Assignment 3 of Object-Oriented Design and Analysis

Name- Saurabh Chawla

Student-ID- 0838092

Question 1: - On p. 138 we make the bold statement that "duplicate code is a bad idea!" Do you agree? Why or why not? What can go wrong if you duplicate your code and why?

Answer – As a matter of fact, page 138 has proved that "duplicate code is a bad idea!" by justifying the affirmation with several strong arguments. A few problems caused by duplicate code can be seen in the software development process, especially concerning error management, maintainability, and productivity.

1. **Difficulties in Maintenance**

   Code duplication significantly increases the maintenance burden. For any change in logic, replicate code at various places must be changed. Some of them might be missed out, which could result in inconsistency in the behavior of the system. For a larger codebase, such inconsistency might lead to surprising behavior or subtle hard-to-trace defects.

   Suppose the following scenario: you find some defect in certain function, which has been duplicated or other modules. You must find each place the duplicated code has been present and fix it. If you miss one, then the problem would remain there making the software unstable. Another problem – also with feature enhancements is that implementing any enhancement more than once is time-consuming and error-prone.

   An example which was discussed in the class by Mathew Sir was that code was copied and pasted for a customer 25000 to 30000 times. Developers refactored it into a class later; this cleaned up not only thousands of lines of unnecessary code but also made the application much faster and easier to maintain. It acts to reinforce how important removing redundancy is when attempting to keep systems manageable.

2. **Greater Use of Resources**

   More memory and physical code space are consumed by duplicate code. The codebase size of one code piece increases with more its appearance. This adds weight, complicates management, and may cause the system to crash due to potential performance problems.

   From a computational point of view, a certain logic is executed continuously instead of being embedded once into one reuseable function or class; the system will allocate resources for each copy of that very same logic. This processing of duplicate code adds unnecessary strain and can, under certain circumstances, result in performance degradation; hence, resources are under high.

Developers then used an example to refactor out code duplication with little work further than creating a reuseable class to increase the efficiency of the system. In general, when redundancy is reduced, efficiency of a system increases as less resource needs are necessary.

3. **Mistake Spreading**

   But one of the major issues with duplicated code is that if there is some kind of error in one of the copies of this code, it affects all the other copies, potentially leading to several failure points. For example, duplicated code may have a defect, and you fix the bug but don't update one of the copies. Then, that part of the system does not work correctly.

   It means, suppose some defect that was originally overlooked is copied with the original code; much later, when that defect is to be fixed, all its copies must be found, which can take much time and be error prone. Moreover, the developer should keep in mind that all the instances are identical, which again becomes an added hurdle in adding new functionality or updates.

   As was explained in class lecture in the given example, could only be done provided the developers knew the number of times the same logic had been duplicated. This represents one of the potential dangers that can be taken away with good design practices because the duplicate code can spread issues throughout the entire codebase.

4. **The quality of code and its readability**

   Readability, especially when in teams, is a very important feature of code quality. The more repetition and litter that exist in the codebase, the more difficult it will be to read or understand the code. Certainly, it is hard for the developers to pay attention to the actual functionality since they must go through a lot of duplicate code.

   Encapsulation and modularization may be used to split complex systems into manageable parts. You could ensure the code would be more readable, reuseable, and understandable by consolidating redundant functionality. A new developer joining the team will sooner be productive and get familiar with the codebase easier, as he won't need to read the same logic many times.

   The example used in the lecture for illustrating the idea of readability was the client complimenting the system on increased performance after superseding unnecessary code. Clean code improves performance, which is easier to maintain and expand by present and future developers.

5. **Modularity and Refactoring**

   Good design principles, on the other hand, encourage refactoring code into reusable components such as classes, functions, or methods. This decreases code duplication.

Modularity is encouraged, with the attendant characteristics of well-organized software. Modular code is more testable, easier to debug, and less problematic to maintain.

This also assists in wrapping code into a function or class instead of copying and pasting it more than twice. This has to do with the principles of object-oriented design that advocate for bundling functionality into classes that are system-wide reusable. It enhances maintainability, enables scaling and reduces repetition within the code.

6. **Elegant Coding and Optical Design**

   Aside from being helpful, code duplication is, in fact a matter of taste, as lecture notes humorously put it, duplicate code is “ugly”. Indeed, one characteristic of a great programmer is near well-organized code. In addition to testifying about poor coding habits, duplication results in cluttered, disorganized code bases.

