Source Code Quality Education and Improvement Tool

CPSC 490 Project Proposal

Scott Smith

February 2, 2017

1 Abstract

Writing code that works and meets all of the specifications of the project is the first goal of any software developer. However, the quality of that code is of nearly equal importance yet it receives much less of the developer's time. A major reason for this is simply the time pressure that developers are often placed under. Another reason is that many developers have not learned the strategies and techniques of writing high quality code. My project aims to provide a tool to quicken the learning process of writing high quality code.

Many books have been written on this subject, and they provide a great starting point for this project. The problem with these books is that they are not easy to learn practically with. Reading large books full of explanations of techniques, heuristics and explanations is a good first step, but just as with learning to code, learning to improve quality requires practice. My tool will analyze a user's code and point out problem areas and suggestions instead of requiring the user to do this themselves. This way, the user will be able to get immediate feedback on how to better structure, format, and design their code.

2 Comparison to Existing Tools

There are many tools available to developers to help with formatting and easy-to-fix errors. However, these tools are mainly used to streamline and quicken the development process. What these tools fail to focus on are the higher level issues of structure and readability. Since these issues are usually not able to be addressed algorithmically they are simply not addressed. Fixing these issues generally takes a non-trivial amount of time and thought which is not conducive to writing code more quickly.

This is where my tool differentiates itself. Its goal is not to streamline development, but rather to be an educational tool that the developer will use specifically when he or she wants to work on improving the quality of their code. In a normal use case the developer will be writing code without a deadline, such as a personal side project.

3 Deliverables

Internal Representation of Code

The first challenge that will need to be faced is the task of converting source code which begins as simple text files into manipulable and useful internal representations. To do this I will make use of available lexical parsers for certain programming languages. These lexical elements will then need to be parsed into the main structures that will be analyzed. I will need to first create representations for the smallest elements, variables. Then functions, and finally Classes or

Objects. The intention for this internal representation is that it will be flexible enough to allow easy adoption of new programming languages by simply parsing the code in a different way.

3.1 Specific Checks

The goal for this project is to build a base system which can easily be expanded upon. Not only will the internal representation of code allow for easy adoption of new progrmaming languages, I plan to build the analysis system in a way that allows for the easy addition of new code quality checks. During this semester I will implement a few different quality checks which will demonstrate the usefulness and power of this program.

Note: The following heuristics and checks have been taken from *Clean Code* by Robert C. Martin.

3.1.1 Magic Number Alert

Magic numbers are unique values that appear without context and could preferably be replaced by named constants. These numbers should almost always be replaced by constants since it makes editing easier and improves readability. Checking for magic numbers will be a simple process that will work as a good indication that the internal representation of the code and general structure of the program is working.

3.1.2 Variable Name Length vs. Scope

In general, longer variable names are better than shorter ones. However, if a variable is only in scope for a few lines, it makes sense for that variable to have a shorter name. A good example of this is a loop variable which will just iterate over a range of values. Unless this variable is used throughout a function with long scope (defined by the number of lines it is in scope) it is alright for this variable to have a short name. I will implement a check to compare the length of the variable name with the length of the scope that it is in and alert the user when that ration is smaller than a certain value.

3.1.3 Conditional Readability

Often, developers will write conditional statements which include multiple conditional checks or negations. For example:

```
if (timer.hasExpired() && !timer.isRecurrent()) {
```

should probably be encapsulated as

```
if (shouldBeDeleted(timer)) {
```

In addition to this, a conditional check that includes only a negation is much less obvious than one which does not include a negation. I will implement a check for these issues so the user can improve the readability of their conditionals.

3.1.4 Class Cohesiveness

Having cohesive classes is incredibly important for having readable and maintainable object-oriented code. A good check for cohesion is that a class should have a small number of member variables, and each member function should manipulate a large percentage of the member variables. I will develop a way of quantifying cohesiveness and alert the user when they have written classes that appear to be non-cohesive.

3.2 Analysis Interface

There are several possible ways in which this program can display the analysis to the user. The simplest method would be to provide a printout in the form of a text file which includes the relevant code snippets along with a description of the problem and possibly how to fix it. This is how I will implement my program in the first stage. As the actual functionality of the program is more important, I will only work on sophisticating the form of display once I feel satisfied with the core functionality.

Another possible way of displaying the information is to create a graphical user interface in which the user can click through the different suggestions one at a time. In addition to being able to more easily see and understand the suggestions, the GUI can provide an easy to user interface with which the user can set parameters for the analysis. For example, the user can turn on certain checks selectively depending on what they want to focus on improving.

A third way to interface with this program is to create a plugin for a text editor. Atom is an open-source text editor which has countless plugins already written for it. Depending on how this project progresses, it may make sense to convert the program to be streamlined with the user's text editor. This is certainly a stretch goal and will only be pursued if time allows and if it seems that a plugin would be more useful than the other interfaces.

4 Open Source Analysis

As a final part of the project, I will run my program on a large number of open-source codebases to see what quality trends can be found in code that is written by professionals. My hypothesis is that my program will be able to supply many suggestions even with a limited range of quality checks. This would support the motivation for the project which is based on the belief that code quality is often overlooked, even by professionals.