizes, and validity codes. In this study, only fixations and their durations are used to measure visual effort. Tobii Studio was used.

4.7 Subject System

We used an open source system GanttProject version 2.5.4 (http://www.ganttproject.biz) as our subject system. Since this was a between subjects experiment we were able to use one subject system for the both groups (SeeIT 3D and No SeeIT 3D). GanttProject is a Java based project management software for project scheduling that runs under Windows, Linux, and Mac OS X operating systems. Gantt Project lets you easily break down a project into tasks, show dependencies, and manage resources. It features most basic project management functions like a Gantt chart for project scheduling of tasks, and doing resource management using resource load charts. It does not have advanced features like cost accounting, message and document control. It has a number of reporting options (MS Project, HTML, PDF, spreadsheets). GanttProject has approximately 614 classes, 76 packages, 4889 methods and 60344 LOC.

4.8 Conducting the Study

The process started with the background questionnaire that all subjects were required to fill out. Based on this, we split the participants into two groups balancing the expertise level in each group. This was done at least a week prior to the actual study.

A couple of days before the study, the SeeIT 3D group was asked to do a tutorial to familiarize themselves with the SeeIT 3D tool (see Appendix A.4. for the tutorial used). They had to turn in the answers to three challenge questions at the end of the tutorial. Both the groups were also asked to view a GanttProject video (15 minutes) available on the GanttProject website (http://www.ganttproject.biz). The viewing of the video was optional. Instead they could just read a one page description of GanttProject prior to the study.

When the subject came in on the day of the actual study, they were first asked to read and sign the informed consent form to give us persmission to record their eye movements, audio, and video. See Figure 11 for the workspace used during the study at Youngstown State University. They were then positioned in front of the right monitor at an appropriate distance from the eye tracker. Their eyes were first calibration in a short 45 second calibration session. After that they began the study. During the study, subjects had to context switch between qualtrics.com and eclipse to answer questions.

Throughout the study, voice, user's face and eye movement were recorded. These recordings act as evidence and references to evaluate the study results.

Subjects were not allowed to search the Internet for answers or to go back and fix an answer. The back button was disabled. Eclipse Juno was used without any additional plug-ins installed. The Classic SDK was installed.

At the end of the study, they filled out a short post-questionnaire (see Appendix for questions). The post questionnaire for the SeeIT 3D group had additional questions asking about their experience with the tool.

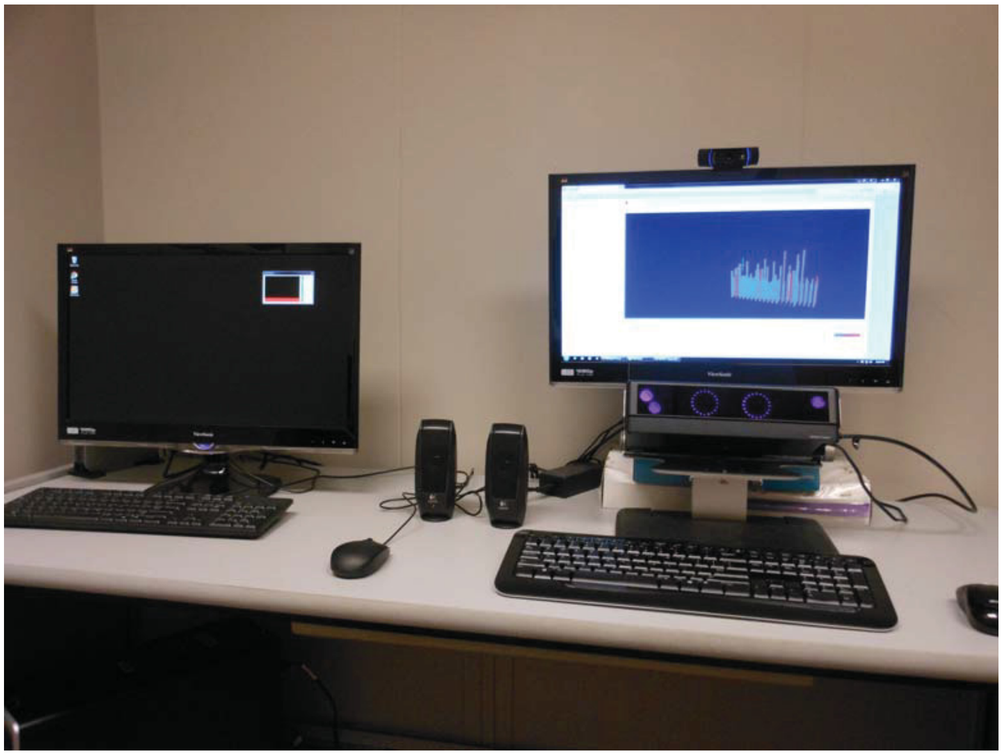


Figure 11. Work space of a person participating in the study. The screen on the left is for the experimentor, the right screen is used by the subject. The eye tracker Tobii X60 is seen at the base of the right screen.

Participants in the SeeIT group were not forced to use the plug-in, however they were instructed to use the plug-in whenever they think it is useful for the task at hand. The study set up includes Window 7 operating system with 64 GB hard disk, Tobii Eye-tracker, Snow ball voice recorder, and Logitech webcam and speakers.

CHAPTER 5

Results and Analyses

This chapter presents the results from our controlled experiment. The linear mixed models regression model is fit to the data to determine significance. Alpha is set at 0.05 that determines significance with a 5% error.

5.1 SeeIT 3D Tutorial Results

We scored the SeeIT 3D tutorial and found that twenty two subjects got a perfect score of 10. Eight people scored 9. The rest (13) of the scores were between 3 and 8. The mean score was 8.6 for subjects that took the tutorial. Overall, everyone in the SeeIT 3D group had an idea of the features provided by SeeIT 3D and how to use it.

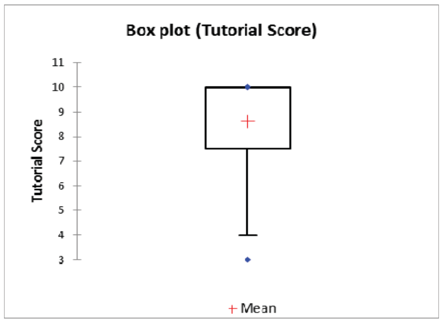


Figure 12. Descriptive statistics for the tutorial score

5.2 Accuracy

The results were scored based on patches and comments given in the google code issue tracker for the bug and new feature tasks. The descriptive statistics for accuracy is given in **Figure 13**. We generated several models for accuracy for each of the task categories and present them below in **Table 5**. **Table 4** summarizes the p-values showing that we only found the overview tasks to score significantly higher in SeeIT 3D. In this case, we can only reject the null hypothesis H_a for the *overview* tasks. No significant difference was found for the new feature and bug finding tasks.

Table 4. p-values for Accuracy split by task category

Task Category	**p-value**
Overview	<0.0001 *
New Feature	0.939
Bug Fix	0.128
Total	**0.007 ***

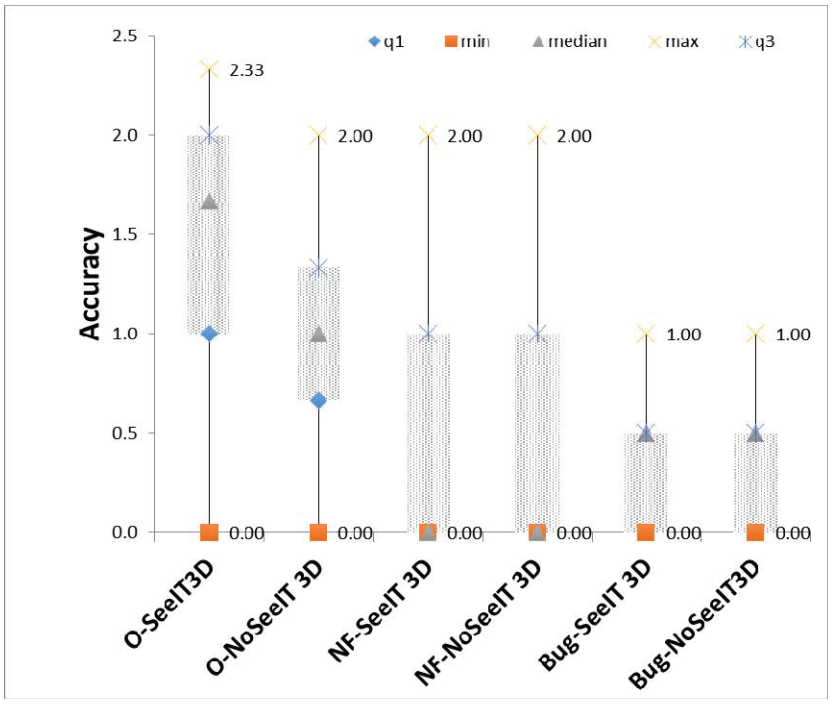


Figure 13. Descriptive statistics for Accuracy across groups and task categories.

Table 5. Model for accuracy (score) of the three task categories

Model parameters (Variable Bug-score):

Source	Value	Standard error	t	Pr > |t|	Lower bound (95%)	Upper bound (95%)
Intercept	0.418	0.044	9.558	< 0.0001	0.331	0.505
Group-No SeeIT 3D	0.000					
Group-SeeIT 3D	-0.095	0.062	-1.534	**0.128**	-0.219	0.028

Model parameters (Variable O-score):

Source	Value	Standard error	t	Pr > |t|	Lower bound (95%)	Upper bound (95%)
Intercept	0.986	0.085	11.659	< 0.0001	0.818	1.154
Group-No SeeIT 3D	0.000					
Group-SeeIT 3D	0.521	0.120	4.328	**< 0.0001**	0.282	0.759

Model parameters (Variable NewFeat-Score):

Source	Value	Standard error	t	Pr > |t|	Lower bound (95%)	Upper bound (95%)
Intercept	0.531	0.102	5.219	< 0.0001	0.329	0.732
Group-No SeeIT 3D	0.000					
Group-SeeIT 3D	0.011	0.145	0.076	0.939	-0.276	0.298

We also observed that for accuracy, the average bug score was higher for the No SeeIT 3D group. The average new feature score was higher for the SeeIT 3D group. As mentioned above, the only significant difference was in the overview score that was higher for the SeeIT 3D group. This means that SeeIT 3D gave more accurate answers than the No SeeIT 3D group for overview tasks.

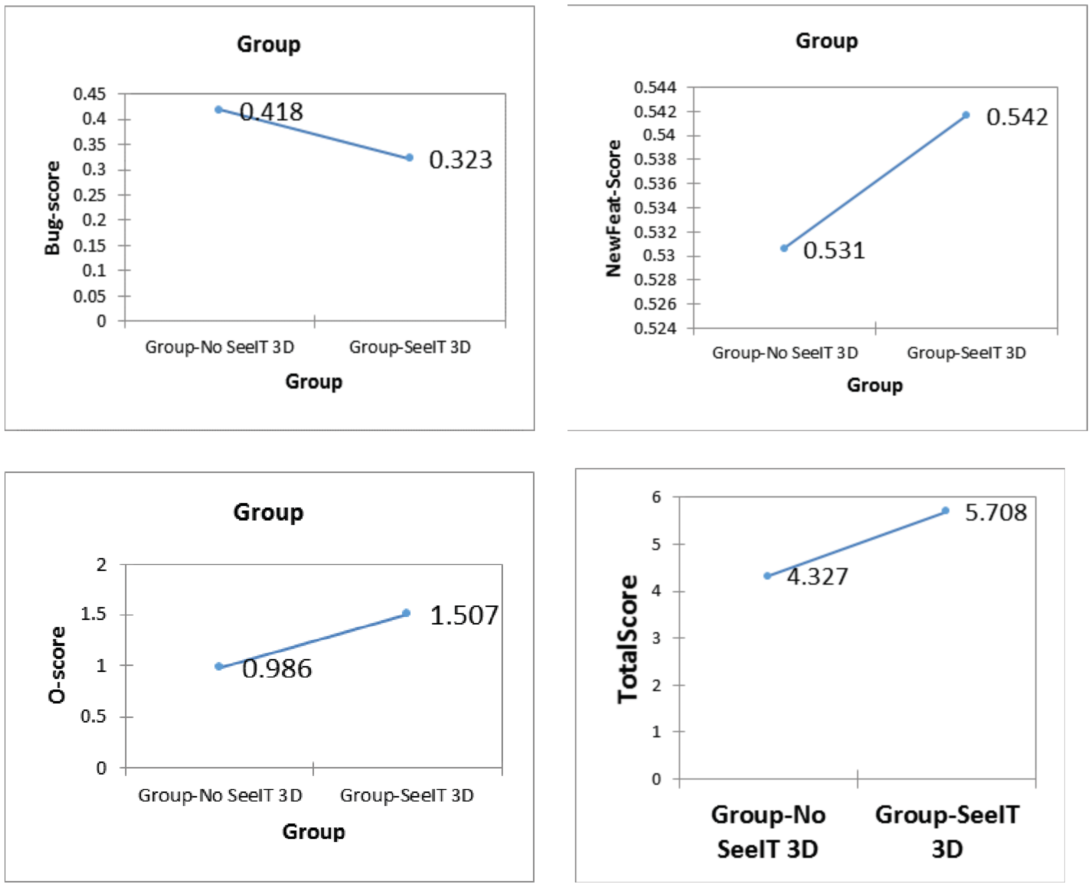


Figure 14. Average scores for each task category between the groups.

The above figure plots the means of each group showing the difference in means in each group for each task category.

5.3 Time

Each question was timed via the online questionnaire. The descriptive statistics for time is given in **Figure 15**. We generated several models for time for each of the task categories and present them below. **Table 6** summarizes the p-values showing that we only found the overview tasks to take significantly less time in SeeIT 3D. In this case, we can only reject the null hypothesis H_t for the *overview* tasks. No significant difference was found for the new feature task. We found the opposite effect for the bug fix tasks. The SeeIT 3D group took significantly longer for bug finding tasks (p-value = 0.036). This was an interesting results but not too surprising. We had expected the tool to do well in overview tasks. We did not know how the tool would perform with the other task categories so this was an interesting find.

Table 6. p-values for Time split by task category

Task Category	**p-value**
Overview	0.00021 * (seeit3d is faster)
New Feature	0.438
Bug Fix	0.036 * (seeit3d is slower)
Total	**0.624**

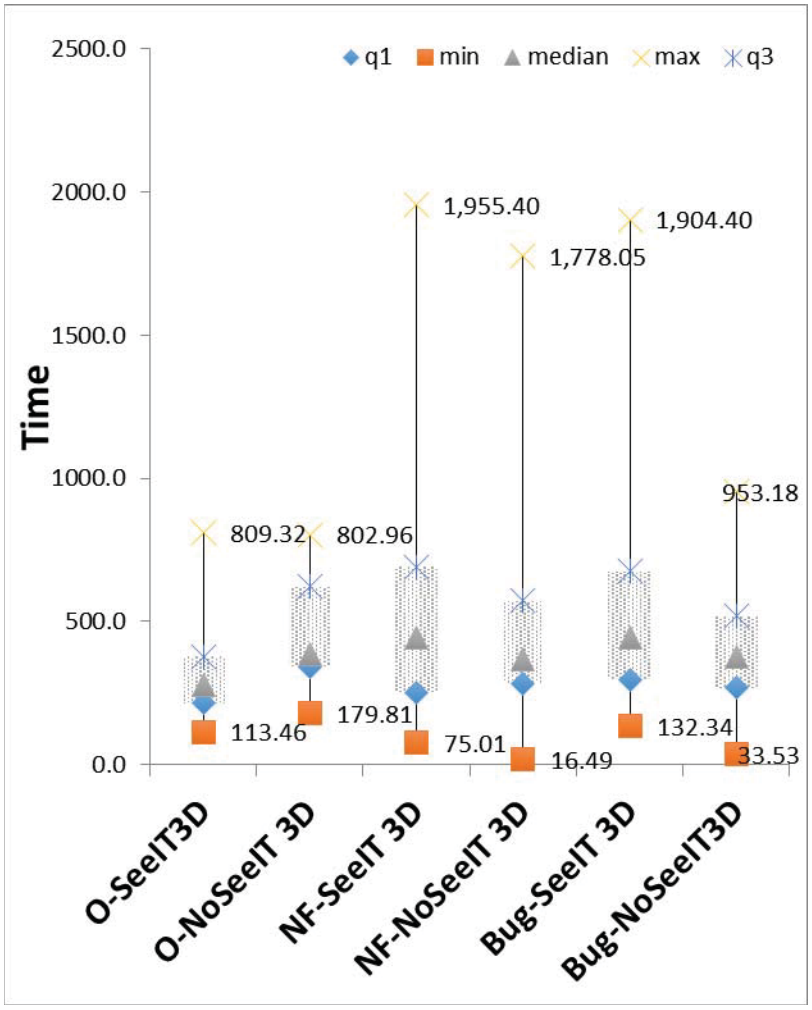


Figure 15. Descriptive statistics for Time across groups and task categories.