Question 2: - Removing duplicate code is one example of a programming practice called refactoring. Search for definitions of refactoring your code in various resources. Write down those you find and where you found the resources. Which do you think is the best one and why?

Answer Refactoring is the art of improving the internal structure of the current code without even changing its external behavior. Refactoring is for enhancing readability, maintainability, and cleanliness of the code. In refactoring, the programmer can refine the architecture incrementally and improve the code while keeping its functionality intact. This makes the code simpler and more effective.

The following many important aspects illustrate refactoring:

1. **Removing Duplicated Code: - An essential example of Refactoring**

   One of the most useful refactoring techniques that were discussed during the class lecture is removing duplicated code. As described: -

   “**If you need to copy your code more than twice, you should refactor it into a function, method, or class.**"

   This comment points out that redundancy should be eliminated from the codebase. Similar code needs to be refactored to a reusable component to have the following advantages:

   - **Easy Maintaining: -** Repetitive code requires all its instances to be updated in the case of future changes, which opens the door for mistake occurrence. In the case of refactoring into one function or class, the chance of introducing error or inconsistencies reduces since this code only needs to be changed once.

   - **Improved Modularity and Reusability: -** Refactoring cleans up methods, classes, or functions that can later be utilized elsewhere in the code. Therefore,

refactoring helps in fostering modularity since it allows for more organized code, which is easier to extend and modify and consistent with good object-oriented principles.

This is a good example of how the programmers on this project were able to eliminate 25000 to 30000 lines of duplicate code by developing a reusable class. In this case, system performance significantly improved, system maintainability was also improved. This is an example of how refactoring, or the process of eliminating redundant code, makes code more readable and easier to write and maintain.

2. **Iterative Improvement and Agile Development**

The presentation also agrees that iterative processes, in most of the agile development Methodologies, consider refactoring an important part of these processes. This it says:

"**We've gone through another iteration. Requirements, Design, Code, Test. We took on a manageable chunk of work. We delivered a working system.**"

This iterative approach to development concerns small, achievable enhancements to the code. The agile way of thinking fits refactoring well, since it too works in small and gradual steps that improve the internal structure and quality of the code without changing the output of the overall program. By doing this, developers may adjust the program to evolving requirements; hence, refinements are progressive.

It is reflected in the presentation that this incremental approach has been adopted since the 1980s. Agile development relies, through constant refactoring, on enabling developers to create high-quality software that can evolve organically over time without major redesigns.

3. **Information Expert Pattern and Refactoring**

Another valuable concept extracted from your presentation and notes is the information Expert pattern. During the process of refactoring, this pattern assigns responsibilities to classes that have the information needed to carry out these responsibilities.

Using the application Dog Door for example,

"**We applied the information Expert pattern to our design and moved the responsibility for automatically closing the door to the Dog Door class.**"

It is now the responsibility of the Dog Door class to close the door. This simplifies the design because the component that has the relevant data is responsible for the decision. If either the Bark Recognizer class or the Remote were responsible for closing the door, closed door logic would become overly coupled and harder to maintain. This kind of refactoring makes sure each class has a unique responsibility.

4. Preventing Duplication Issues

The lecturer mentioned in class the dangers of having redundant code and the necessity of refactoring. This makes sense because:

- **Error Propagation**: Any time there's duplicated code, bugs will start manifesting in every single instance of that code. If one desires to make modifications, all instances must be updated in turn. Usually, the instances that were missed lead to bugs and erratic behavior.

- **Performance Issues:** Running duplicated code for the second time is too expensive. As it was clear from the example, though the change was theoretically simple and straightforward, once the engineers refactored the duplicated code into a reusable class, the improvement in the performance of the system was radical.

- **Readability and Quality of Code:** The code becomes redundant, cluttered, and is less readable for junior engineers. Refactoring to reduce redundancy helps in maintaining structured and clean code, say the presentation.

  Instructor's notes: -

  "**Find and replace. Don't copy and paste your code.**"

  That is clear advice to rewrite the code into reusable components and not to copy. In that way, programmers save themselves from the hassle of managing many copies of the code, thus making it more understandable and maintainable.

  Based on the lesson materials and presentation, the following is the best explanation of refactoring:

  "If you need to copy your code more than twice, you should refactor it into a function, method, or class."

  This works as an analysis because

  1. **Practical Advice: -** It gives a developer a very specific and actionable cutoff point. Any code that is reused more than twice should be in a reusable component. This basic concept is very easy to apply and understand.

  2. **Modularity: -** Duplicated code can be factored into methods or classes to yield a maintainable and scalable design.

