Assignment 1

Patrik Ackland

- **Understanding a large codebase.** I find that it is difficult to navigate large codebases to find out where new functionality should be added or what section of the code a certain bug report is referring to. This problem usually occures when you are new to a codebase. As you get more familiar with the code this problem disappears. However, when additional people are added to the project they will encounter the same problem. Even if you know parts of the code you might want to get familiar with another part of the code which introduces this problem again. A large number of files is not the only problem. Design documents that are supposed to give this information might be poorly written or out of date. This could be addressed by automatically generating design documents from the cosebase. The data that can be used for this is the code itself and discussions between developers. Perhaps the original design documents can be used to generate the design and the version control system can be used to update the documents automatically. There are tools that do this to a certain extent. Microsoft Visual Studio can generate diagrams that show how classes interact with each other. However this is usually limited in helping out with a large codebase.

- **Writing good documentation.** I find it difficult to write useful documentation that help other people understand my code. The underlying cause is that I often feel that I am explaining my work twice. Once with code and once with documentation. This leads to not putting as much effort into documentation. Another reason for this is that documentation is usually an afterthought. First I write the code and then I come back to write the documentation. Many of the tools that exist to help with documentation such as autogenerating javadoc in Eclipse only generates annotations for parameters and their names but does not assist in writing the actual documentation. Tools that autogenerate documentation are a way to solve this problem. They could use the code and design documents as data. It might be hard to generate good documentation but it takes less effort from the programmer to correct a few mistakes in otherwise good documentation.

- **Writing useful testcases.** I find that it is often difficult to write test cases that cover every possible case. I often write tests that cover the normal use cases that would get run most of the time. However, the edge cases that are forgotten about might be the ones that end up causing the most trouble later. The most common use cases would cause a bug report quickly. Ways to currently solve this problem is to use profiling techniques to see what parts of the code is being executed. Another way is to randomize input. One way this could be solved is to automatically generate test cases from the code itself and perhaps use other data as a guide to what inputs should be used. I think the reason this has not been solved effectivly is because it is hard to reason about what test cases are needed. The automatically generated test cases need to do a better job than a human would.

- **Lack of effective autocomplete.** When writing code I feel that more effective autocomplete would be useful. Current autocomplete tools can assist with filling in names of objects and functions or give the code for a general for loop. However, I think a lot more can be done to reason about the program that is being written and what code the programmer is expecting to write. If I write a function with the word *sum* in the function name which takes an array as an argument, it could be inferred that I would like the sum of the elements in the array as a result of the function. This kind of autocomplete would assist the programmer with simple tasks and let the programmer focus on more difficult tasks that cannot be generated automatically. This can be extended in other functions to using recently used or recently declared variables. The data for this kind of analysis could be previously written code. With machine learning techniques it could start giving poor suggestions and learn more as it gets used to the programmers style. Many IDEs can automatically generate functions with generalized names and arguments based on a call to a function that does not exist. These are often general and do not help with the actual code. I find it easier to write the function declaration myself since Eclipse also generates comments about the function being autogenerated that I have to remove.