Source Code Quality Education and Improvement Tool
CPSC 490 Project Report

Scott Smith

Spring 2018
1 Motivation

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 resulted in a tool to quicken the learning process of writing high quality code.

2 The Tool

2.1 Internal Representation of Code

The first challenge for this project was to create an internal representation of code that could be used for analysis. While not every programming language follows the same structure, I chose one which should fit most popular languages. In my representation code is made up of Lines, Blocks, Functions, and Classes. When parsed, the internal representation resembles a tree with multiple roots representing the top-level functions, blocks, or classes. Any of these three structures can contain other functions, blocks, lines, or classes in them. In the end, a file may look something like this:

- Class
  - Line
  - Line
  - Function
    - Block: if
      - Line
    - Function
- Function
- Function

With this structure, it is very easy to analyze the code since every part of it is easy accessible and the relationship to other parts of the code is still maintained.

For my project, I implemented a parser for Python. In Python, whitespace is syntactically important and the code is organized by lines. This made parsing relatively simple and made for a good starting point to show the possibilities of this code.
Parsing the code is by far the most difficult part of this project, because once
the internal representation is complete, checks for code quality are very simple.
I did not use any python tokenizer, which may have led to some issues with
particularly strange files. For example, the parser runs into issues on files with
many multi-line strings when they are a part of other multi-line structures (such
as when a string is an element of an array, or explicitly written as an argument
to a function). This is one area where there can certainly be improvement, but
after having run the program on thousands of files, it has failed to parse less
than 3% of the files I have tested.

2.2 Code Quality Alerts

I took my inspiration for the code quality checks from the book *Clean Code* by
Robert C. Martin. I chose four “smells” or heuristics for good code mentioned
in this book in order to showcase the usefulness of clean code, as well as the
power of this tool. The code alert component of this program is built to analyze
a file for each requested quality check, and then report back to the user where
any issues appeared along with a suggestion on how to fix it.

2.2.1 Magic Number Alert

Magic numbers are simply explicit numbers that appear in code. They are a
problem since most of the time it is hard for someone to understand why the
number is what it is. This was a very simple check that only required running a
regular expression on each line of code in the file. It was important to recognize
when a number was being assigned to a variable, however, as that is generally
good practice which should not be alerted.

![Figure 1: Example magic number alert.](image)

2.2.2 Variable Name Length

Variable names should in general have a length which is proportional to the
size of their scope. For example, a loop variable which only has a scope of a
couple of lines can be ‘i’ or ‘idx’, but a variable which is used throughout a large
block of code should have a longer, more descriptive name. To check for this,
all that was needed to do was to use a regular expression to parse out variable
names and associate them with the block of code that they are declared in (this
is done during parsing) and then comparing the length of the name to the size
of the block that it operates in.
2.2.3 Conditional Complexity

Conditionals appear in many parts of code, such as if statements and while or for loops. In general, it is bad practice for a condition to have more than a couple of parts. For example, 'if x and y or c and d' is much harder to read than 'if conditionFunction()'. Wrapping conditions into well-named functions can greatly improve readability. To check for this issue, I had to determine if a block began with a conditional statement, and then simply counted the connective keywords in it. For python that consisted of 'and, or, not, etc.' and can easily be made compatible with other languages by switching out the keywords. If a conditional statement was found to have too many connectives, the user was alerted and suggested to wrap this condition in a well-named function.

2.2.4 Class Cohesion

Classes (or objects) should be tightly cohesive, which means that it should have a small number of class variables, and each member function should access or modify at least one of these variables. A class which is not cohesive can usually be split up into multiple smaller classes to improve readability. This check is the most conceptually abstract, but still very easy to implement because of the way the internal representation works. Class variables are parsed out and associated with the class during parsing, and then each function inside the class can be analyzed to see which variables it accesses. If multiple functions in a class do not access any variables, the user is alerted that their class is not cohesive. An important point to remember with this check is that the functions need to be analyzed recursively since a class function may not access a variable explicitly,
2.3 Extensibility

This project is useful currently only for python programmers. However, I wrote the code in such a way that makes it very easily extensible. Since most programming languages can be thought of as a tree of classes, blocks, functions and lines, all that needs to be done to make this program work for different languages is to implement a new parser. If someone wrote a javascript parser for this program, it could be plugged in immediately and automatically work for the code quality checks that have already been implemented.

Likewise, the quality checks can easily be extended. All that needs to be done is to write the logic of the quality check and then create a new alert object which tells the user how the problem may be fixed. The four checks that I have implemented just scratch the surface of what is possible with this tool and more checks will make the tool truly powerful and useful.

3 Open-Source Code Analysis

As a final part of this project, I ran my program on some of the most popular public github repositories which have a large number of python files. I wanted to see how big of a problem code quality was even in projects written by respectable and talented programmers. The results of my analysis showed that code quality is still a major issue.

3.1 pytorch/pytorch

<table>
<thead>
<tr>
<th>Number of Files</th>
<th>859</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magic Number Alerts</td>
<td>169</td>
</tr>
<tr>
<td>Condition Complexity Alerts</td>
<td>341</td>
</tr>
<tr>
<td>Variable Name Length Alerts</td>
<td>4465</td>
</tr>
<tr>
<td>Class Cohesion Alerts</td>
<td>69</td>
</tr>
</tbody>
</table>
3.2 tensorflow/models

<table>
<thead>
<tr>
<th>Category</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Files</td>
<td>1053</td>
</tr>
<tr>
<td>Magic Number Alerts</td>
<td>686</td>
</tr>
<tr>
<td>Condition Complexity Alerts</td>
<td>297</td>
</tr>
<tr>
<td>Variable Name Length Alerts</td>
<td>5345</td>
</tr>
<tr>
<td>Class Cohesion Alerts</td>
<td>54</td>
</tr>
</tbody>
</table>

3.3 ContextLab/hypertools

<table>
<thead>
<tr>
<th>Category</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Files</td>
<td>72</td>
</tr>
<tr>
<td>Magic Number Alerts</td>
<td>17</td>
</tr>
<tr>
<td>Condition Complexity Alerts</td>
<td>2</td>
</tr>
<tr>
<td>Variable Name Length Alerts</td>
<td>148</td>
</tr>
<tr>
<td>Class Cohesion Alerts</td>
<td>0</td>
</tr>
</tbody>
</table>

4 Conclusions

This project resulted in a basic form of a tool which can be very useful for new and experienced developers alike. Currently it can only be used on python files, but due to the internal representation of code it can be easily extended to support many other languages in the future. In addition, the analysis of open source code has proved that this tool is needed. As can be seen, each of these projects has hundreds if not thousands of issues. While many of these issues will probably be viewed as unimportant by the developer, there is certainly a lot of work that can be done to make this code more readable and maintainable. I believe that my tool can help developers fix their poorly written code and also help prevent them from doing this in the future.

The first step for making this tool more useful is to simply implement more quality checks. There are hundreds in Clean Code, most of which would be easy to implement. Discussions with experienced programmers could also be a good source of ideas for which checks to build into the tool. Secondly, writing parsers for popular languages would greatly extend the usefulness of this tool.

5 References

2. https://github.com/ContextLab/hypertools