  3. **Ease of Maintenance: -** When code is encapsulated in one place, rather than being spread across multiple instances, changes are easier to deal with. There's less likelihood of a defective code base with bugs because of this approach.

General software engineering ideas support this view of refactoring given in the course notes through placing a strong modularity, maintainability, clean code, and iterative improvement in the development of robust and flexible software.

Question 3: - Assume that you have just found a time machine and you go back to the ancient days of 1993, before Jeff Bezos started Amazon.com. You have all the knowledge from today, and you know what a great idea like Amazon.com can turn into. This is your chance to become famous and rich. Jeff Bezos will just be an entry in a phone book if you can come up with a new online bookstore. Sure, there are other things that can be sold, but start out with something simple. Identify 3 major use cases (names only) and their primary actors for your online store. As an additional constraint, each use case must have a unique primary actor. Draw the results in a UML use case diagram.

Answer- Based on your PowerPoint presentation, select which use cases fulfill certain user goals by focusing on different actors who would be using the system. Three main use cases are: -

1) **Search for books**

- **Primary Actor**: - Customer

- **Description**: - Through the "Search for Books" use case, the user is given a chance to look around in the online library for books of interest. A customer can develop a focused search through various filters on title, author, genre, and keywords. The main purpose for a customer when entering an online bookshop is to identify the appropriate product; this use case satisfies this objective.

- **Importance**: - The bookshop would identify the customer's main point of contact with the system by ensuring that e-commerce's most important feature, product discovery, is realized.

2) **Process Payment**

- **Primary Actor**: - Payment Service

- **Description**: - The "Process Payment" use case represents the customer payment transaction when buying books. The main actor is the payment service, which allows the transaction to be processed securely, whether by verification of credit cards or confirmation of a payment.

- **Importance**: - A dedicated payment solution ensures that financial processing for any type of transaction is done securely and effectively. Payments form a part of every e-commerce platform. The Actor-Goal view as per the lecturer notes describes how this led to a separation of business logic and finance processing concerns of the bookshop.

3) **Manage Inventory**

- **Primary Actor**: - Store Administrator

- **Description**: - Use case “Manage Inventory” lets the shop administrator control the inventory of the bookstore: Update of stock levels, remove goods that are no longer carried and add new books to the catalog. It’s the responsibility of the shop administrator that the product data is correct and up to date.

- **Importance**: - This ensures that inventory management responsibility falls on the store administrator to keep the catalog up to date and relevant by effecting proper product lifecycle management.

Applying for Use Case guidelines: -

The use case guidelines help us develop strong goal-driven use cases. They apply to those use case in the following ways: -

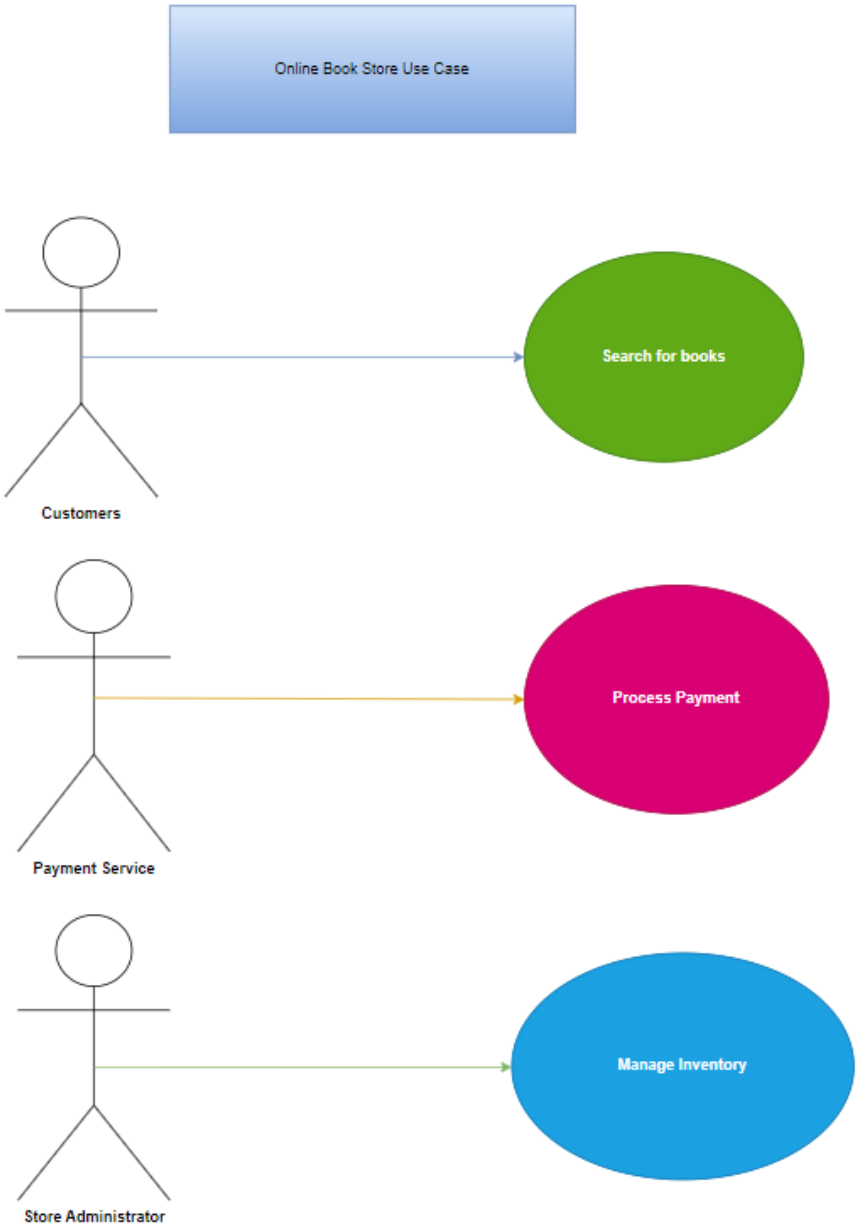
1. **Essential Approach**: - Each use case is written in a simple, UI-free style which emphasizes the goal of the actor. That is, “Search for Books” mostly conveys the objective of the customer to find a product rather than describing the UI elements like search buttons.

