THE DESIGN AND COMPLEXITY ANALYSIS OF THE LIGHT-UP PUZZLE PROGRAM*

FACULTY POSTER

Douglas Selent
Department of Computer Science
Rivier College
Nashua, NH 01886
978-692-8478
dselent@rivier.edu

The purpose of this project was to use the McCabe IQ tool to analyze source code complexity of the Light-up Puzzle program. The McCabe IQ tool is software used in quality assurance for code analysis. It can allow people to determine if any parts of the source code are unreliable or unmaintainable without the user having to look at the source code. This can save both time and money as well as improve the quality of software code in businesses. Basically what the tool does is convert the source code into a graph, which it then uses to analyze the source code. The features of the tool that were used in this project are the Battlemap, System Complexity metrics, method flowgraphs, scatter diagrams, Halstead metrics, Class metrics, and Object-oriented metrics.

The McCabe IQ tool provides two main benefits to students. The first benefit is that students can learn to debug large volumes of code that is not their own. This is a task students rarely do in school, but may have to do in a work environment. The McCabe IQ tool can be used to quickly identify areas of code that have a high probability of errors. It is unlikely that the student would have to search through source code to find bugs. The second benefit is getting a better idea of what good programming is. Although good programming practices are taught, specific cut-off points for function size and complexity are usually not mentioned or followed. Several functions that are thought to be well-coded are really not. Students who use the McCabe IQ tool can get a better idea of this by programming to the standards of the McCabe IQ software.

The code chosen for analysis is my program of the puzzle "Light-Up". The program consists of roughly 4,000 lines of code with around 20 functions among six classes. It has a GUI for users to actually play the puzzle and a heuristic for attempting to automatically solve any puzzle. The main volumes of code are the GUI setup and event

---

handlers, the algorithms for manipulating the puzzle, and the heuristic that attempts to automatically solve any puzzle.

The source code was analyzed by the McCabe IQ tool and the data was interpreted to identify problems in the code. A new version of the program was coded to fix the problems found in the old version. Data for both versions were compared to each other to identify whether there was improvement between the two versions.

The Battlemap of the original version had 22 (63%) green modules, 1 (3%) yellow module, and 12 (34%) red modules, for a total of 35 modules. The Battlemap of the new version had 101 (81%) green modules, 1 (1%) yellow module, and 23 (18%) red modules, for a total of 125 modules. Although the number of red modules almost doubled from the old version to the new version, the new version was an improvement. The percentage of green modules increased from 63% in the old version to 81% in the new version and the percentage of red modules decreased from 34% to 18%. This indicates significant improvement in the source code between the old and new version. The reason why the numbers are deceptive is because 90 new less-complex modules were created by breaking up a few complex modules.

There was also significant improvement in the System Complexity metrics. The old version had a highest Cyclomatic complexity value of 281, highest Essential complexity value of 125, and highest Design complexity of 98. All three values are awful. The new version had a highest Cyclomatic complexity value of 31, highest Essential complexity value of 13, and highest Design complexity of 18. All three values are slightly bad. Considering that the worst values in the new version are only slightly high, that means the rest of the modules and the program as a whole is good. The problem occurred with the Class Metrics. The original version had no problems at the class level because there were not many functions. When 90 new functions were created from breaking up big functions, it introduced problems for the Class metrics. The RFC and WMC values for the "Dougrithm" class (the class with all the puzzle algorithms) jumped from 12 to 102.

There were two key discoveries made in this project. The first is that blindly using the McCabe IQ tool is not a good way to analyze the code. Users must know how to interpret the data correctly in order to gain the benefits of the tool. The Battlemap was a clear example where the Battlemap of the new version looked much worse than the Battlemap of the old version. At first glance people notice how there is many more red modules in the new version than the old version. They also notice the increase in the number of modules. The percentage of red/green modules is usually unnoticed. Therefore someone might think that the new version is worse when it is really much better. The second discovery is that changes to fix one problem could have unforeseen consequences on other metrics of the program. The changes made to fix the System Complexity metrics had a negative effect on the Class metrics. When there were only a few functions there was no problem. When those functions were broken down into several new functions, the communication among functions dramatically increased. For future work the functions of the "Dougrithm" class could be grouped together by what task they carry out. Then all the function groups can be separated into different classes to fix the problem that was introduced at the class level.

I would like to acknowledge Tim Houle for contributing to the analysis.

REFERENCES

[1] McCabe, T., Watson, A., Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric. *NIST Special Publication 500-235*, August 1996.

The Journal of Computing Sciences in Colleges

Papers of the Sixteenth Annual CCSC Northeastern Conference

April 15-16, 2011
Western New England College
Springfield, Massachusetts

Volume 26, Number 6 **June 2011**