Quiz 06: Refactoring

Due Mar 21 at 11pm **Points** 50 **Questions** 11 **Available** until May 15 at 11:59pm
Time Limit None

Instructions

You **may** use the slides from the lecture and other sources to answer these questions. Please be sure to cite any references but be sure to answer the following questions in your own words. Do NOT simply cut and paste the information from the slides. You will receive a score of 0 if you copy the prose from the slides.

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	980 minutes	50 out of 50

Score for this quiz: **50** out of 50
Submitted Mar 21 at 5:46pm
This attempt took 980 minutes.

Question 1 — 5 / 5 pts

Describe two bad smells in code that suggest you should refactor.

Your Answer:

The bad smells in code that suggest we should refactor are:

Firstly, using too many parameters which in turn make the code difficult to understand. When a method takes in too many parameters, it gives off a nasty odor. Parameters should be limited to one or two elements, however I'm sure there may be occasions where more are required. Limiting oneself to two factors concentrates the method's goal and keeps it from becoming overly complicated.

Secondly, Duplicating the code. When there are instances of duplicated code, it's also a negative smell. When the same logic/code is utilized in two or more regions, it is tough to make changes to this code. Not only would you have to alter it many times, but you'd have to do so in the identical way in each of these spots. This is begging for bugs to enter. The redundant code should be relocated to a method so that it can be easily accessed.

One bad smell in methods specifically, is when it takes in too many parameters. Parameters should be limited to 1 to 2 items, although I'm sure there are instances where more are necessary. By limiting yourself to 2 parameters, you are focusing the objective of the method and preventing it from getting too complex.

Another bad smell is when there are instanced of duplicated code. If there are two or more areas where the same logic/code is being used, it makes it difficult when you have to make a change to this code. Not only would you have to change it in multiple places, but you have to make the same exact change in all these areas. This is just asking for bugs to fly in. The duplicated code should be moved into a method for easy access.

Question 2

5 / 5 pts

What is refactoring?

Your Answer:

The process of refactoring involves reorganizing existing code. Refactoring aims to enhance the software's design, structure while maintaining its functionality. Improved code readability and decreased complexity are two potential benefits of restructuring. By reducing the underlying logic and removing needless levels of complexity, code refactoring can assist software engineers uncover and repair hidden or latent faults or vulnerabilities in the system. These can help to promote extensibility by making the source code more maintainable and creating a simpler, cleaner, or more expressive internal architecture or object model. Improved performance is another possible refactoring target.

Changing the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior

Question 3

5 / 5 pts

Describe the long term refactoring workflow

Your Answer:

When we are preparing to make a major modification or a large-scale restructure to the codebase, we adopt the long-term Refactoring technique. These modifications are usually spread out over numerous iterations over a long period of time, such as several months. Typically, the team will

establish a basic strategy and work toward that aim. Following that, this entails setting aside time during development for restructuring the code and doing all necessary to progress in the direction of the defined goal. As a result, even when new features are introduced, the code base is generally functional.

> Example: Need a complex feature that that can't be added in a single sprint, like adding a new database to the system. It'd require Adding interface layer to support both old and new database technologies, leading to refactoring over time.

Question 4

5 / 5 pts

Describe the Litter Pickup refactoring workflow

Your Answer:

We may come across a piece of code that is particularly untidy or filthy sometimes. It might be that way for a variety of reasons, including the fact that it was developed by an unskilled developer or programmed by an expert individual in a rush, and so on. In any case, it's always a good idea to clean up after yourself and leave the code in a better state than when you discovered it. This is called Litter refactoring rule, sometimes also known as the Boy Scout rule, is a method of replacing problematic code with a cleaner and more elegant alternative.

The Litter Pickup refactoring workflow works like, as the name suggests, picking up litter that you come across on the side of the road on a walk. It's based on the idea of "leaving the code better than you found it," instead of passively either ignoring the problem or assuming someone else with pick it up/fix it. As a developer finds bad code, he/she should fix it right then and there. No matter your abilities or schedule, bad code should always be improved upon, just like litter should be picked up.

Question 5

5 / 5 pts

Describe the TDD refactoring workflow

Your Answer:

Refactoring using TDD is a three-step process:

To begin, we create a test case for the functionality we wish to include in our application. Because the code hasn't been developed yet, it will very certainly fail (It will give red color since the test case failed)

After that, we begin developing logic to pass the test scenario (Keep on rewriting logic and customize it until we pass the test i.e., color changes from red to green).

Finally, after developing the logic, it's time to clean up the code and rework it, making sure that it still passes the test after the restructuring.

Consider the following example: there are two unique hats. Where one indicates that the person wearing the first hat is in charge of restructuring code and the other indicates that he or she is in

charge of creating new functions. It's easier to understand if you consider the TDD situation, where we only refactor with green tests and every failed test indicates a mistake.