Table 7 shows the model parameters after the linear mixed models regression was run on the data.

Table 7. Model for accuracy (score) of the three task categories including total

Model parameters (Variable Bug-Time):

Source	Value	Standard error	t	Pr > |t|	Lower bound (95%)	Upper bound (95%)
Intercept	401.489	39.578	10.144	< 0.0001	322.938	480.041
Group-No SeeIT 3D	0.000					
Group-SeeIT 3D	119.590	56.263	2.126	0.036	7.924	231.256

Model parameters (Variable NewFeat-Time):

Source	Value	Standard error	t	Pr > |t|	Lower bound (95%)	Upper bound (95%)
Intercept	465.150	50.514	9.208	< 0.0001	364.894	565.407
Group-No SeeIT 3D	0.000					
Group-SeeIT 3D	55.935	71.809	0.779	0.438	-86.586	198.456

Model parameters (Variable O-Time):

Source	Value	Standard error	t	Pr > |t|	Lower bound (95%)	Upper bound (95%)
Intercept	458.984	24.216	18.954	< 0.0001	410.922	507.046
Group-No SeeIT 3D	0.000					
Group-SeeIT 3D	-132.512	34.424	-3.849	0.00021	-200.834	-64.189

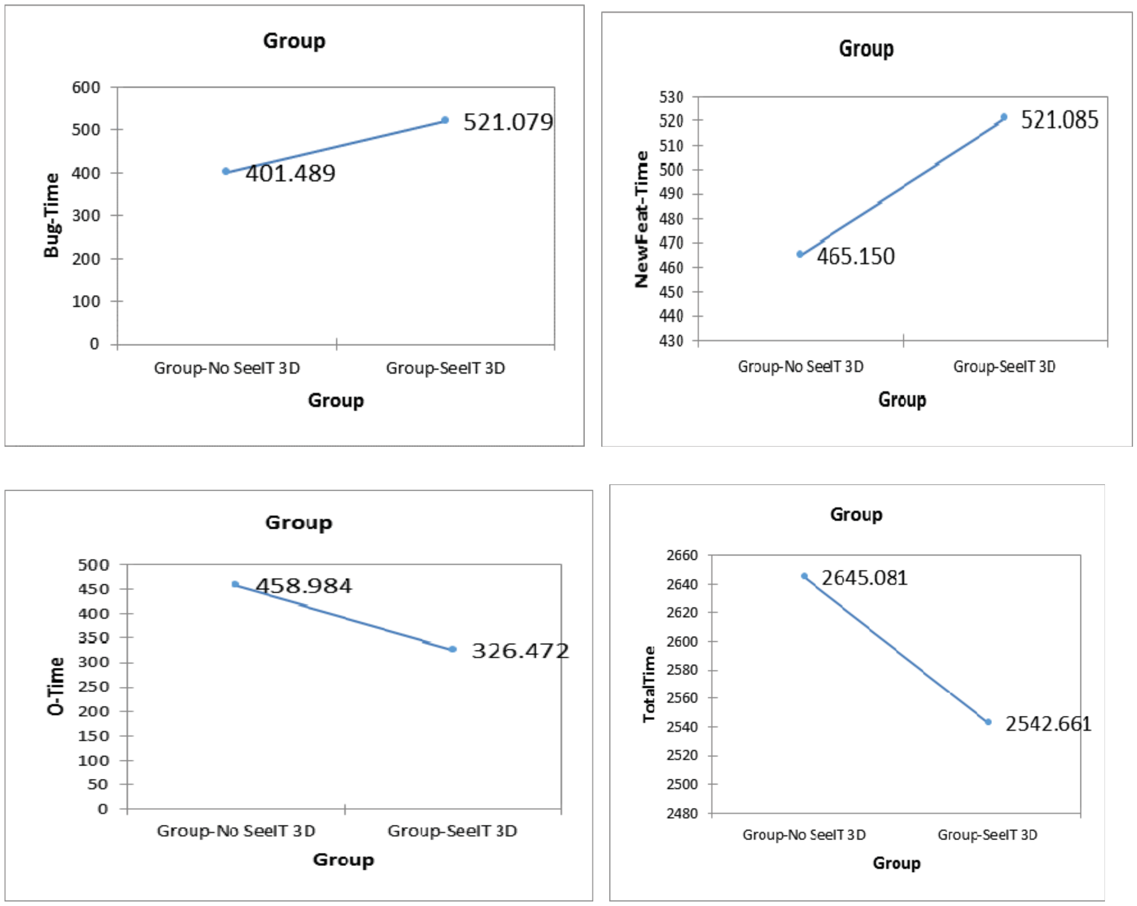


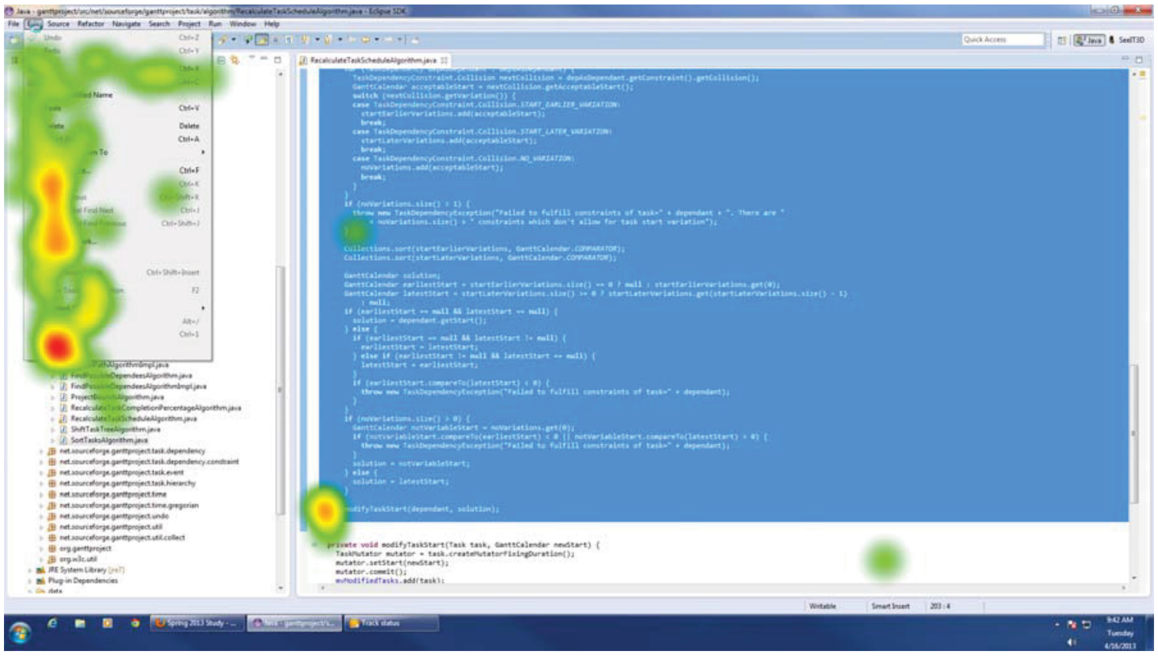
Figure 16. Average time for each task category between the groups.

The above charts plot the mean time for each group across each task category. With respect to total time, The No SeeIT 3D group took longer. But as mentioned earlier, we found that the SeeIT 3D group took significantly longer for bug finding tasks.

5.4 Visual Effort

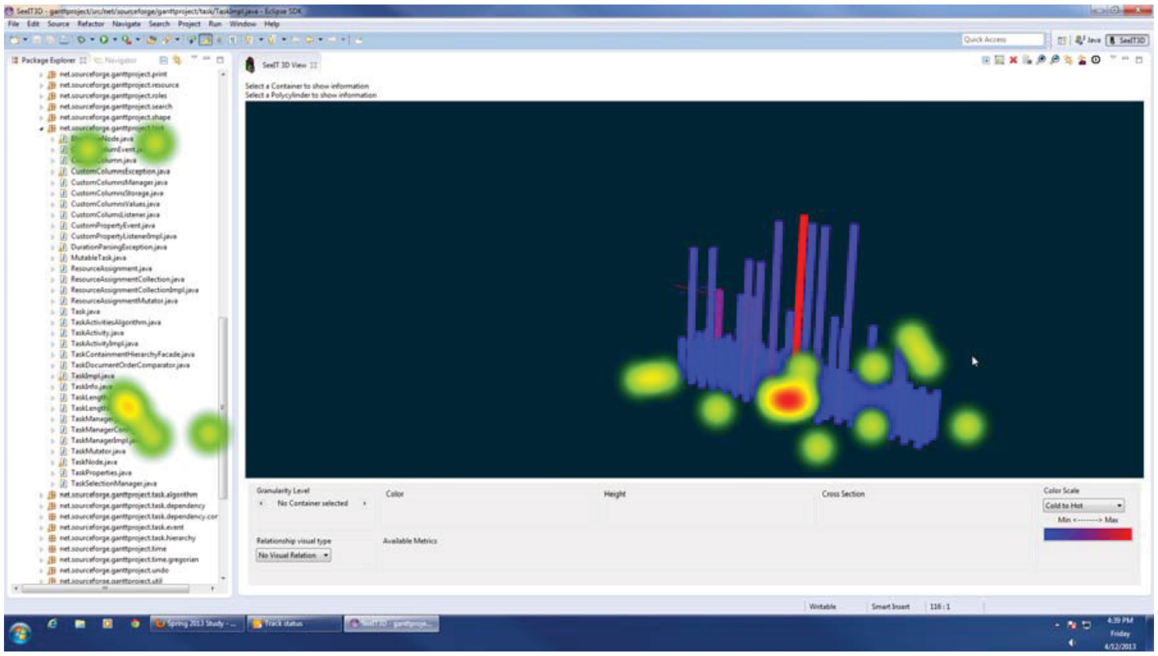
Thte visual effort analysis was conducted on only twenty subjects that participated at Youngstown State University. We present a qualitative analysis of the findings here. We take a look at fixation counts and fixation durations to determine visual effort. There is no significant difference with respect to fixation duration, fixation counts reported by the linear mixed effects regression model. So we are unable to reject the H_{ve} hypothesis.

In **Figure 17** below, we see the heatmap of an expert without the SeeIT 3D tool. This task asked to find the number of LOC of a particular method. The subject was trying to find this feature in a menu option but couldn't find it. In **Figure 18** however the expert subject is able to quickly find this information by clicking on the red poly cylinder in the visualization.



```
    TaskDependencyConstraint.Collision nextCollision = depAsDependant.getConstraint().getCollision();
    GanttCalendar acceptableStart = nextCollision.getAcceptableStart();
    switch (nextCollision.getVariation()) {
    case TaskDependencyConstraint.Collision.START_EARLIER_VARIATION:
      startEarlierVariations.add(acceptableStart);
      break;
    case TaskDependencyConstraint.Collision.START_LATER_VARIATION:
      startLaterVariations.add(acceptableStart);
      break;
    case TaskDependencyConstraint.Collision.NO_VARIATION:
      noVariations.add(acceptableStart);
      break;
    }
  }
  if (noVariations.size() > 1) {
    throw new TaskDependencyException("Failed to fulfill constraints of task=" + dependant + ". There are "
        + noVariations.size() + " constraints which don't allow for task start variation");
  }

  Collections.sort(startEarlierVariations, GanttCalendar.COMPARATOR);
  Collections.sort(startLaterVariations, GanttCalendar.COMPARATOR);

  GanttCalendar solution;
  GanttCalendar earliestStart = startEarlierVariations.size() == 0 ? null : startEarlierVariations.get(0);
  GanttCalendar latestStart = startLaterVariations.size() >= 0 ? startLaterVariations.get(startLaterVariations.size() - 1)
      : null;
  if (earliestStart == null && latestStart == null) {
    solution = dependant.getStart();
  } else {
    if (earliestStart == null && latestStart != null) {
      earliestStart = latestStart;
    } else if (earliestStart != null && latestStart == null) {
      latestStart = earliestStart;
    }
    if (earliestStart.compareTo(latestStart) < 0) {
      throw new TaskDependencyException("Failed to fulfill constraints of task=" + dependant);
    }
  }
  if (noVariations.size() > 0) {
    GanttCalendar notVariableStart = noVariations.get(0);
    if (notVariableStart.compareTo(earliestStart) < 0 || notVariableStart.compareTo(latestStart) > 0) {
      throw new TaskDependencyException("Failed to fulfill constraints of task=" + dependant);
    }
    solution = notVariableStart;
  } else {
    solution = latestStart;
  }

  modifyTaskStart(dependant, solution);

private void modifyTaskStart(Task task, GanttCalendar newStart) {
  TaskMutator mutator = task.createMutatorFixingDuration();
  mutator.setStart(newStart);
  mutator.commit();
  myModifiedTasks.add(task);
```

Figure 17. Heatmap of an Expert with No SeeIT 3D



Granularity Level: No Container selected

Relationship visual type: No Visual Relation

Color Scale: Cold to Hot

Figure 18. Heatmap of an Expert with SeeIT 3D

Figure 19 shows the heatmap of a novice subject in the SeeIT 3D group. Their eye movements indicate the use of the action bar on the top right as well as the options available at the bottom of the visualization.

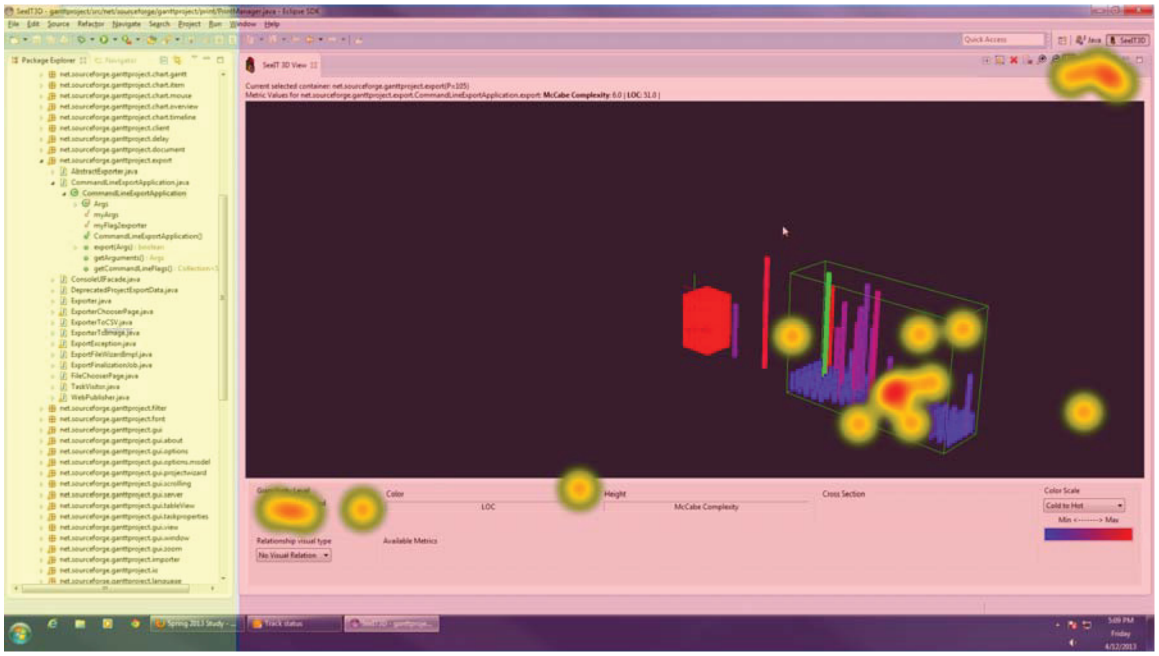


Figure 19. Heatmap of a Novice with SeeIT 3D

5.5 Secondary Interactions on Accuracy and Time

This section presents plots of any effecs that expertise may have on the accuracy variable. In general, we cannot reject the hypotheses H_e for all task categories.

