Assignment 3

Essay questions

Question 1.

I think measuring the quality of an arbitrary section of code is possible, but not necessarily important in the process of software development, because "high quality" of standalone code blocks does not imply high quality of the whole project.

There are several metrics that could be used for measuring a block of code: Cyclomatic Complexity, Essential Complexity, etc. The first meaning "the number of linearly independent paths within the source code of application," and the later being the minimum logic points or depth of the code that is necessary for the block to function. (Source) Usually these metrics could tell how concise the code is, and thus in an aspect "decide" the quality of the code. Readability could also be a metric to code quality, which means that the easier for others to understand the code the better. This aspect could involve how clear coder names the variables, how well the code was documented, and how clean the code is, which reflects the code complexity as well. Thus, it is possible to measure the quality of code.

In details, low cyclomatic complexity may imply a "good code," for it will be easier to track the code logic, and thus easier to modify or debug. Essential complexity is the level of branches when cyclomatic complexity is boiled down to minimum. Thus, it takes a step further to describe the complexity of the code. There are other similar metrics that could also measure the code, and generally they lead to concise code and good readability. In my understanding, such code could be considered high quality.

However, measuring these metrics on an arbitrary section of code is not necessarily *meaningful*, even though it is *possible*. The reason is that code in a closed context should be considered as a whole entity, and measurements of the aforementioned metrics on such entity are the metrics that really reflects the quality of code. A block of code alone could have a high "quality" by being concise and clear. However, If such block is repeated again and again in the whole project, the project will end up being repetitive and cumbersome, which will be considered low quality in the end. Another reason is that even though code quality is not limited to the metrics mentioned before. Although these mentioned metrics could be used, there are other metrics that is used to measure other aspects of code quality: the coupling of different parts of the code, the number of interfaces and entity, the number of objects, and some principles like SOLID principles. These principles are not applicable for any arbitrary section of code, because they tend to observe the quality of code in a broader sense. With these said, the quality of an arbitrary code, using only some metrics does not matter that much to the project functionality. Thus, in coding practices, it is more important to focus on the overall quality using comprehensive measurements than to stick with the limited

quality of a certain block of code.

Thus, the conclusion is that the quality of an arbitrary section of code is measurable using certain measurement metrics such as complexity, but these measurements is not as important as measuring the quality of the whole project.

Question 2

My own idea was that the connection of refactoring and quality could be as high as 8 to 9, since the idea of *refactoring* is *changing the design of code without breaking functionality*, which intuitively suggests a better program, better codes. However, after scanning through some pages like Study finds that refactoring doesn't improve code quality, How refactoring impacts code quality and productivity, by Tomasz Korzeniowski, Refactoring--Does It Improve Software Quality?, and Does the Act of Refactoring Really Make Code Simpler? A Preliminary Study, Sokol, et al. , I believe that the points should be deducted a little - but not to a low degree. I would say around 5.

This is because of two reasons: the first is that refactoring may not increase the quality of code when using specific, mathematical methods such as *complexity* to measure, and the other is that refactoring actually increases readability and in result increases maintainability, which could be considered as conceptual code quality as well.

The analysis done by Sokol, et al. shows that, "Quantitative analysis showed that refactoring indeed does not decrease Cyclomatic Complexity." As presented below, the example in the paper shows that refactoring could decrease complexity by extracting and combining certain replicated methods in a fragment, but practical study did not suggest that such changes is obvious in all refactoring: most refactoring in fact increases the complexity, but there are still 23% of documented refactoring decreases complexity while only 12% had the same effect among other commits. This suggests that refactoring may not have a strong association with the quality of code when measured by quantitive metrics. Other articles also support this point.

Example: Refactoring decreases cyclomatic complexity by 1 (Provided by Sokol, et al.)

```
1  public void Employee {
2      ...
3      public void pay(double value) {
4          double factor = 1.0;
5          if (isManager()) {
6              factor = 0.9;
7          }
8          this.money += value * factor;
9      }
10     public void payBonus(double value) {
11         double factor = 1.0;
12         if (isManager()) {
13             factor = 0.9;
```

```
        }
        this.bonus += value * factor;
    }
    public void payOvertime(int hours) {
        double factor = 1.0;
        if (isManager()) {
            factor = 0.9;
        }
        this.money += hours * getHourValue() * factor;
    }
...
}
```

Each method of the code has a complexity of 2 by having an `if` statement, and thus the block has a total of 6 CC. (This also shows that arbitrary block of code can be measured by some metrics to reflect quality, as stated in the first question)

A method called "Extract Method" is applied to decrease cyclomatic complexity:

```
public void Employee {
    ...
    public void pay(double value) {
        this.money += value * calculateFactor();
    }
    public void payBonus(double value) {
        this.bonus += value * calculateFactor();
    }
    public void payOvertime(int hours) {
        this.money += hours * getHourValue()
           * calculateFactor();
    }
    private double calculateFactor() {
        double factor = 1.0;
        if (isManager()) {
            factor = 0.9;
        }
        return factor;
    }
...
}
```

By extracting the `if` statement to standalone `calculateFactor()`, CC is decreased by 1 by decreasing 1 for each original method and adding another method of complexity 2.

However, the other point, "Refactoring increases readability and maintainability," is also important, and should be considered a reason supporting the association of code quality and refactoring. This point is supported largely by Korzeniowski, while stating that keeping the code clean could be helpful in long term project development and maintenance, as the quality of code will degrade as time goes on, and refactoring - keeping the code readable and clean - will slow down the process. This makes sense as redesigning the code will make the code more simple and structured than before. Even if the complexity is not decreased largely, possibly because of unavoidable complexity (essential complexity), it will be easier for others to understand the code functionality, and cumulatively make the whole project concise, and provide better usability of different fragments in future development. This point is also supported by Sokol, et al., even though the research did not cover this point quantitatively.

Thus, the conclusion is that there is association of approximately 5/10 between quality and refactoring, because of the fact of the weak association of complexity and refactoring, and long term influence of refactoring on complexity.