Quiz 06: Refactoring

Due Mar 9 at 10pm **Points** 50 **Questions** 11 **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	8,655 minutes	0 out of 50 *

* Some questions not yet graded

Score for this quiz: **0** out of 50 *
Submitted Mar 9 at 5:42pm
This attempt took 8,655 minutes.

Question 1

Not yet graded / 5 pts

Describe the preparatory refactoring workflow

Your Answer:

Preparatory refactoring workflow means that whatever changes are needed in the code, first fix up them all and then it would be easier to add a new feature in the existing code. It might be a better approach to step-back some distance and choose some alternative path to reach the destination.

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 2

Not yet graded / 5 pts

What is refactoring?

Your Answer:

- Refactoring means to change the structure of the code for the better and easy understanding.
- Refactoring does not let anyone know that the part of code is being changed.

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

Question 3

Not yet graded / 5 pts

Describe the two Hats of Software Development.

Your Answer:

- First hat refers to **adding functionality** to the system - It means that new code is added with the previous code and also with this new test cases gets added.
- Second hat refers to the term **Refactoring -** It means the change of code without letting know the user about the change. User should not feel the change of code. Refactoring provides easy understanding of the code.

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 4

Not yet graded / 5 pts

Describe two benefits of refactoring

Your Answer:

Two benefits of refactoring:

- **Efficiency:** Refactoring helps in improving the efficiency of the code and gives better result. It reduces the efforts that needs to be applied in the future change of code . Refactoring helps in faster running of code.
- **Better Maintainability:** When ever any software gets developed then it might have a weak codebase structure. With the help of refactoring , it helps to develop a better codebase and structure, so that in future it gets easy to develop it further.

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 5

Not yet graded / 5 pts

What is technical debt? How is it paid off?

Your Answer:

If a company has a deadline for delivering a product to the client, then they will expedite the work on building software which might lead to a bad product or some features might not get implemented in a better way. Because of this, this features needs some rework on that and that rework is known as **technical debt**.

Technical debt gets paid off with the help of Refactoring . One should work upon the time management and strictly follow the sprint timeline. They should be able to complete their work on time. In this ways, technical debt gets paid off.

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 6

Not yet graded / 5 pts

Describe the TDD refactoring workflow

Your Answer:

TDD refactoring workflow:

This workflow says that , we need to Write test cases at the first and run that. Then after writing some part of code , rerun the test once again. Repeats the same process until the bugs get fixed. Run test cases until all of them gets passed successfully.

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 7

Not yet graded / 5 pts

Describe the planned refactoring workflow

Your Answer:

Planned refactoring workflow means that one new attribute gets added in the workflow called "time". This attribute keeps track of the work of the project. It helps us to complete the project on time. There is a timer in it which shows the time for refactoring.

Include time in the project schedule time for refactoring. Hardening sprint is an example

Question 8

Not yet graded / 5 pts

Describe the Litter Pickup refactoring workflow

Your Answer:

Litter Pickup refactoring workflow means that the user first reads the code and the user find the bad code or the bugs.

Then at the same time the user fixes the bad code and gives a better solution. User fixes at the same time because it will be easier to add new features in the future.

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 9

Not yet graded / 5 pts

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

Your Answer:

Two bad smells:

Long method: Remove duplicacies of variables. Write the method as efficient as you can. Declare an object instead of passing many parameters. Write shorter conditional statements.

Large class: Avoid unwanted methods and variables. Try to use the variables globally rather than declaring it for each method. Refactoring techniques are extract class, subclass and the interface.

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 10

Not yet graded / 5 pts

Describe the long term refactoring workflow

Your Answer:

Long term refactoring workflow means that it will take a long time to come up with a solution. It says that firstly clean up the existing code so that it gives a better readability and then after write the new code from the scratch by adding some new layer or interface that supports both, the old one and the new one .

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 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: **0** out of 50