In the No SeeIT 3D group, there was a significant difference in accuracy between novices and experts for the new feature task.

There is a significant difference between experts and novices with respect to total accuracy of tasks (p-value = 0.025). A significant difference in time was found between the three universities (p-value< 0.001).

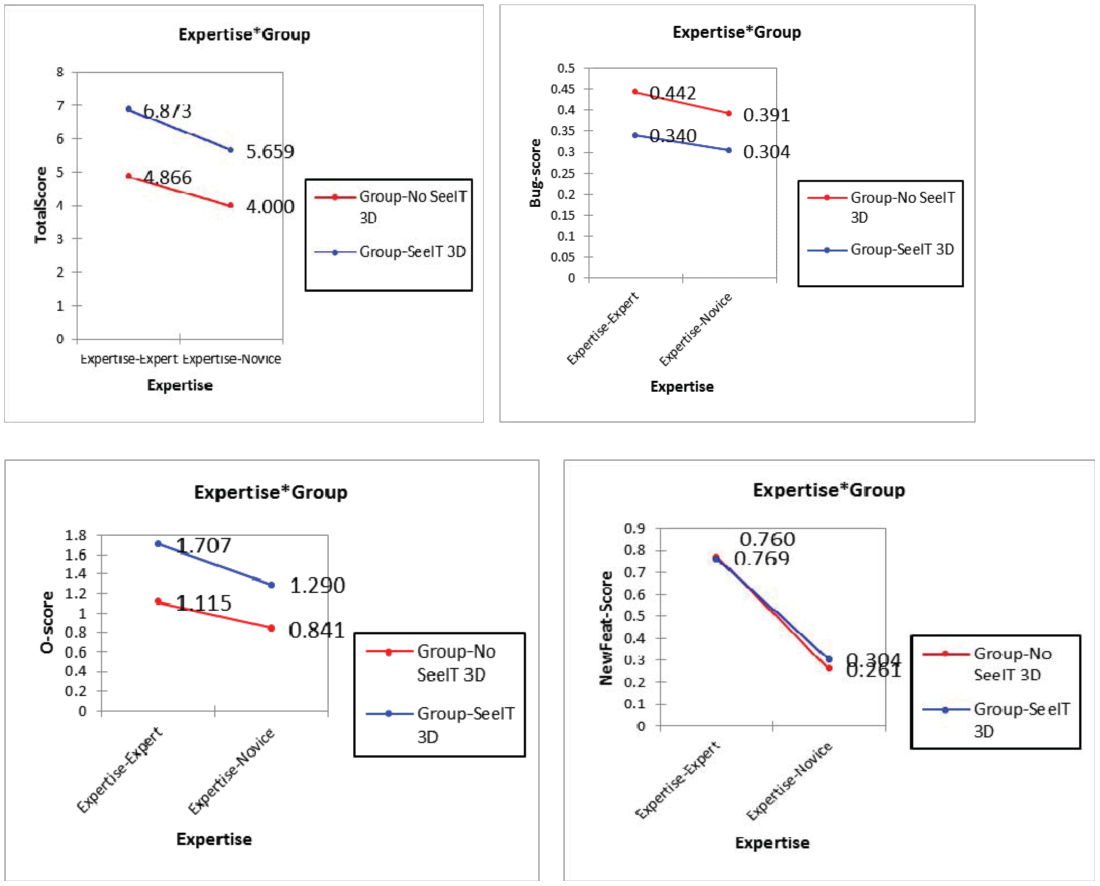


Figure 20. Secondary Interactions on Accuracy

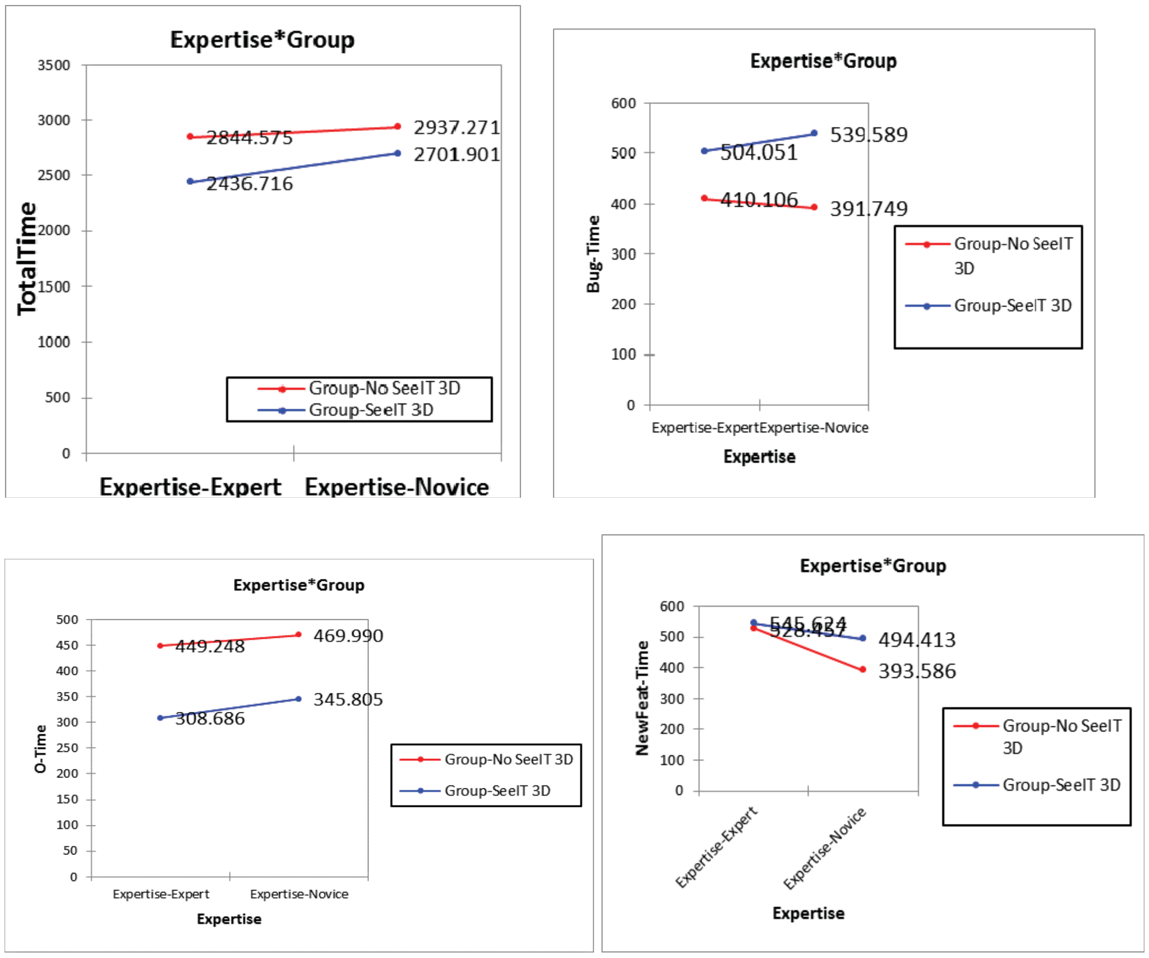
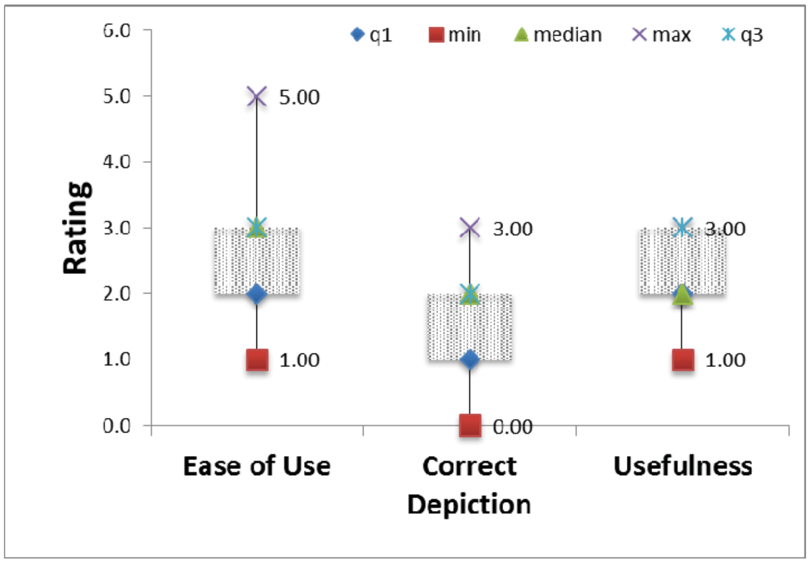


Figure 21. Secondary Interactions on Time

5.6 Post Questionnaire Results

The results of the post questionnaire are given below. None of the subjects were familiar with the design of Gantt Project. The questions were considered to be of average to difficult in terms of difficulty. The "correct depiction" measure determined if SeeIT 3D correctly depicted the items in the visualization. The median stated that the tool was somewhat useful.



Ease of Use: Very Easy = 1 Easy =2 I needed more time = 3 Somewhat Difficult = 4 Very Difficult = 5

Correct Depiction: Always = 3 Frequently = 2 Occasionally/Sometimes =1 Rarely = 0

Useful: Very useful =3 Somewhat useful =2 Somewhat not useful =1

Figure 22. Post Questionnaire Descriptive statistics

5.7 Subject Comments

Some of the quotes we received via comments are mentioned below.

- *Overall great tool. I wish I had the ability to zoom in the screen and not just for a specific package. There were a few times I had multiple packages and wanted to zoom all, but they would collide and it was hard to read from it.* (SeeIT 3D - expert user)
- *Had a difficult time finding where code was located without the use of diagrams or documentation.* (No SeeIT 3D - expert user)
- *I think this tool is the best for determining where the most code is in a given project and how different packages are related to each other. It allows you to*

visualize a large project in a consistent manner that is easy to understand. I would highly recommend this tool to software developers. (SeeIT 3D – expert user, advanced java level)

5.8 Threats to Validity

Every experiment is subject to various threats to validity. They are outlined below. We tried to avoid any learning effects by randomizing the tasks. Every user saw a different order of tasks. With respect to external validity, we can compare our subject pool to junior level developers for novices and mid-level developers for experts. The research participants did not know about the hypotheses used in the research. They only knew that they would participate in helping us understand how software visualization tools work to understand systems. During the study, there was minimal contact between the experimenter and the participants. The experimenter did not interact or direct the participants to complete the questions in one way or another. Since we had unbalanced groups we used the linear mixed effects regression model to determine significance.

CHAPTER 6

Conclusions and Future Work

A study was conducted to determine the usefulness and added benefit of SeeIT 3D, a software visualization tool realized as an Eclipse plug-in. There was a significant difference in accuracy for overview tasks between the SeeIT 3D and Non SeeIT 3D groups with the SeeIT 3D group performing better. The SeeIT 3D group took significantly less time for overview tasks but not for the new feature task. Surprisingly, we found that SeeIT 3D took significantly longer for bug finding tasks, a finding we did not anticipate in the beginning. There was a significant difference in accuracy between experts and novices with experts performing better. There were no significant differences to report with respect to visual effort (fixations and durations) for the twenty subjects in the eye-tracking subject pool. This could be due to the low sample size in the eye-tracking data category.

In future work, we plan to compile a list of suggestions in order of priority derived from comments we received from the post questionnaire and comments from answering each question. These will be used to collaborate with the SeeIT 3D tool developer to incorporate these suggestions into the tool to improve it thereby increasing its acceptance into developers working environments.

APPENDIX

Study Material

A.1. Background Questionnaire

First and Last Name: __________________________

Name of University: ___________________________

1. **Rate your software analysis and design skills.**
    a. Poor
    b. Below Average
    c. Average
    d. Above Average (Good)
    e. Excellent
2. **Rate your software coding skills.**
    a. Poor
    b. Below Average
    c. Average
    d. Above Average (Good)
    e. Excellent
3. **Rate your Java programming skills.**
    a. I don't know Java
    b. Beginner
    c. Intermediate
    d. Advanced
4. **Select years of experience in programming with Java.**
    a. None
    b. Between 1 and 2
    c. Between 3 and 5
    d. Between 6 and 10
    e. More than 10
5. **Select years of experience in programming with ANY language.**
    a. None.
    b. Between 1 and 2
    c. Between 3 and 5
    d. Between 6 and 10
    e. Above 10
6. **Which operating system do you work on for coding? (select all that apply)**
    a. Windows
    b. Linux

c. Mac
d. Other, please specify.

____

7. **Which IDE do you use for programming? (select all that apply)**
   a. I don't use an IDE
   b. Eclipse
   c. Netbeans
   d. Dr.Java
   e. Dev C++
   f. Other, please specify.

____

8. **If your answer above is Eclipse, how often do you use Eclipse for programming?**
   a. Occasionally /Sometimes
   b. Almost every time
   c. Every time

9. **Do you use any 2D or 3D visualization tools to visualize software** Yes No
   If Yes, please specify the tools:

   ____

10. **List languages you are able to program in.**

   ____

A.2. Main Study Questionnaire

Read the 1 page instructions given to you before you begin.
GanttProject Overview

In this study we will use the GanttProject system. You will be asked questions with respect to this system.

GanttProject is a Java based, project management software for project scheduling that runs under Windows, Linux, and Mac OS X operating systems. Ganttproject lets you easily break down a project into tasks, show dependencies, and manage resources.

It features most basic project management functions like a Gantt chart for project scheduling of tasks, and doing resource management using resource load charts. It does not have advanced features like cost accounting, message and document control. It has a number of reporting options (MS Project, HTML, PDF, spreadsheets).

The major features include:

- Gantt chart
    - Create work breakdown structure
    - Draw dependencies
    - Task hierarchy
    - Define milestones
- Resource load chart
    - Assign human resources to work on tasks
    - See their allocation on the resource load chart
- Generation of PERT chart from Gantt chart
- PDF, HTML, PNG reports
- MS Project import/export
- Exchange data with spreadsheet applications
- WebDAV based groupwork – share projects with colleagues

A screen shot of Gantt project is shown below.

