SE-2832: Introduction to Software Verification
Lab 10: Analyzing Source Code for Complexity

Due: By the end of Lab on 5/15/2014

1. Objectives

- Use source code analysis to determine the complexity of software modules
- Using source code metrics, critique the quality of a software product
- Determine which modules may require extra testing or are otherwise overly complex and may pose problems during development

2. Introduction

CyVis is a free software metrics collection, analysis and visualization tool for Java. It collects metrics from java .class files and .jar files and creates graphical displays which indicate the software complexity for these modules. Cyvis includes GUI tool, but it can also be operated on the command shell as well as integrated into an Ant task. Colors can be modified to suite the users desires, and thresholds are also configurable.

Once the metrics are collected, the statistical information can be viewed as charts, graphs and tables. Lots of importance has been given to how the information is shown on the charts. They are drawn in such a way, that the user immediately knows where something might be wrong or bad in their software. Alternatively, HTML & Text reports can also be produced. Raw metrics can be exported in XML format, as the user chooses.

In this lab, you will be analyzing four different programs, two of which you have used in class or in other SE courses previously, and one which you will use next year in Operating Systems.

Detailed instructions

1. Download the Cyvis software from http://cyvis.sourceforge.net/index.html. Extract the zip file to c:\ so that the software is located in c:\cyvis-0.9.
2. Download the winPlotter jar file from the course website. (This code was previously used in other SE courses.) Create a new CyVis project for this code and analyze this project. Open the preferences window and set the "high complexity threshold to be 12 and the "moderate complexity" threshold to be 6. When analyzing this code, how many methods have a high complexity? How many methods have a moderate complexity? What is the highest complexity measured? How many classes have methods with high or moderate complexities? Given this data, which methods would you be most concerned with regarding testing and quality, and how concerned would you be? (Note: You can do some analysis on this by selecting the Metrics ->

Generate report text menu item and creating a text report. The report generated is a text file which has the method name followed by cyclomatic complexity and instruction count on a given line. You may need to "massage" the data either with a tool such as Notepad++ or another tool before importing it into Excel where you can do a more indepth analysis of the data.)

3. Repeat step 3 with the Pizzaria project code that you used previously, also available on the website. Answer the same questions with this code.
4. Now open the the JGraph jar file in the same manner. JGraph is an open-source graphics utility used for the creation and visualization of program graphs. How does this rate relative to the questions asked above?
5. Now, as a last effort, download the code for the Virtual Memory Simulator. This tool is used within the Operating Systems course to analyze the performance of virtual memory systems. How does it perform, and what might you be most concerned about from a testing standpoint? Take a look at the source code (available as a zip file. Does the code quality match with the output of the CyVis tool?

3. Lab Deliverables

Working with a partner, submit one report with the following contents:

1. Introduction
   a. What are you trying to accomplish with this lab?
   b. This section shall be written IN YOUR OWN WORDS. DO NOT copy directly from the assignment.
2. Questions:
   a. Answer the questions, in narrative format, posed regarding the usage of the tool and the complexity measures for each of the jar files.
   b. Can you see (or at least potentially see) how this tool / technique could be used on a large codebase for assessment of testability?
3. Things gone right / Things gone wrong
   a. This section shall discuss the things which went correctly with this experiment as well as the things which posed problems during this lab.
4. Conclusions
   a. What have you learned with this experience?
   b. How can this be applied to the design and testing of software?
   c. What improvements can be made in this experience in the future?

This material should be submitted using the course upload system as a single file named <studentnames>_SE2832_Lab10Report.pdf.

If you have any questions, consult your instructor.

4. Preliminary Grading Rubric

	Weight Factor	Rubric Score	
			Introduction - Introduction fully describes what was intended for the lab - Introduction written in words separate from the lab assignment - Introduction written in multiple sentences
			Questions: Winplotter -high complexity? - moderate complexity? - highest complexity measured? - high or moderate complexities class count? -which methods most concern you? - How concerned are you about this code?
			Questions: Bank -high complexity? - moderate complexity? - highest complexity measured? - high or moderate complexities class count? -which methods most concern you? - How concerned are you about this code?
			Questions: JGraph -high complexity? - moderate complexity? - highest complexity measured? - high or moderate complexities class count? -which methods most concern you? - How concerned are you about this code?
			Questions: Virtual Memory -high complexity? - moderate complexity? - highest complexity measured? - high or moderate complexities class count? -which methods most concern you? - How concerned are you about this code?
			Reflection: Things gone right and things gone wrong - Things which went right with the lab fully described. - Things which went wrong with the lab fully described.
			Conclusions - What was learned from the lab described - How can the information be applied to the design and testing of software? - Written in multiple sentences
			Material submitted to website in pdf format.