2. **Limited Use Cases**: - The use cases identify the primary interaction and are concise. Identifying a book, making a payment, and managing the inventory are their three equal yet different objectives.

3. **Black-Box Use Cases**: - These use cases define the system’s responsibilities but do not outline how the system will accomplish them. For example, “Process Payment” defines that there is a requirement to process the payment but does not define how it will be done.

4. **Actor and Actor-Goal Perspective**: - Both actor and actor-goal aspects for each use case are investigated. This with the intention of finding out what the performer is attempting to achieve: -

    - Customer searches for a book.
    - The Payment Service aims at safely processing the payment.
    - The store administrator wants to manage the stock efficiently.



Question 4: - Create casual format use cases for all the use cases chosen in question 3. Use the example from the PowerPoint slides as a guide. Slide 44/45 in the Chapter 2 PowerPoints contains a description of a casual format use case.

Answer: - Detailed Casual Use Case for Online Bookstore-

1. **Use Case:** Search for books

Primary Actor: Customer

Description: This use case, "Search for Books," allows customers to view all the books offered in the online bookshop by inserting any search parameter, including but not limited to book title, author, and genre. This tool will help customers quickly identify books they want and will make it very easy and efficient to complete a purchase.

Main Success Scenario:

1. Customer visits the website:
    - The customer opens his browser and logs onto the web store.
    - The system boots up to the home page displaying a search bar, categories and featured books.
2. The Customer initiates the Search:
    - The user inputs a query in the search field-for example, a book's title, its author, or genre-followed by clicking on the search button.
3. System processes the request to carry out the search
    - System, upon processing the word searched for, it runs a comparison with the available list of books.
    - The system retrieves the book that match the searched criteria.
4. System displays the result of the search
    - System displays a list of books matching the keyword typed by the customer.
    - A list of basic details about each book, including the title and author, price and current availability.
5. Customer Chooses a Book:
    - The customer searches for the book and then clicks on the book to retrieve more information.
6. System Displays Book Info:
    - The system displays all information on the book chosen-including book cover picture, complete title, author, description, price, customer reviews, and available formats, such as hardback, paperback, eBook.

Alternate Scenarios:

- **2a**: **No Search terms were entered.**
    - In the event the customer clicked the search button without having filled in something in the search box, it should return a list of some popular books or suggest browsing categories such as “Best Sellers” or “New Arrivals”.
    - It could also further prompt the user to make a more specific asking them to enter keywords.
- **3a: No Books Found:**
    - In the event of no books matching, the system displays “No books found.” Please try other keywords combinations.”
    - The system suggests some of the often searched for phrases or recommended search terms.
- **4a: Network Problem during Search:**
    - If there is an issue at the network end and it is unable to process the search request, then the system flashes “Unable to complete the search at this time”. Please try again later or check your internet connection.”