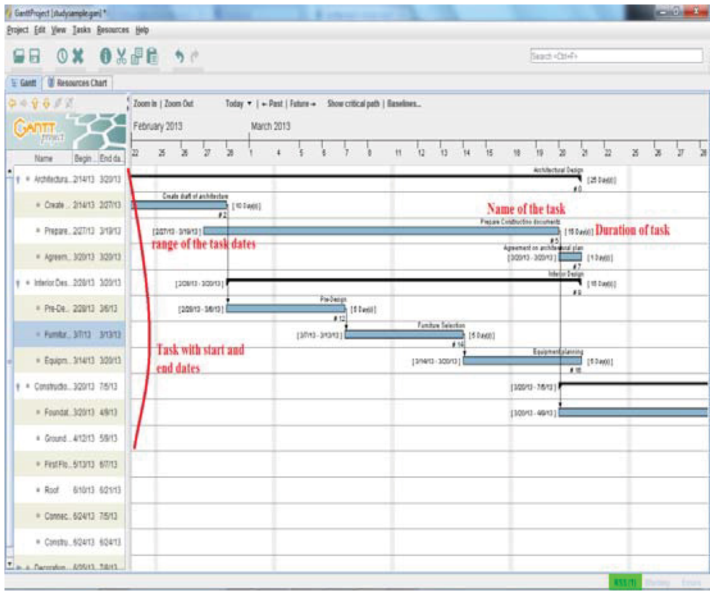


Fig a) Gantt project with tasks and dependencies

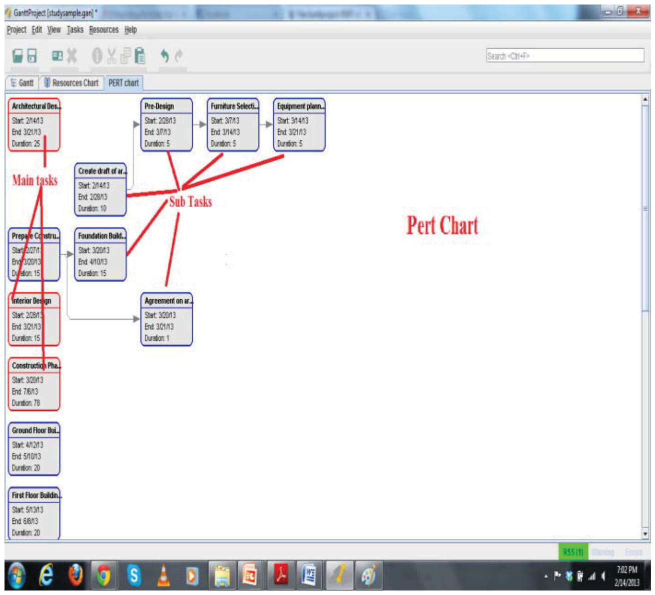


Fig b) Gantt Project for Pert Chart

You may also optionally see the 15 minute "Introduction to GanttProject" video. It will show you the various features available in GanttProject. The video is available at http://www.youtube.com/watch?v=5rHCSa5ad34

Please state your answer.

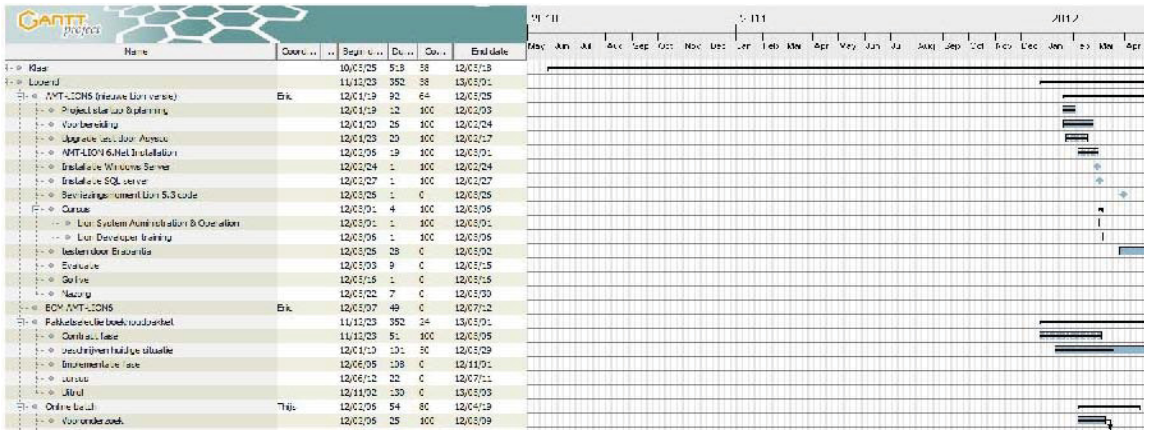
I read the GanttProject overview	Yes	No
I saw the GanttProject You tube video	Yes	No

Bug Fix Task: B3

You are given the following bug description.

> **Bug Title:** Poor Print Quality
>
> **Bug Description:** When the project is printed in print preview function (no zoom on target paper size A3) it is difficult to read the task list on the left side.
>
> See the figure below for an example of poor print quality.

Question: Which class(es) would need to be changed/added in order to fix this bug? Please enter the class(es) name(s) only (not the full qualified name).



Name	Coord...	..	Begin d...	Du...	Co...	End date
Klaar			10/05/25	518	58	12/05/18
Lopend			11/12/23	352	38	13/05/01
AMT-LIONS (nieuwe Lion versie)	Erik		12/01/19	92	64	12/05/25
Project startup & planning			12/01/19	12	100	12/02/03
Voorbereiding			12/01/20	25	100	12/02/24
Upgrade test door Aysela			12/01/23	20	100	12/02/17
AMT-LION 6.Net Installation			12/02/06	19	100	12/03/01
Installatie Windows Server			12/02/24	1	100	12/02/24
Installatie SQL server			12/02/27	1	100	12/02/27
Beveiligingsmoment Lion 5.3 code			12/03/25	1	0	12/03/25
Cursus			12/03/01	4	100	12/03/06
Lion System Administration & Operation			12/03/01	1	100	12/03/01
Lion Developer training			12/03/06	1	100	12/03/06
testen door Eraventis			12/03/26	28	0	12/05/02
Evaluatie			12/05/03	9	0	12/05/15
Go live			12/05/16	1	0	12/05/16
Nazorg			12/05/22	7	0	12/05/30
BCM AMT-LIONS	Erik		12/05/07	49	0	12/07/12
Pakketselectie boekhoudpakket			11/12/23	352	24	13/05/01
Contract fase			11/12/23	51	100	12/03/05
beschrijven huidige situatie			12/01/10	101	50	12/05/29
Implementatie fase			12/06/05	108	0	12/11/01
cursus			12/06/12	22	0	12/07/11
Uitrol			12/11/02	130	0	13/05/03
Online batch	Thijs		12/02/06	54	80	12/04/19
Vooronderzoek			12/02/06	25	100	12/03/09

Please give the rationale for your answer:

Difficulty Level:	Very easy	Easy	Difficult	Very difficult
Confidence:	Very confident	Confident	Somewhat confident	Not confident

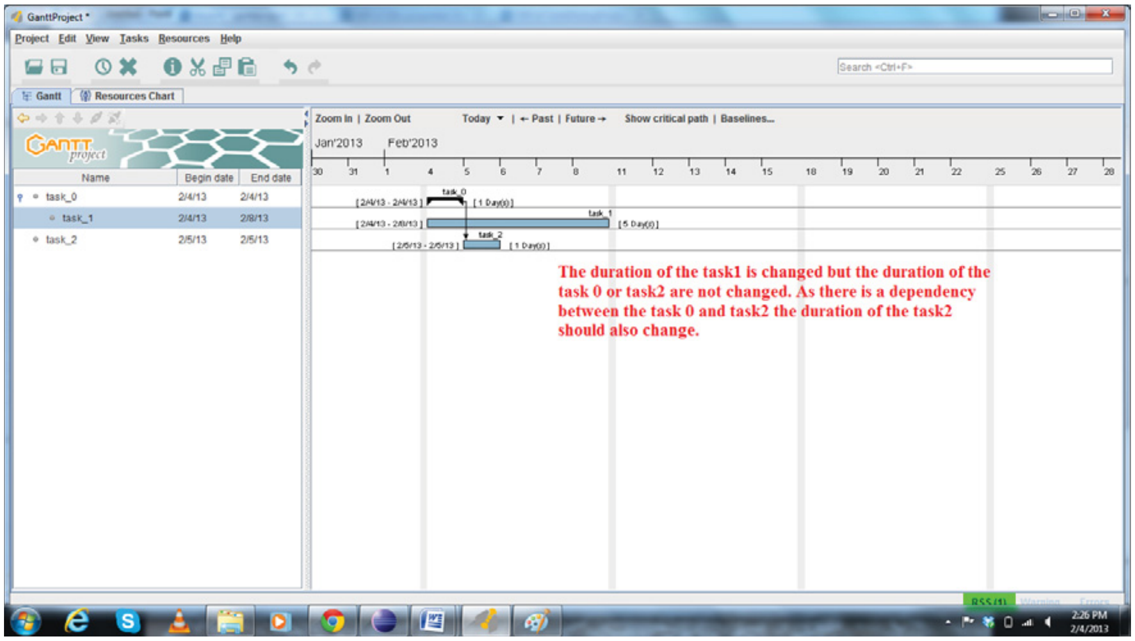
Bug Fix Task: B4
You are given the following bug description.

Bug Title: Wrong update of dependencies on subtask duration change
Bug Description:

1. Create the following hierarchy of tasks:
```
task0
    task1
task2
```
2. Draw dependency task0->task2 (task2 depends on task0)
3. Open properties of task1 and increase its duration. Press OK.\

After doing the above, we get the following actual behavior when we should be getting the expected behavior.
Expected behavior: task0 also becomes longer and task2 shifts.
Actual behavior: task2 stays in its place
See the figure below for an example

Question: Which class(es) would need to be changed/added in order to fix this bug? Please enter the class(es) name(s) only (not the full qualified name).



Name	Begin date	End date
task_0	2/4/13	2/4/13
task_1	2/4/13	2/8/13
task_2	2/5/13	2/5/13

The duration of the task1 is changed but the duration of the task 0 or task2 are not changed. As there is a dependency between the task 0 and task2 the duration of the task2 should also change.

Please give the rationale for your answer:

Difficulty Level:	Very easy	Easy	Difficult	Very difficult
Confidence:	Very confident	Confident	Somewhat confident	Not confident

New Feature Task: N6

You are given the following feature request.

Feature Request

> To be able to import data from basically any source, GanttProject needs to have the ability to import a CSV file with some more details than only task name.
>
> Could be something like:
> [task name],[start date],[end date],[coordinator],[priority]
> where all fields but the task name would be optional.
>
> That way, we can generate data from any other source to bring into GanttProject. With the CVS file format defined, it's easy to create scripts to convert files from any other format to the format understood by GanttProject.

Question: Which class(es) would need to be added or changed in order to add this new feature? Please enter the class(es) name(s) only (not the full qualified name).

Please give the rationale for your answer:

Difficulty Level:	Very easy	Easy	Difficult	Very difficult
Confidence:	Very confident	Confident	Somewhat confident	Not confident

Overview Task: O8

Find the largest method in terms of lines of code (LOC) in following package

```
net.sourceforge.ganttproject.task.algorithm
```

Answer the three questions below.

State the name of the method (only enter the method name):
____

State the class name the method was found in (only enter the class name):
____

Lines of code (LOC) of method: ____________________________

Difficulty Level:	Very easy	Easy	Difficult	Very difficult
Confidence:	Very confident	Confident	Somewhat confident	Not confident

Comments (if any):

Overview Task: O9

Let’s assume that a package **A** is connected with another package **B** if at least one class in package **A** has an *import clause* that imports something that is in package **B**.

Answer the questions below with respect to the following package

```
net.sourceforge.ganttproject.action.task
```

Number of packages that are connected with the above package (You do not need to list them):

____

Which of the connected packages has the largest number of classes? Name the package.

____

Comments (if any):

Difficulty Level:	Very easy	Easy	Difficult	Very difficult
Confidence:	Very confident	Confident	Somewhat confident	Not confident

Overview Task: O10

How many methods does the largest class (in terms of LOC) have in each of these packages?

- `net.sourceforge.ganttproject.export`

  ____

- `net.sourceforge.ganttproject.io`

  ____

Comments (if any):

Difficulty Level:	Very easy	Easy	Difficult	Very difficult
Confidence:	Very confident	Confident	Somewhat confident	Not confident

End Time: ______________

Thank you for participating. Your input is greatly appreciated.

A.3. Post Questionnaire

A.3.1. Group using SeeIT 3D

1) **Were you familiar with the design of the GanttProject system before being introduced to it in this study?**
   a. Yes, I am familiar with the design.
   b. No, I am not familiar with the design.
2) **How frequently do you use GanttProject?**
   a. Never
   b. Almost Never
   c. Occasionally /Sometimes
   d. Almost every time
   e. Every time
3) **What was your overall difficulty level in answering the questions?**
   a. Very easy
   b. Easy
   c. Average
   d. Difficult
   e. Very Difficult
4) **Was the overview on GanttProject you did prior to the study useful?**
   a. Yes
   b. No
5) **Did you have sufficient time to complete the study?**
   a. Yes
   b. No
6) **Have you ever used the SeeIT 3D tool before this study?**
   a. Yes, I used the tool before.
   b. No, I never used the tool before.
7) **How easy is SeeIT 3D to use?**
   a. Very easy
   b. Easy
   c. I needed more time to understand all the features of SeeIT 3D to use it to it's full potential.
   d. Somewhat difficult
   e. Very difficult
8) **Do the SeeIT 3D visualizations correctly depict what you expected for the tasks you had to solve?**
   a. Never
   b. Rarely
   c. Occasionally/Sometimes
   d. Frequently

e. Always

9) **Was the SeeIT 3D tutorial and sample tasks you did prior to the study useful?**
   a. Yes
   b. No

10) **Please assess the usefulness (utility) of SeeIT 3D as a tool for supporting the comprehension and visualization of large systems. Please answer this based on its potential future benefit.**
   a. Very useful
   b. Somewhat useful
   c. Somewhat not useful
   d. Not useful

11) **Please give us your comments about SeeIT 3D. It will help make the tool better in the future.**

12) **Please give us your comments about the study in general**

A.3.2. Group not using SeeIT 3D

1) **Were you familiar with the design of the GanttProject system before being introduced to it in this study?**
   c. Yes, I am familiar with the design.
   d. No, I am not familiar with the design.

2) **How frequently do you use GanttProject?**
   a. Never
   b. Almost Never
   c. Occasionally /Sometimes
   d. Almost every time
   e. Every time

3) **What was your overall difficulty level in answering the questions?**
   a. Very easy
   b. Easy
   c. Average
   d. Difficult
   e. Very Difficult

4) **Was the overview on GanttProject you did prior to the study useful?**
   a. Yes
   b. No

5) **Did you have sufficient time to complete the study?**
   a. Yes
   b. No

6) **If you had the choice to use a visualization tool that visualizes a software system in the form of diagrams, would you consider using it to help you answer the tasks in this study?**
    a. Yes
    b. No
7) **Please give us your comments about the study in general**

A.4. SeeIT 3D Tutorial Used in the Study

Name/ID: ____________________________________
SeeIT 3D Training Session
Abstract
This tutorial describes the usage of SeeIT 3D, a software visualization tool. It is a hands-on training session that takes around 20 minutes and follows a step by step approach. The main functionality of SeeIT 3D is demonstrated by visualizing the JFreeChart open source software system.

Prerequisites

This tutorial assumes that the Eclipse IDE (Juno or Indi) and SeeIT 3D plug-in are already installed in your machine. It is also assumed that the JFreeChart project (v. 1.0.14) has been imported within the IDE as a Java project.
You can download JFreeChart from http://sourceforge.net/projects/jfreechart/files/1.%20JFreeChart/1.0.14/
To install the project in Eclipse, use the "new project" option, and then, the "Java Project from Existing Ant Buildfile". Then choose the build file that is in the ant folder of the project.

Overview

SeeIT 3D is a software visualization tool for the Eclipse IDE. It allows Java developers to visualize and analyze information about a software project. The tool allows the user to navigate, explore and change the mapping between software artifacts (e.g. packages, classes, and methods) and visual properties that include the **color**, **height** and **width** of 3D objects. These 3D objects are *poly cylinders*: three dimensional bars with a polygonal base. The poly cylinders are always grouped in *containers*. For instance, Figure 1 shows a container, drawn with green lines, that represents a class of the JFreeChart system. Each poly cylinder within this container represents a method of this class, and the color and height of these poly cylinders represent the LOC and McCabe metrics, respectively.