TDD refactoring involves starting with writing tests before any code is even written. After seeing these tests fail, write code to ensure the tests pass and run them again. Continue to write code and debug until the tests pass. Once you have finished and are ready to move on to another section, repeat the process by starting with tests. With this workflow, there are only small amounts of untested code at any given time, which leaves less room for errors.

Question 6

5 / 5 pts

How would you respond if our boss asks why you are spending time refactoring working code rather than building new features?

Your Answer:

I would explain to him why refactoring is required and what advantages it provides. There are several reasons why I could go into further detail, such as making the code cleaner and hence more understandable. As a result, if another developer wants to work on it, he will be able to comprehend and work on it quickly. Other advantages of making it flexible include the ease with which any feature may be introduced because all technical debts were paid off as we refactored the code. Modularizing the code would save time since the same piece of logic could be reused and made accessible throughout the project, making the code more efficient and reusable. If the code hadn't been refactored, overhead expenses would have increased because of the increased efficiency, saving a lot of resources in terms of people and time.

I would say that the work I'm doing now may not seem beneficial at the moment, but it is actually doing wonders for us in the future by making things more organized and readable. Once refactoring has been completed, building new features will come much quicker and with more ease. It will also be much easier for more people to work on project since it will be much more understandable. Trust me!

Question 7

5 / 5 pts

Describe the preparatory refactoring workflow

Your Answer:

Martin Fowler once said, **"Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior".**

The workflow of preparatory refactoring can be better understood with an example.

Consider the following scenario: we need to add a new functionality to an existing program. We find it tough to add the appropriate piece of code since there is a lot of technical debt while adding the new functionality. Preparatory refactoring can be utilized in this situation.

The following are the steps in its basic workflow:

Firstly, see if the code can be readily added or if it must be restructured. Secondly, if not, we'd have to restructure/refactor the code into a manner that readily accepts modification or allows for the addition of new skillsets without changing the application's real essence. Finally, after the code has been refactored, we can add the necessary new functionality with ease.

References: https://martinfowler.com/tags/refactoring.html (https://martinfowler.com/tags/refactoring.html)

Example: need to add a new feature but it's easier to add the new feature after making changes to existing code. Sometimes it's better to go backward to go forward

Question 8

5 / 5 pts

What is technical debt? How is it paid off?

Your Answer:

We always want to begin developing code with a decent design, but to satisfy the needs, we must compromise on the code structure or coding standards. This code may meet the requirement, but it will slow down the program in the long term since it is too hard to alter, and changing inflexible code is expensive. We'll need to put in more work because of poor design, shortcuts, or attaining the desired outcome through coding that isn't up to par. This is referred to as technical debt. So, if we think about it, it's like a financial obligation that we must repay. This code may be paid off by removing poor smells and rewriting the codebase on a regular and short-term basis. Duplicate code, needless complexity, big classes, too many arguments, and other foul odors should all be avoided.

Additional development, testing, and maintenance effort. Caused by: Bad design, Taking shortcuts, Not implementing the “right” solution throughout the lifecycle. Can be paid off by refactoring to fix smelly code.

Question 9

5 / 5 pts

Describe two benefits of refactoring

Your Answer:

The two benefits of refactoring are:

- Refactoring aids in the discovery of flaws that may have been concealed behind stinky code. You could find issues you hadn't seen before, or even better solutions to existing bugs, by going through and tidying things up.
- Refactoring makes it easier to read and comprehend programs. You can make it easy for someone who has never seen the code before to grasp it faster and more accurately by adjusting repetitious code and improving the naming conventions.

One benefit of refactoring is that it makes programs easier to read and understand. By making changes to either repetitive code or poor naming conventions, you are making it easier for someone who has never seen the code before to be able to understand it faster and with more accuracy.

Another benefit to refactoring is that it helps you find bugs that may have been hidden under smelly code. By going through and cleaning things up, you may notice bugs that may not have been found before, or even better solutions to existing bugs.

Question 10

5 / 5 pts

Describe the two Hats of Software Development.

Your Answer:

Two hats in software development is a metaphor that stretches back to the earliest days of refactoring. The two hats indicate the fact that the person wearing the first hat oversees restructuring code and the person wearing the second hat oversees creating new functions. It's easier to understand if you consider the TDD scenario, where we only refactor with green tests and every failed test indicates a mistake has occurred. Refactoring is a tiny yet behavior-preserving modification, but any other change in the code is considered adding functions and is no longer considered refactoring. By changing the hat, the person is wearing during refactoring, a person can

switch between refactoring or adding functions. He/she can switch hats as frequently as every few minutes, but the sole stipulation is that only one hat can be worn at a time.

1.Adding functionality to the system: Not changing existing code, Adding code, Adding tests (may break existing tests)
2.Refactoring: Not adding new functionality, Not adding tests, Not changing tests (unless necessary), Small, quick, behavior-preserving changes

Question 11

0 / 0 pts

“I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination. I further pledge that I have not copied any material from a book, article, the Internet or any other source except where I have expressly cited the source.”

Correct!

- (●) True
- ( ) False

Quiz Score: **50** out of 50