2. **Use Case Process Payment**

   Primary Actor: Payment Service

   Description: The “Process Payment” use case represents the transaction processing, securely initiated whenever a customer intends to make a purchase. This use case is very important in ensuring that customer orders are correctly processed, and their payment information is correctly verified through the service known as a Payment Service.

   Main Success Scenario:

   1. Consumer Checkout:
      - The consumer has placed one or more books in the shopping basket and clicks on the “Checkout” button.
   2. System Requests Client Information:
      - The system prompts the user to either log in or continue as a guest.

- The client provides contact information along with an address where the item may be delivered.

3. Client Provides Payment Details:

- Payment information is asked like credit card number, expiration date, and CVV when the consumer feeds the payment details in the system.
- The client, after entering all the details, click the "Pay Now" button.

4. The Payment Service Handles the Payment:

- Sending the details of the payment for processing, the system forwards the information to the Payment Service.
- The payment service will process the transaction and verify the payment details in accordance.

5. System Shows Payment Confirmation:

- In case of successful payment, a message "Your payment has been successfully processed." Is shown to the consumer by the system itself.
- The system generates an order summary and assigns an order confirmation number. These are then sent to the customer's email address.

Alternate Scenarios:

- **3a**: **Payment Information are Not Correct:**
  - System shows a notification, "Invalid payment information" in case the customer provides wrong payment information – incomplete card number or expired. Please check your card details and try again.
  - A prompt requesting the client to provide new and correct payment details.
- **4a: Payment Rejected:**
  - The system notifies the consumer that the just provided payment was declined because the Payment Service refused to accept it due to any reason such as lacks funds, card banned, etc. Please choose other methods of payments here.
  - The client has two choices here. He or She can either choose to proceed with Pay Pal or use another card.

- **4b: Failed Communication with the Payment Service:**
    - If communication with the payment service breaks down at any point, the payment system displays, “Unable to process payment currently due to technical issues. Please try again later.”
    - The client should be advised to try again later.
- **5a: Partial Payment Completions:**
    - When the transaction is halfway and it is an issue, for instance, a network failure, the system flags the transaction and notifies the client for them to contact customer care for further assistance.

3. Use Case: Manage Inventory

Primary Actor: Store Administrator

Description: The “Manage Inventory” use case enables the shop administrator to keep a record of what books are in stock and up to date by managing the inventory of the bookstore, including adding new titles, updating the inventory quantity and deleting books no longer carried.

Main Success Scenario:

1. Login of System Administrator
    - The administrator of the shop logs into the inventory management system using his/her log-in credentials.
2. Administrator Navigates to Inventory Management
    - The system displays an admin dashboard with various tools to manage user accounts, orders, and books.
    - The administrator clicks the “Manage Inventory” option
3. Administrator Selects Option:
    - The administrator decides on adding a new book, updating an old one, or deleting a book from the inventory.
4. Add New Book:
    - If the administrator selects and clicks “Add Book,” he/she is asked to enter the details of the title of the book, author, genre, price, and quantity of the book.
    - The system updates the new book details in the inventory.

5. Edit Book Information:

   - Admin clicks on “Update Book,” searches the book by title or author, edits the fields concerned - say price and quantity, updates the changes.

   - System: Inventory information shall be updated accordingly.

6. Remove a Book from Inventory:

   - When clicked on “Remove Book,” the administrator locates the book and confirms deletion.

   - System updates the book in the inventory as “Discontinued”.

Alternate Scenarios:

- **4a**: **Same Book Entry:**

  - The system asks the administrator “Book already in inventory” if he chose to add a book that is already in stock from the same author and title. Would you like to update the current entry?”

  - Instead of creating a duplicate about the same book, now the administrator could choose to update information regarding the presently existing book.

- **5a: Out of stock Notification:**

  - If the stock is reduced to zero, the system will automatically mark a book as “Out of Stock”.

- **6a: Incorrect Book Removal:**

  - If there is some sort of error in removal, for instance, this book being pending orders in some customer’s purchase lists, the system flashes a message to the administrator. “This book cannot be removed as it is associated with pending orders. Please delete after fulfilling or canceling the orders.”

- **2a: Wrong Attempt to Login:**

  - The system is supposed to display an “Login failed” error message in case the store admin fails to log in using invalid credentials. Please check your password and username.”

  - The administrator can try again prior to being locked out, for security purposes up to three times.