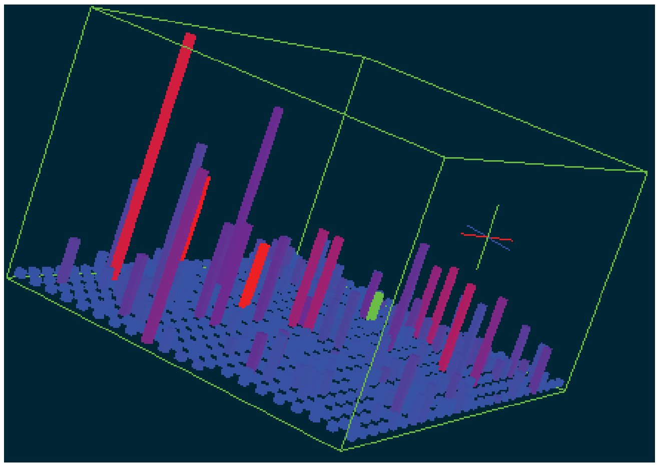


Figure 1: A Container representing a class of the JFreeChart system

The rest of this tutorial explains how to use the various features of the SeeIT 3D plug-in. Please follow the instructions step by step, using your computer.

Step 1: Opening the SeeIT 3D perspective

To open the perspective provided by the SeeIT 3D plug-in:

From the main menu bar select *Window*, and click on the *Open Perspective* option, and then, select *Other…* from the drop-down menu.

Within the *Open Perspective* window select the perspective called *SeeIT 3D*, and press *OK*.

After that, the SeeIT 3D view opens, as well as, the Package Explorer view. Inside the former view, the information regarding the source code will be rendered. Figure 2 shows how this perspective looks.

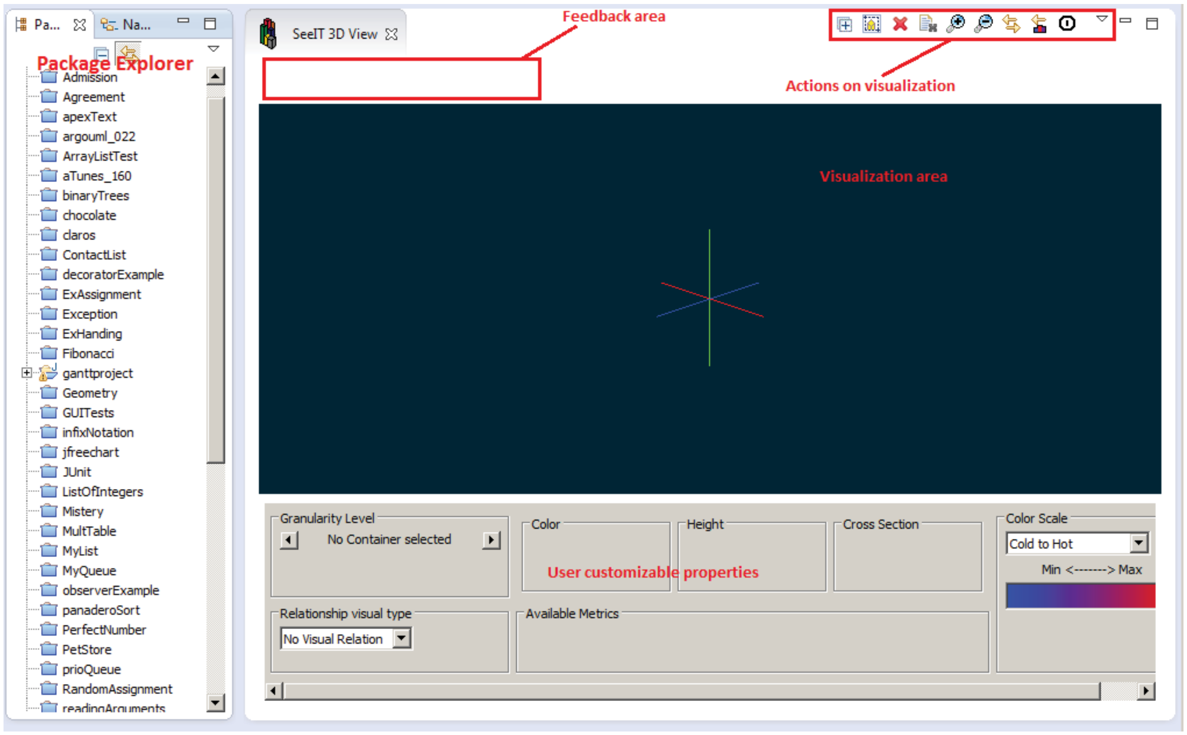


Figure 2: The SeeIT 3D perspective and its components

Figure 2 also shows the main areas of perspective: (i) the *feedback area*, where the tool shows names and properties of the software artifacts being displayed in the visualization area; (ii) The *visualization area*, where the containers and poly cylinders are shown; (iii) The list of buttons that allow the user to perform several *actions on visualization* elements such as the containers and poly cylinders displayed; and (iv) the *user customizable properties* area at the bottom, where the user can modify the mapping between the properties of the poly cylinders and the metrics of the software artifacts. In Figure 2, the visualization and feedback areas are both empty since there are no artifacts displayed.

Step 2: Visualizing a software project

In the Package Explorer view, shown by default in the Java and SeeIT 3D perspectives, select the JFreeChart project, that is, right click on it and select the option *Visualize In SeeIT3D* from the drop-down menu. Alternatively, you can use *Ctrl* + *Alt* + *X*. Figure 3 shows the resulting visualization.

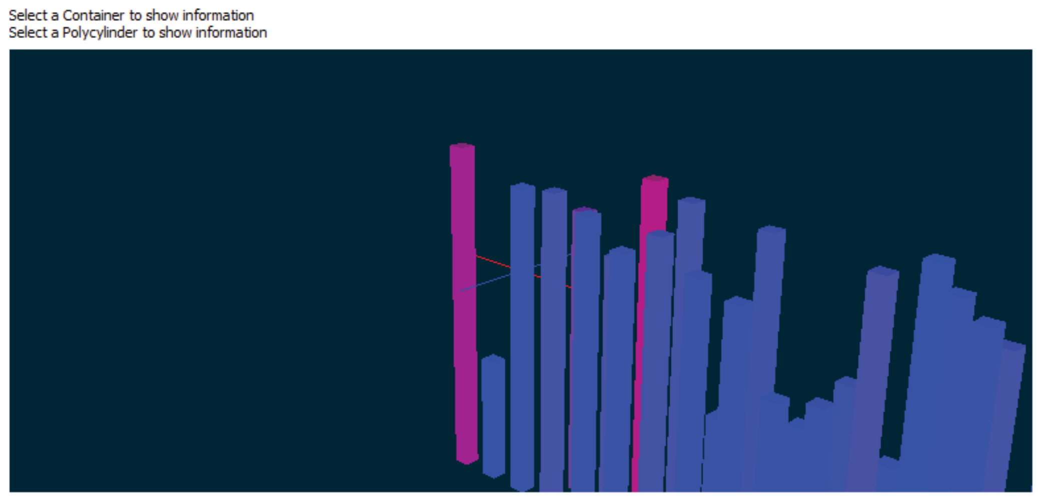
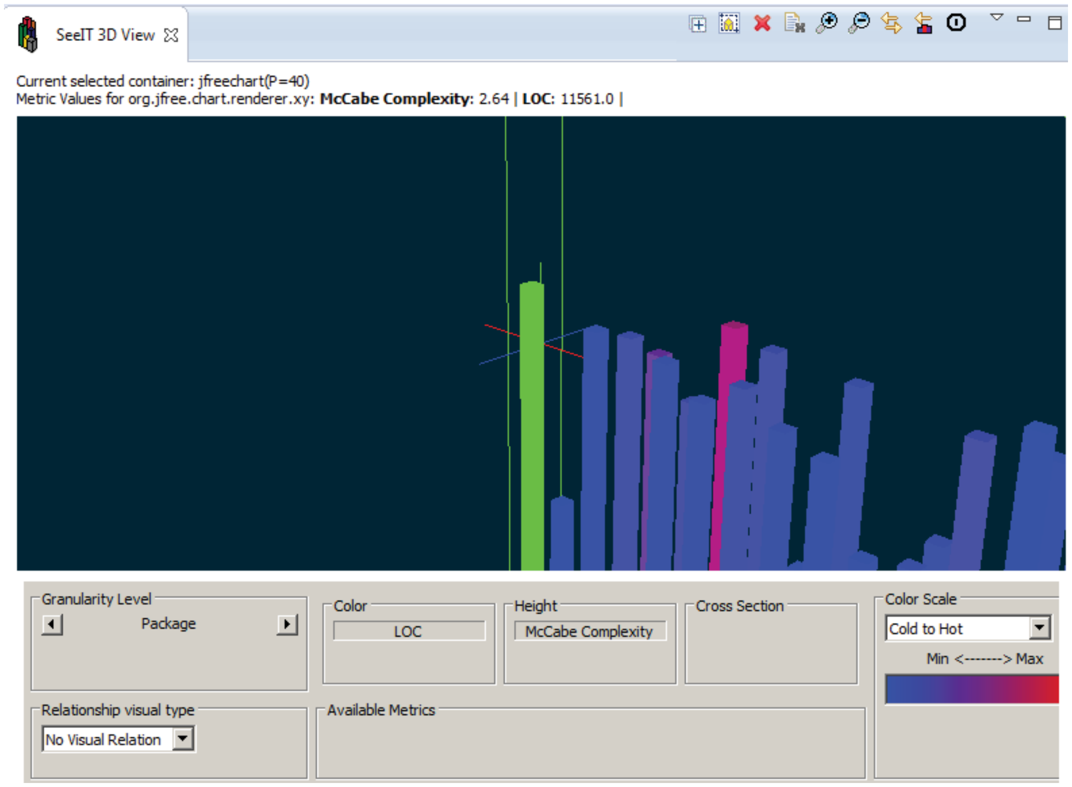

Select a Container to show information
Select a Polycylinder to show information

Figure 3: The feedback and visualization areas after visualizing a project

Since you chose to visualize the entire project, each poly cylinder in Figure 3 represents a package of the JFreeChart project.

Step 3: Selecting a poly cylinder
To select a poly cylinder just click on it. By default, the selected poly cylinder is green as well as the lines of the container where it is.

Please select the leftmost poly cylinder of the entire JFreeChart visualization. Figure 4 shows the results.



Current selected container: jfreechart(P=40)
Metric Values for org.jfree.chart.renderer.xy: **McCabe Complexity:** 2.64 | **LOC:** 11561.0 |

Granularity Level: Package
Color: LOC
Height: McCabe Complexity
Cross Section
Color Scale: Cold to Hot
Min <-------> Max
Relationship visual type: No Visual Relation
Available Metrics

Figure 4: Selecting a poly cylinder

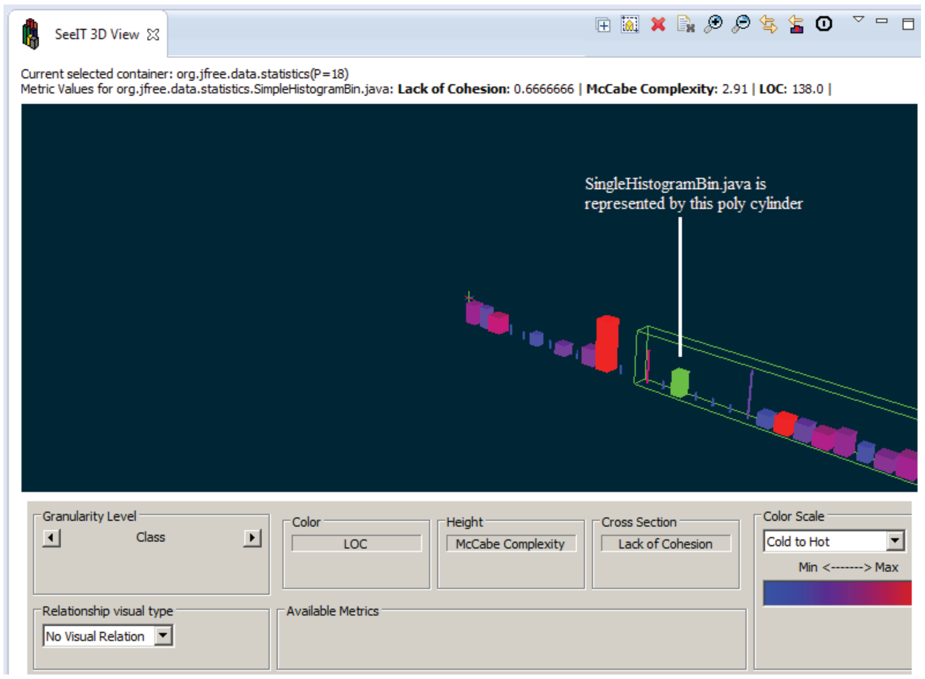
Note that when you select a poly cylinder the name of the current selected container and the properties of the artifact represented by this poly cylinder are shown in the Feedback area. In Figure 4, the container is the entire JFreeChart project, which has 40 packages (P=40) and the selected poly cylinder represents the package org.jfree.chart.renderer.xy.

When you select a poly cylinder, **the customization area** at the bottom of the figure shows important information related to the **granularity level**, the **current mapping** and the **color scale**. For instance, in Figure 4 this information is the following:
The granularity level is ***Package***. It means that each poly cylinder in the container represents a package.
The current mapping indicates that the ***Color*** and ***Height*** of the poly cylinders are representing the ***LOC*** and ***McCabe*** metrics of the packages, respectively.
The **color scale** goes from cold to warm colors. This means that cold colored poly cylinders represent packages with few lines of code.

Step 4: Clearing the visualization area
Now delete all the items in the visualization by clicking on the *Delete All Containers in View* button (The icon is), which is on the top right corner of the visualization area. Alternatively, you can use *Shift* + *Delete*.

Clearing the visualization area before beginning each task is a good working practice.

Step 5: Visualizing more than one container
In the Package Explorer view select the package “org.jfree.chart.editor” and press *Ctrl + Alt + X*. Then, select the package “org.jfree.data.statistics” and press *Ctrl + Alt + X*. Finally, select the polycylinder that represents the class SimpleHistogramBin (see Figure 5). The resulting scene includes the two containers, where the selected poly cylinder and its container are in green.


SeeIT 3D View
SingleHistogramBin.java is represented by this poly cylinder

Current selected container: org.jfree.data.statistics(P=18)
Metric Values for org.jfree.data.statistics.SimpleHistogramBin.java: **Lack of Cohesion:** 0.6666666 | **McCabe Complexity:** 2.91 | **LOC:** 138.0 |

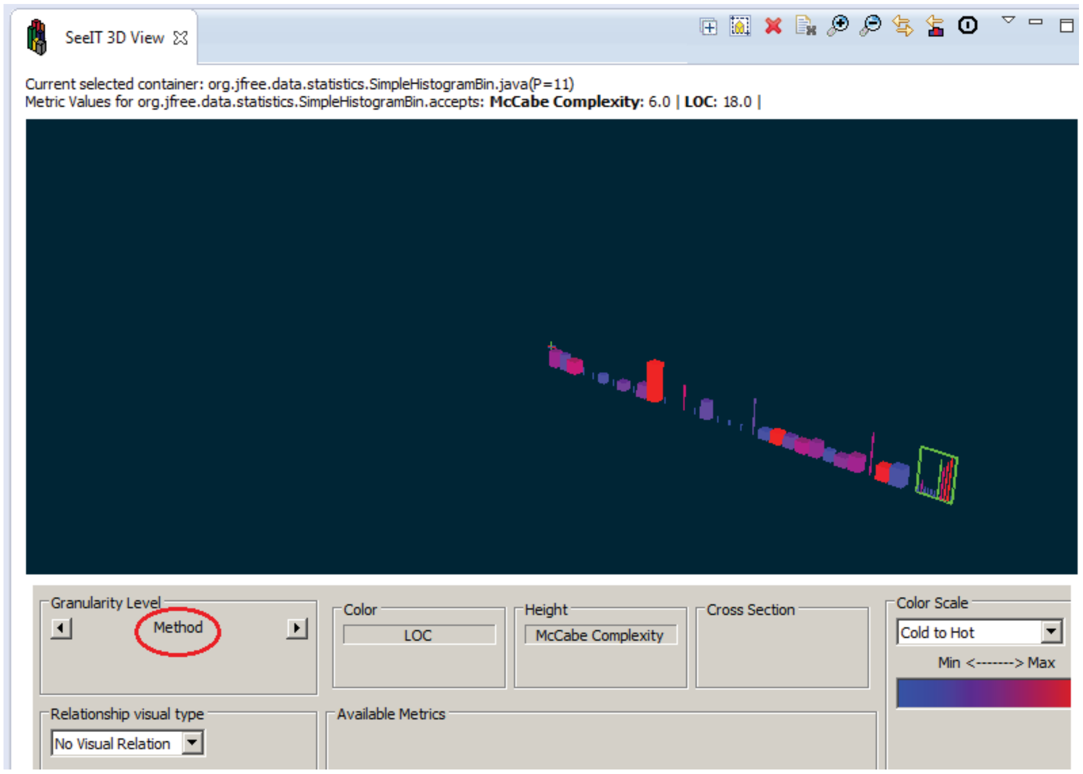
Granularity Level: Class
Color: LOC
Height: McCabe Complexity
Cross Section: Lack of Cohesion
Color Scale: Cold to Hot
Min <-------> Max
Relationship visual type: No Visual Relation
Available Metrics

Figure 5: Visualizing two containers

The feedback area reports that the selected container is the package “org.jfree.data.statistics” that has 18 classes (poly cylinders). The second line in this area indicates that the selected poly cylinder is the java class declared in “SimpleHistogramBin.java”. Also, the metric values of this class are reported (**Lack of Cohesion**: 0.6666666 | **McCabe Complexity**: 2.91 | **LOC**: 138.0 |).

The customization area at the bottom of the figure shows the granularity level, the current mapping and the color scale.

Step 6: Expanding a poly cylinder
Verify that the selected poly cylinder is “SimpleHistogramBin.java” (see Figure 5), and then, click on the *Expand the selected Polycylinder as a Container* button (the icon is ⊞)or press *Ctrl + E*.As a result, a new container is added to the scene. Finally, select this new container. Figure 6 shows the resulting scene.



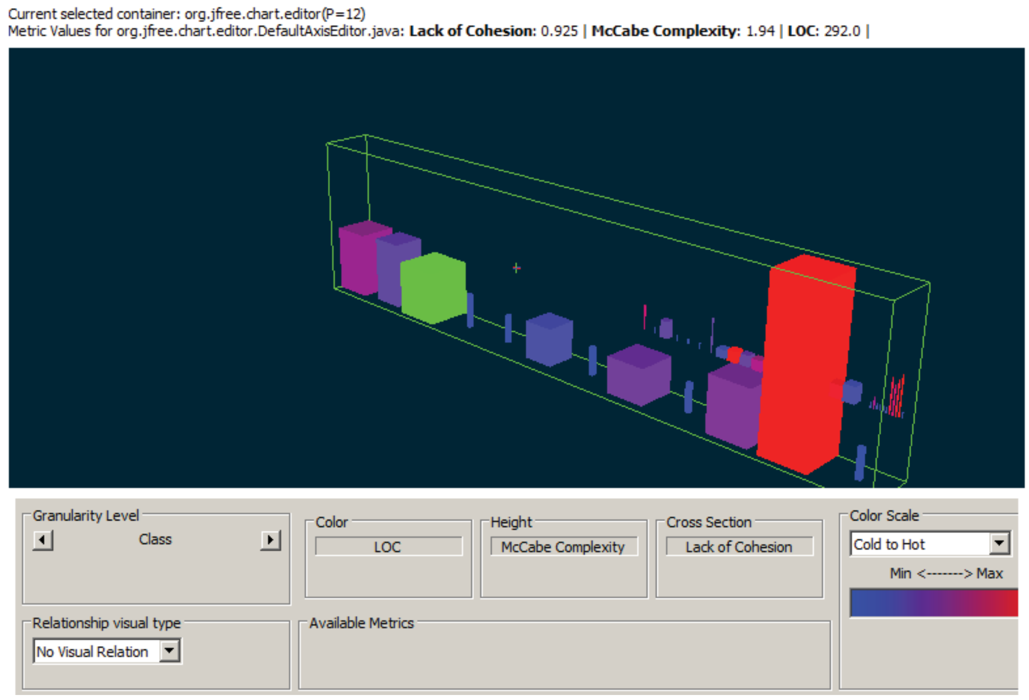
Current selected container: org.jfree.data.statistics.SimpleHistogramBin.java(P=11)
Metric Values for org.jfree.data.statistics.SimpleHistogramBin.accepts: **McCabe Complexity:** 6.0 | **LOC:** 18.0 |

Granularity Level: Method
Color: LOC
Height: McCabe Complexity
Cross Section
Color Scale: Cold to Hot
Min <-------> Max
Relationship visual type: No Visual Relation
Available Metrics

Figure 6: Expanding the selected poly cylinder

The new small container is the result of expanding the selected polycylinder, i.e., the class "SimpleHistogramBin". The poly cylinders of this new container represent the methods of the class "SimpleHistogramBin". Also, notice that the feedback area reports information about this class and the customization area indicates that the granularity level is *Method*.

Step 7: Zooming in and out containers
Select the leftmost container in the visualization area which represents the "org.jfree.chart.editor" package. Then, click on the the *Scale Up Container* button (the icon is) several times to increase its size. The resulting scene should look like Figure 7.


Current selected container: org.jfree.chart.editor(P=12)
Metric Values for org.jfree.chart.editor.DefaultAxisEditor.java: **Lack of Cohesion:** 0.925 | **McCabe Complexity:** 1.94 | **LOC:** 292.0 |

Granularity Level: Class
Color: LOC
Height: McCabe Complexity
Cross Section: Lack of Cohesion
Color Scale: Cold to Hot
Min <-------> Max
Relationship visual type: No Visual Relation
Available Metrics

Figure 7: Zooming in a container

Use the *Scale Down Container* button (its icon is) to perform the opposite action, i.e., zooming out the container. Use this button until the container recovers its original size, as in Figure 6.

Alternatively you can use the mouse wheel to resize all the containers at the same time. Although the process is slightly slower, there is finer control over the size of the containers. Please use the wheel mouse to check this.

Step 8: Translating, rotating, and removing containers
To explain these operations we will assume that the visualization area is as shown in Figure 6. That is, there are three containers that represent three artifacts: the package org.jfree.chart.editor(P=12), the package org.jfree.data.statistics(P=18), and the class org.jfree.data.statistics.SimpleHistogramBin.java(P=11)

To translate (move) the entire scene, simply right click on a place where no containers are present and drag the mouse in the desired direction. Please, translate the visualization so it is centered on the screen. The resulting scene should look like Figure 8.

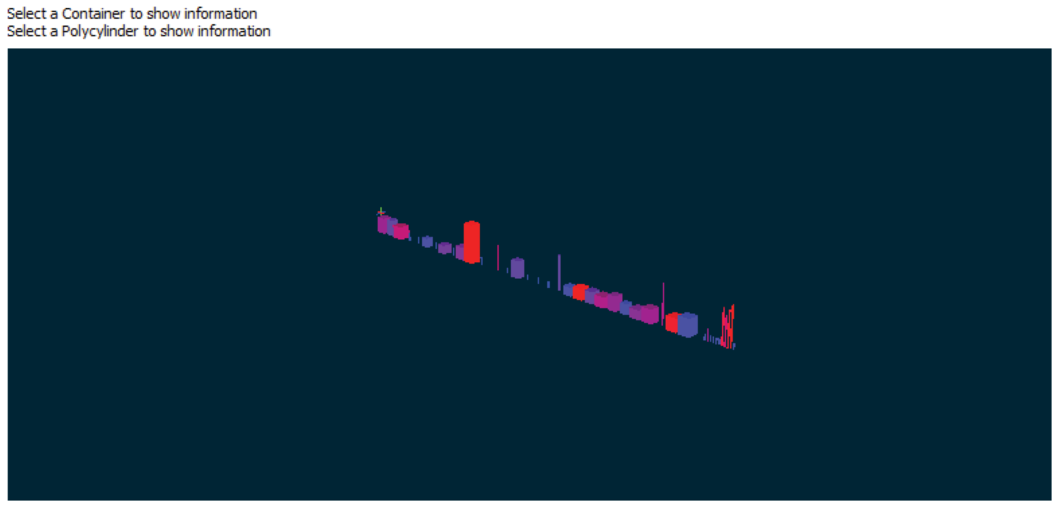

Select a Container to show information
Select a Polycylinder to show information

Figure 8: Translating the entire scene

To rotate the entire scene, use left button of the mouse. Please, press the mouse left button on any place of the visualization area where no containers are present and drag the mouse to rotate the visualization so that it is horizontally aligned with the screen. The resulting scene should look like Figure 9.

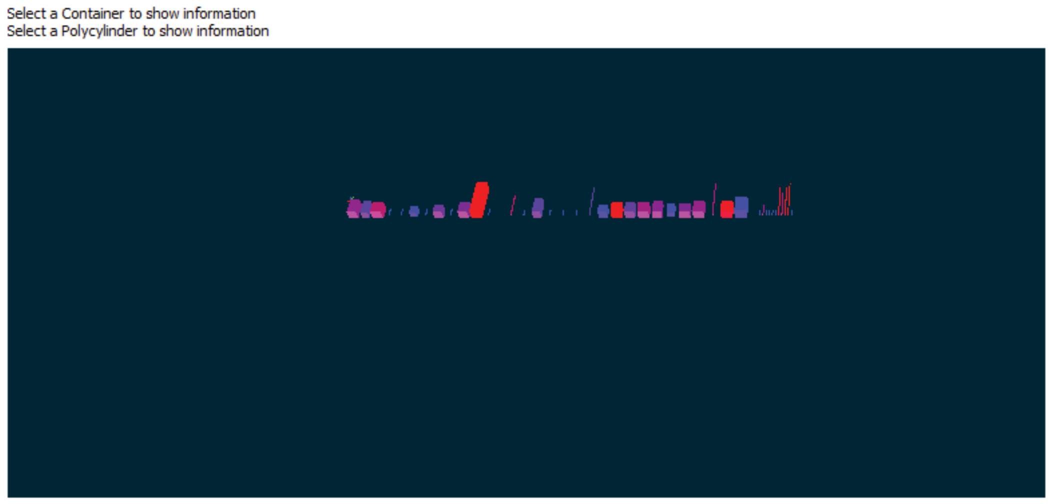

Select a Container to show information
Select a Polycylinder to show information

Figure 9: Rotating the entire scene

These two operations (rotation and translation) can be also performed on each of the containers in the view. To do so, select a container (or multiple holding the Ctrl key) and translate it using the right button or rotate it using the left button. So please translate, scale and rotate the containers so they match the scene in Figure 10.

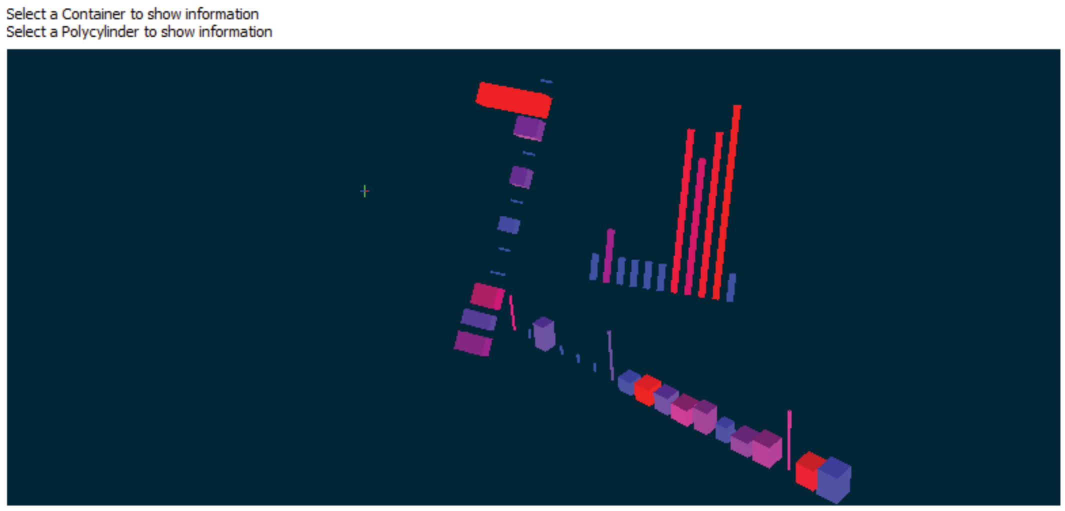

Select a Container to show information
Select a Polycylinder to show information

Figure 10: Manipulating individual containers

To remove an individual container, simply click on it, and then, use the *Delete the current container* button (Its icon is ✖). As a final operation on the current scene (Figure 10) select and remove the leftmost container, i.e., the one that represents the org.jfree.chart.editor package. The resulting scene should look like Figure 11.

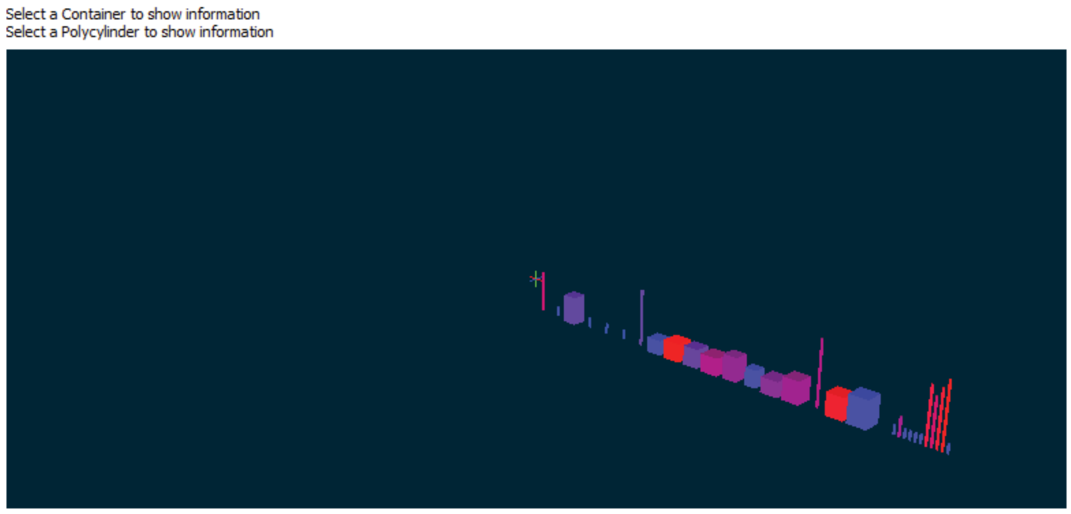
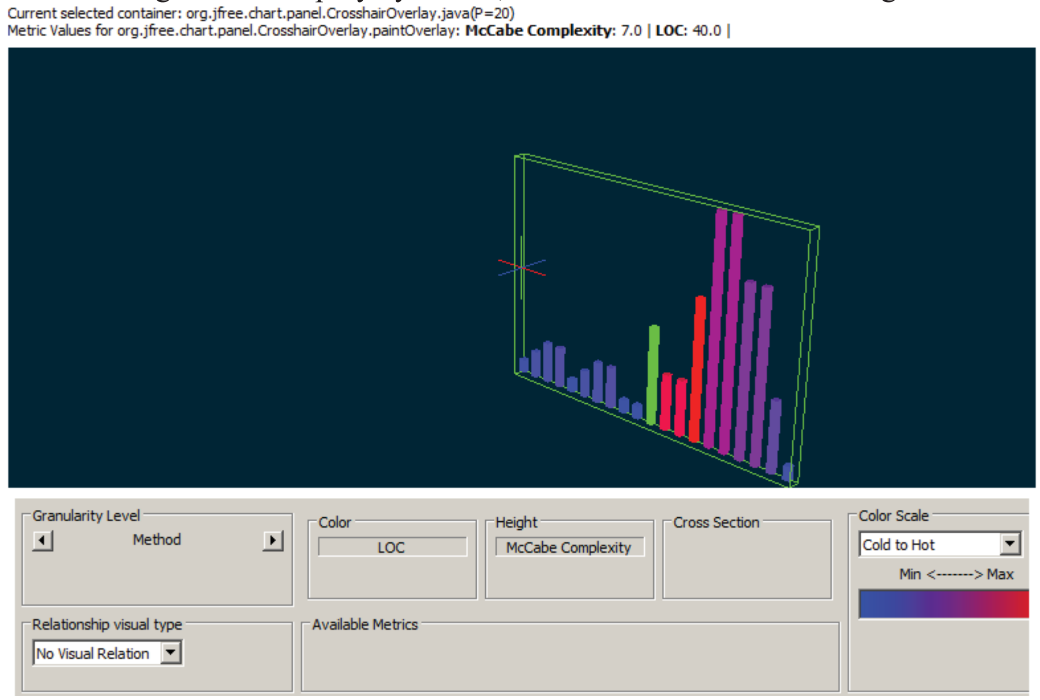

Select a Container to show information
Select a Polycylinder to show information

Figure 11: Removing an individual container

More importantly, after removing an individual container, the remaining containers back to their original size and position. In this regard, SeeIT 3D offers the *Reset Visualization* button that allows the user to reset the entire visualization, i.e., all the visualized containers return to their original position and size.

Step 9: Changing the mapping and the granularity level

To start this part of the tutorial, remove all the containers using the *Delete All Containers* button. Then, use the package explorer to find and visualize the CrosshairOverlay class. This class is one of three classes of the package org.jfree.chart.panel. After zooming out and selecting one of the poly cylinders, the scene should look like Figure 12.


Current selected container: org.jfree.chart.panel.CrosshairOverlay.java(P=20)
Metric Values for org.jfree.chart.panel.CrosshairOverlay.paintOverlay: **McCabe Complexity:** 7.0 | **LOC:** 40.0 |

Granularity Level: Method
Color: LOC
Height: McCabe Complexity
Cross Section
Color Scale: Cold to Hot
Min <-------> Max
Relationship visual type: No Visual Relation
Available Metrics

Figure 12: Visualizing a class

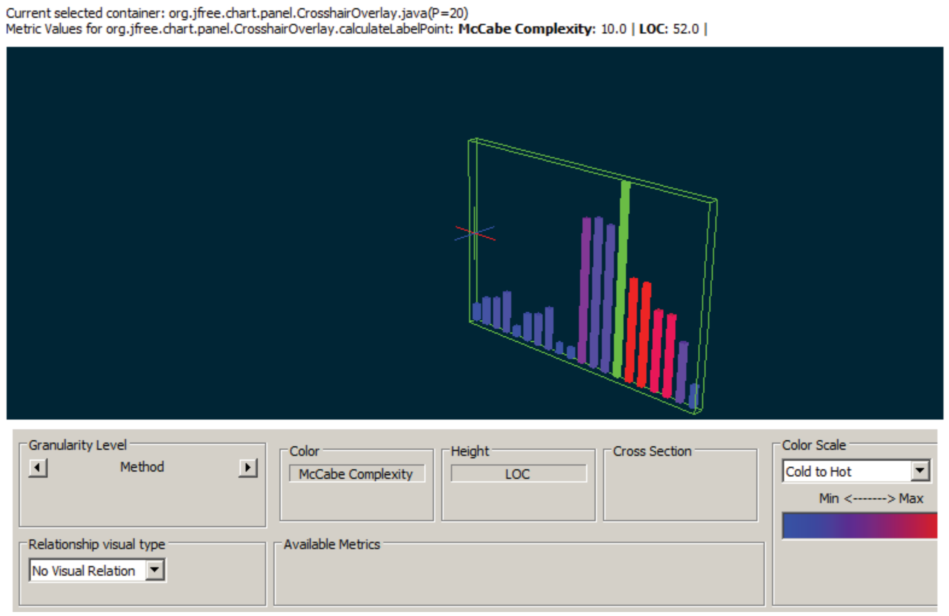
As you can see at the bottom of Figure 12, **the customization area** indicates that the granularity level is *Method*. It means that each poly cylinder represents a method of the CrosshairOverlay class. Additionally, it shows the mapping: the *Color* is linked to LOC while the *Height* denotes the McCabe Complexity.

Moreover, the first line of the feedback area indicates that the container represents the CrosshairOverlay class which has 20 methods (P=20). The second line shows the metric values for method paintOverlay which is represented by the selected poly cylinder (in green).

The interaction options in the customization area (in the lower section of the SeeIT 3D view) can be classified into various sections:

Metric Mapping: allows changing the mapping between software metrics (LOC, McCabe, LCOM, etc…) and visual properties (*Color*, *Height*, or *Cross Section*). This task is accomplished by dragging a certain metric to a specific visual property or to the *Available Metrics* box to not map it to a visual property. The number you see for the metrics are calculated based on some formula that you do not need to know at this time.

Use this dragging feature to swap the current mapping, that is, McCabe Complexity and LOC are mapped to Color and LOC, respectively. After zooming out and selecting the highest poly cylinders, the scene should looks like Figure 13.

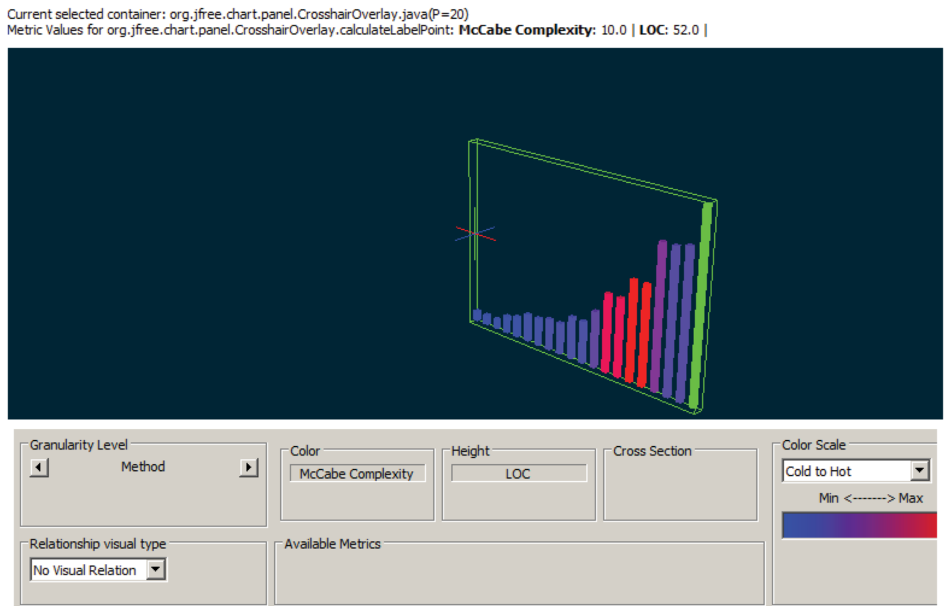


Current selected container: org.jfree.chart.panel.CrosshairOverlay.java(P=20)
Metric Values for org.jfree.chart.panel.CrosshairOverlay.calculateLabelPoint: **McCabe Complexity:** 10.0 | **LOC:** 52.0 |

Granularity Level: Method
Color: McCabe Complexity
Height: LOC
Cross Section:
Color Scale: Cold to Hot
Relationship visual type: No Visual Relation
Available Metrics

Figure 13: Changing the mapping

Since the highest poly cylinder is selected, the feedback area shows that the longest method is calculateLabelPoint and has 52 lines of code. Additionally, you can use the *Sort Polycylinders* button (its icon is) to sort the poly cylinders of the selected container. If you use this button and zoom out the resulting scene, the visualization area should look like Figure 14.

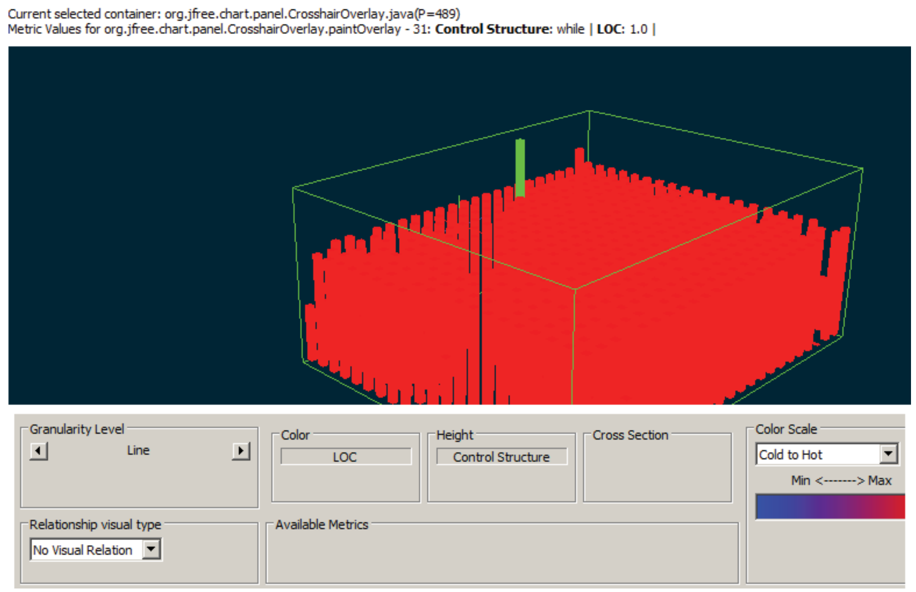

Current selected container: org.jfree.chart.panel.CrosshairOverlay.java(P=20)
Metric Values for org.jfree.chart.panel.CrosshairOverlay.calculateLabelPoint: **McCabe Complexity:** 10.0 | **LOC:** 52.0 |

Granularity Level: Method
Color: McCabe Complexity
Height: LOC
Cross Section
Color Scale: Cold to Hot
Relationship visual type: No Visual Relation
Available Metrics

Figure 14: Sorting poly cylinders

Granularity Level: allows choosing the granularity level of the polycylinders contained in the current selection of containers. With the *right arrow* the granularity will be higher while with the *left arrow* will be lower.

Press the *right arrow* to change the granularity to *Line*. After zooming out and selecting one of the poly cylinders, the scene should looks like Figure 15.


Current selected container: org.jfree.chart.panel.CrosshairOverlay.java(P=489)
Metric Values for org.jfree.chart.panel.CrosshairOverlay.paintOverlay - 31: **Control Structure:** while | **LOC:** 1.0 |

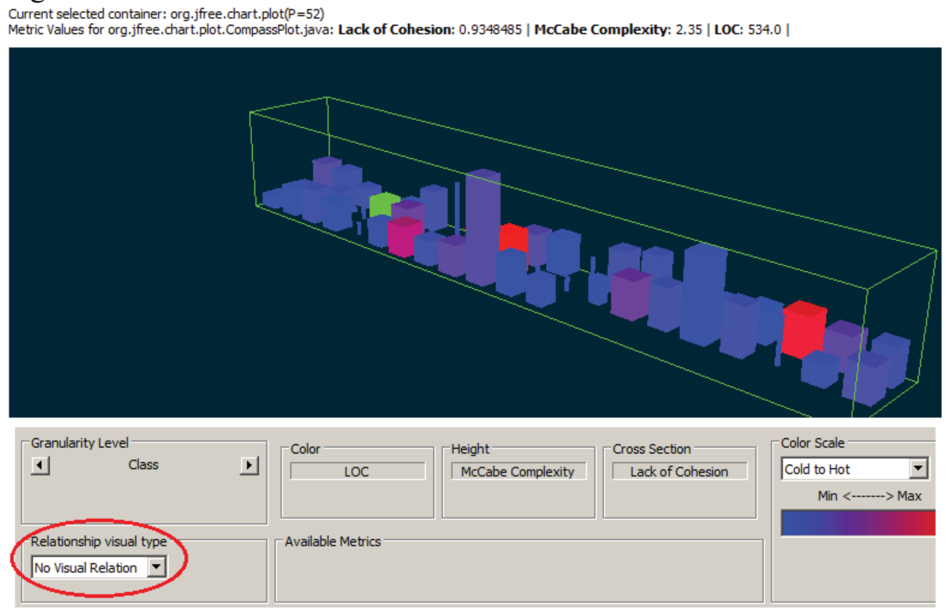
Granularity Level: Line
Color: LOC
Height: Control Structure
Cross Section
Color Scale: Cold to Hot
Relationship visual type: No Visual Relation
Available Metrics

Figure 15: Changing the granularity level

When the granularity level is *Line*, each poly cylinder is a line and SeeIT 3D reports which of them are control structures. For instance, in Figure 15 the feedback area shows that the selected poly cylinder represents the line 31 which is a *while*.

Relationship visual types: allows selecting the mechanism used to represent relationships between containers; this selection applies to every container selected in the visualization area. You can use the options of *Common Base*, *Arcs*, *Lines*, and *Movement*. The next section of this tutorial explains how to use this feature of the plug-in.

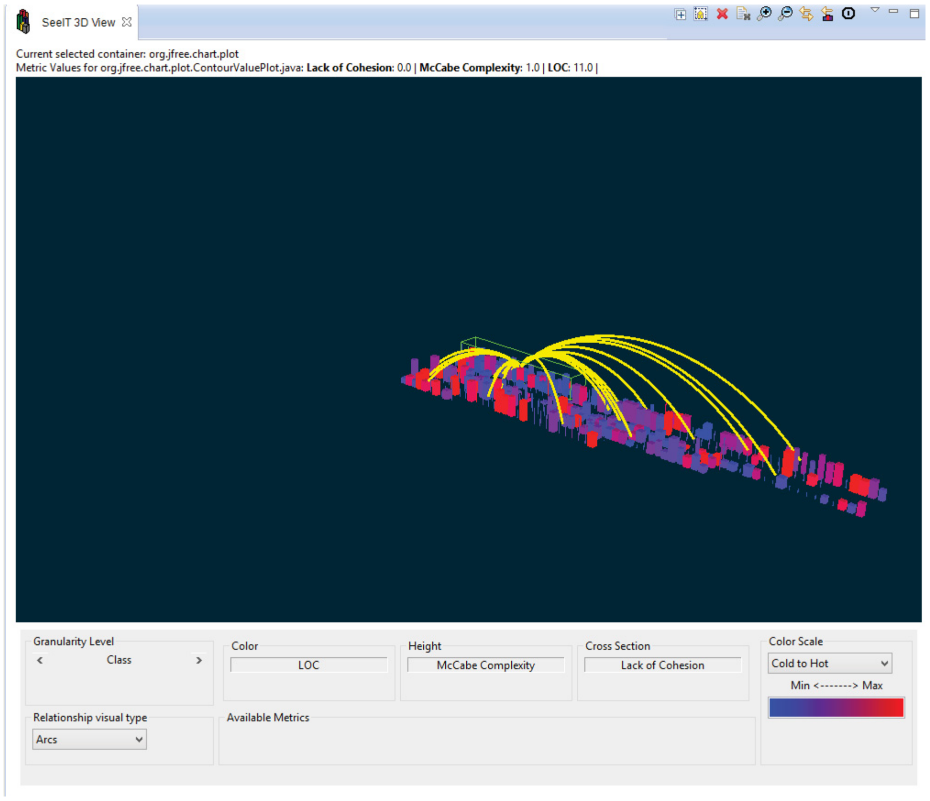
Step 10: Visualizing relationships among packages
To start this part of the tutorial, remove all the containers using the *Delete All Containers* button. Then, use the package explorer to find and visualize theorg.jfree.chart.plot package. This package has 52 classes as it is shown in the feedback area. After zooming out, translating, and selecting one of the poly cylinders, the scene should looks like Figure 16.


Current selected container: org.jfree.chart.plot(P=52)
Metric Values for org.jfree.chart.plot.CompassPlot.java: **Lack of Cohesion:** 0.9348485 | **McCabe Complexity:** 2.35 | **LOC:** 534.0 |

Granularity Level: Class
Color: LOC
Height: McCabe Complexity
Cross Section: Lack of Cohesion
Color Scale: Cold to Hot
Min <-------> Max
Relationship visual type: No Visual Relation
Available Metrics

Figure 16: Visualizing a package

Now select the container and use the *Relationship visual type* option (see the red oval in Figure 16) to choose *Arcs* as the way to show the relationships between this package and other packages of the JFreeChart project. The scene should looks like Figure 17.



Current selected container: org.jfree.chart.plot
Metric Values for org.jfree.chart.plot.ContourValuePlot.java: **Lack of Cohesion:** 0.0 | **McCabe Complexity:** 1.0 | **LOC:** 11.0 |

Granularity Level: < Class >
Color: LOC
Height: McCabe Complexity
Cross Section: Lack of Cohesion
Color Scale: Cold to Hot
Min <-------> Max
Relationship visual type: Arcs
Available Metrics

Figure 17: Visualizing a package and related packages

The yellow arcs in Figure 17 connect the selected package (org.jfree.chart.plot) with the packages it is related to.
In this context, a package **A** <u>is connected (related) with</u> another package **B** if at least one class in package **A** has an *import clause* that imports something that is in package **B**.

NOTE: If the selected package has no relationship with any other package, changing the relationship visual type will only reset the containers to the original layout, and set the relationship visual type to *No Visual Relation*.

List of SeeIT 3D Commands

The table below sums up all of the commands that the user can use to analyze and manipulate the graphical elements of the visualization

Icon	**Description**	**Key binding**
⊞	This option allows visualizing a container from the selected polycylinder. For example, when a polycylinder represents a package, if this action is performed SeeIT 3D will add the corresponding container of the selected polycylinder where the	Ctrl+E

	granularity level will be lower that then original container	
⬚	This option allows drawing a rectangle in the visualization area, in order to select multiple polycylinders and containers at once	S
✖	This button will delete the current selected container from the visualization	Del
🗎	This option will delete all containers in the visualization	Shift+Del
🔍	This action will increase the apparent size of the selected containers	Alt++
🔍	This action will decrease the apparent size of the select containers	Alt+-
⇆	This button will toggle the link between visualization and package explorer view. When activated the selection of a polycylinder in the view will trigger the selection of the corresponding artifact in the package explorer view. This way is easy to know what artifact is selected in the SeeIT 3D view.	
⇇	This button will sort the polycylinders of the selected containers in the view. It will take into consideration the visual property selected for sorting i.e. Height or Color	Ctrl+Alt+S
⏻	This action will reset the visualization. This means SeeIT 3D will place the containers at the origin of the visualization as well as updating the values of the preferences selected by the user	Ctrl+R
💾	SeeIT 3D allows to save a visualization. Choosing this option will ask for a place to save the current visualization for later loading	Ctrl+S

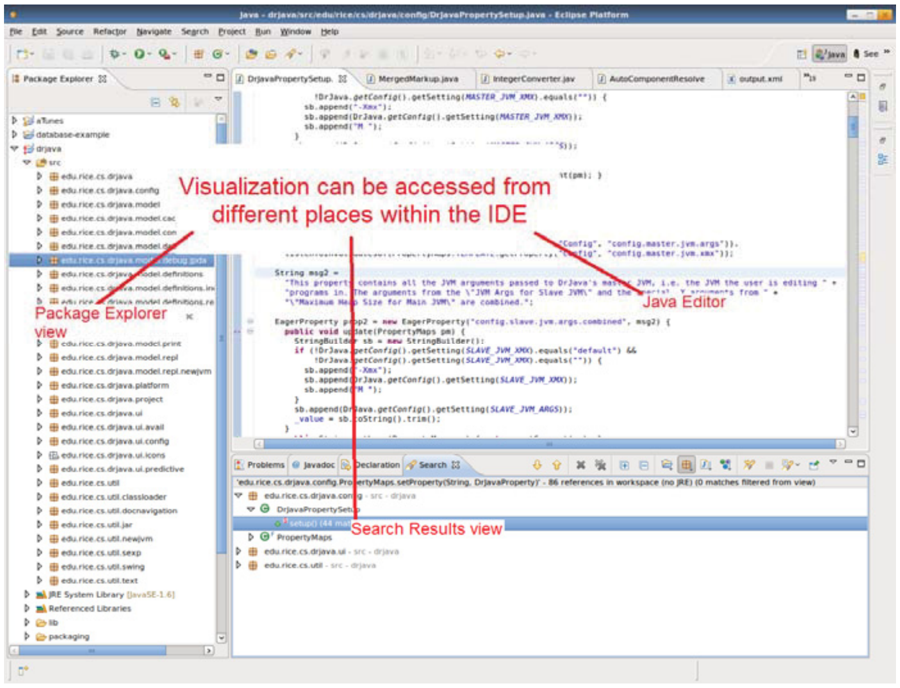
📂	This option will load a previously saved visualization	Ctrl+O
-	Make more or less transparent a set of select poly cylinders	Alt+. or Alt+,

List of available metrics
The table below explains all of the metrics that the user can use to analyze software artifacts represented by the graphical elements of the visualization

Metric	**Description**	**Granularity levels where it applies**
LOC	Software metric used to measure the size of a computer program by counting the number of lines in a source code artifact.	Package, Class, Method
Lack of cohesion	Cohesion metrics measure how well the methods of a class are related to each other. A cohesive class performs one function. A non-cohesive class performs two or more unrelated functions.	Class
McCabe Complexity	The cyclomatic complexity of a section of source code is the count of the number of linearly independent paths through the source code.	Package, Class, Method
Control Structures	This metric only counts how many control structures (if, while, for, do) there are ina specified code artifact. Thus, SeeIT 3D only indicates if the line at hand is a control structure or not.	Line

Accessible places in the IDE
The visualization can be triggered from several places in the IDE. Specifically, SeeIT 3D defines three views where the visualization can be triggered:

- The **Package explorer** view, using the Cltr+Alt+X key combination or by right clicking the element and selecting the option *Visualize in SeeIT 3D*
- The **Search Results** view, using the same mechanism as explained above
- The **Java Editor**, where right clicking the editor will pop up a menu that allows to visualize the current Java File, the Parent Package or the Parent Project of the corresponding file. Figure 18 shows these three options.


Visualization can be accessed from different places within the IDE
Package Explorer view
Java Editor
Search Results view

```
      !DrJava.getConfig().getSetting(MASTER_JVM_XMX).equals("")) {
    sb.append("-Xmx");
    sb.append(DrJava.getConfig().getSetting(MASTER_JVM_XMX));
    sb.append("M ");
  }
```

```
String msg2 =
  "This property contains all the JVM arguments passed to DrJava's master JVM, i.e. the JVM the user is editing " +
  "programs in. The arguments from the \"JVM Args for Slave JVM\" and the
  "\"Maximum Heap Size for Main JVM\" are combined.";

EagerProperty prop2 = new EagerProperty("config.slave.jvm.args.combined", msg2) {
  public void update(PropertyMaps pm) {
    StringBuilder sb = new StringBuilder();
    if (!DrJava.getConfig().getSetting(SLAVE_JVM_XMX).equals("default") &&
        !DrJava.getConfig().getSetting(SLAVE_JVM_XMX).equals("")) {
      sb.append("-Xmx");
      sb.append(DrJava.getConfig().getSetting(SLAVE_JVM_XMX));
      sb.append("M ");
    }
    sb.append(DrJava.getConfig().getSetting(SLAVE_JVM_ARGS));
    _value = sb.toString().trim();
  }
```

'edu.rice.cs.drjava.config.PropertyMaps.setProperty(String, DrjavaProperty)' - 86 references in workspace (no JRE) (0 matches filtered from view)

Figure 18: The views where the SeeIT 3D visualization can be triggered

Challenge: Perform three overview tasks

As a final activity, use the plug-in to solve the following overview task related to the JFreeChart project.

Find and name the largest method (in terms of LOC) in each one of the following packages:
Answer:

org.jfree.chart.imagemap ______________________________

org.jfree.data.gantt ______________________________

org.jfree.data.time ____________________________________

Find and name all the packages *related* with the package org.jfree.data.contour. Hint: Step 10.
Answer:

Find the 3 classes with the lowest lack of cohesion value (Higher than 0.0) in the "org.jfree.data" package.
Answer:

org.jfree.data

(3 classes with lowest Lack of cohesion value higher than 0.0)

References

A. Bragdon, S. P. R., R. Zeleznik, S. Karumuri, W. Cheung, J. Kaplan, C. Coleman, F. Adeputra, and J. J. LaViola,Jr. (2010). Code bubbles: rethinking the user interface paradigm of integrated development environments. 32nd ACM/IEEE International Conference on Software Engineering, New York, NY, USA.

Bassil, S. and R. K. Keller (2001). Software Visualization Tools: Survey and Analysis. International Workshop on Program Comprehension, Toronto, Ont., Canada.

Bednarik, R. and M. Tukiainen (2006). An Eye-tracking Methodology for Characterizing Program Comprehension Processes. Symposium on Eye tracking research & Applications (ETRA), San Diego, California, ACM Press.

Binkley, D., M. Davis, D. Lawrie, J. I. Maletic, C. Morrell and B. Sharif (2013). "The Impact of Identifier Style on Effort and Comprehension." Empirical Software Engineering Journal (invited submission) 18(2): 219-276.

Bragdon, A. (2010). Developing and evaluating the code bubbles metaphor 32nd International Conference on Software Engineering, Cape Town, South Africa.

Cornelissen, B., A. Zaidman and A. van Deursen (2011). "A Controlled Experiment for Program Comprehension through Trace Visualization." IEEE Transactions on Software Engineering 37(3): 341-355.

Duchowski, A. T. (2003). Eye Tracking Methodology: Theory and Practice. London, Springer-Verlag.

Gadapa, S. (2012). Assessing SeeIT3D, A Software Visualization Tool.

Guéhéneuc, Y.-G. (2006). TAUPE: towards understanding program comprehension. 16th IBM Centers for Advanced Studies on Collaborative research (CASCON), Canada, ACM Press.

Jeanmart, S., Y.-G. Guéhéneuc, H. Sahraoui and N. Habra (2009). Impact of the Visitor Pattern on Program Comprehension and Maintenance. 3rd International Symposium on Empirical Software Engineering and Measurement, Lake Buena Vista, Florida.

Just, M. and P. Carpenter (1980). "A Theory of Reading: From Eye Fixations to Comprehension." Psychological Review 87: 329-354.

Kagdi, H., S. Yusuf and J. I. Maletic (2007). On Using Eye Tracking in Empirical Assessment of Software Visualizations. ACM Workshop on Empirical Assessment of Software Engineering Languages and Technologies, Atlanta, GA.

Kagdi, H., S. Yusuf and J. I. Maletic (2007). On Using Eye Tracking in Empirical Assessment of Software Visualizations. Workshop on Empirical Assessment of Software Engineering Languages and Technologies, WEASELTech'07. Atlanta Georgia, USA: 21-22.

Lange, C. F. J. and M. R. V. Chaudron (2007). "Interactive Views to Improve the Comprehension of UML Models - An Experimental Validation." International Conference on Program Comprehension (ICPC'07): 221-230.

Lange, C. F. J., M. A. M. Wijns and M. R. V. Chaudron (2007). MetricViewEvolution: UML-based Views for Monitoring Model Evolution and Quality. 11th European Conference on Software Maintenance and Reengineering (CSMR).

Lanza, M. (2003). Codecrawler – a lightweight software visualization tool. 2nd International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT), IEEE CS Press.

Lanza, M. L. and S. Ducasse (2005). CodeCrawler – An Extensible and Language Independent 2D and 3D Software Visualization Tool. Tools for Software Maintenance and Reengineering.

Maletic, J. I., A. Marcus and M. L. Collard (2002). A Task Oriented View of Software Visualization. IEEE Workshop on Visualizing Software for Understanding and Analysis (VISSOFT'02), Paris, France.

Marcus, A., D. Comorski and A. Sergeyev (2005). Supporting the evolution of a software visualization tool through usability studies. 13th International Workshop on Program Comprehension, IWPC 2005.

Petre, M. and E. Quincey (2006). "A Gentle Overview of Software Visualization." Psychology of Programming Interest Group (PPIG) Newsletter.

Pietinen, S., R. Bednarik, T. Glotova, V. Tenhunen and M. Tukiainen (2008). A method to study visual attention aspects of collaboration: eye-tracking pair programmers simultaneously. 2008 symposium on Eye tracking research & applications, New York, NY, USA.

Price, B. A., R. M. Baecker and I. S. Small (1993). "A Principled Taxonomy of Software Visualization." Journal of Visual Languages and Computing(4): 3.

Price, B. A., I. S. Small and R. M. Baecker (1992). A taxonomy of software visualization. Twenty-Fifth Hawaii International Conference on System Sciences. 2: 597–606.

Ramírez, D. M. "SeeIT 3D." from https://github.com/davidmr/seeit3d

Ramírez, D. M. (2010). Development of a 3D tool for visualization of different software artifacts and their relationships, Universidad Nacional de Colombia.

Rayner, K. (1998). "Eye Movements in Reading and Information Processing: 20 Years of Research." Psychological Bulletin 124(3): 372-422.

Roman, G. and K. C. Cox (1993). "A taxonomy of program visualization systems." Computer 26(12): 11-24.

Sharafi, Z., Z. Soh, Y.-G. Gueheneuc and G. Antoniol (2012). Women and Men - Different but Equal: On the Impact of Identifier Style on Source Code Reading. International Conference on Program Comprehension (ICPC 2012), Passau, Germany, IEEE.

Sharif, B. and J. I. Maletic (2010). An Eye tracking Study on camelCase and Under_score Identifier Styles. 18th IEEE International Conference on Program Comprehension (ICPC'10), Braga, Portugal.

Sharif, B. and J. I. Maletic (2010). An Eye tracking Study on the Effects of Layout in Understanding the Role of Design Patterns. 26th IEEE International Conference on Software Maintenance (ICSM'10), Timisoara, Romania.

Storey, M.-A., C. Best and J. Michaud (2001). SHriMP Views: An Interactive Environment for Exploring Java Programs. 9th International Workshop on Program Comprehension (IWPC'01).

Teyseyre, A. and M. R. Campo (2009). "An Overveiw of 3D Software Visualization." IEEE Transactions on Visualization and Computer Graphics 15(1): 87-105.

Wettel, R., M. Lanza and R. Robbes (2011). Software systems as cities: a controlled experiment. Software Engineering (ICSE), 2011 33rd International Conference on.

Wettel, R. and M. L. Lanza (2008). CodeCity: 3D Visualization of Large-Scale Software. International Conference on Software Engineering (ICSE), Leipzig, Germany.

Wohlin, C., P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell and A. Wesslén (1999). Experimentation in Software Engineering - An Introduction, Kluwer Academic Press.

Yusuf, S., H. Kagdi and J. I. Maletic (2007). Assessing the Comprehension of UML Class Diagrams via Eye Tracking. Proceedings of the 15th IEEE International Conference on Program Comprehension, IEEE Computer Society: 113-122.

Zhang, K., Ed. (2003). Software Visualization: From Theory to Practice, Kluwer Academic Publishers.