**IT5016 Week-7 Lab:1**

TASK:1

1.The Software Developmental Life Cycle defines as the process which is highly structured and is used for the creation of software which also includes the further maintenance of the software. Each stage of the software is highly defined and after the completion of one stage only further could take place.

The main significance of this life cycle is it ensures the quality of the software.  
The Software development Life Cycle holds 6 stages which are Planning, Analysis, Design, Implementation, Testing, Integration and Maintenance

**Phase:1 Planning**

This is the primary stage of the life cycle which is considered important where companies enter to figure out solutions for their issues in the form of building software and planning it in a way by keeping the requirements in mind and the resources given by the company.

**Phase:2 Analysis**

This is the stage where there is a little idea about the software and the final product is still needs to be discussed within the IT Support and the management of the company. The outcome of this stage is coming up with the exact demands of the company from software developers.

**Phase:3 Design**

This stage of the life cycle is about the appearance of the application and also the internal functioning taking place behind the display screen. Apart from this, various feedbacks are called upon to improve the design of the software by software developers and the outcome for this is the perfectly built model.

**Phase:4 Implementation**

The phase of lifecycle where a general code is built by keeping 2 things in mind which are the superior quality code and the other is that the code is built in a less time. The fully functioned code is extracted out by keeping it up to the expectation of the management and the growth of the product is monitored.

**Phase :5 Testing**

This stage of life cycle checks over all the other stages of the process where code is checked line by line which does not effect the quality of the code and fixed by the developers. Ensures to provide quality and accuracy, which functions properly and efficiently.

**Phase:6 Maintenance**

Within this stage, the software is ready to roll it down to the users. However, in future this phase is also considered important as this the user can easily call up to the developers when they face any issues regarding the use so this stage goes on.

2. SDLC in Movie Rental systems.

Applying **SDLC (Software Development Life Cycle)** to the given **Movie Rental System** involves improving the entire process of planning, designing, developing, testing, and maintaining the system. Here's how SDLC principles can be applied step-by-step:

**1. Requirement Analysis**

In this phase, we gather the requirements and define the objectives of the system. For the Movie Rental System:

1. **Functional Requirements**:
   * The system should allow customers to rent and return movies.
   * It should maintain a list of available and rented movies.
   * The system should allow customers to add or remove movies from their rented list.
   * The system should provide functionalities for adding new customers and movies.
   * The system should prevent customers from renting already rented movies.
2. **Non-functional Requirements**:
   * The system should be user-friendly and simple.
   * The system should handle invalid input (e.g., when a movie or customer doesn’t exist).
   * The system should scale for larger customer and movie bases.

**2. Design**

The design phase focuses on creating a blueprint of the system, considering both high-level and low-level architecture.

* **Class Design**:
  + The classes Movie, Customer, and Rental Store are already well-structured. However, improvements could be made.
  + **Design Enhancements**:
    - Add **data validation** in the Movie and Customer classes to ensure data integrity (e.g., checking the format of the year or customer names).
    - Introduce a **Unique ID** system for movies and customers to avoid errors with duplicate names.
* **Error Handling**:
  + Implement error handling when movies are not available, when customers are not found, and when invalid input is provided.
* **Modularization**:
  + Move the menu and main logic into separate modules, or refactor the user interaction part for better scalability and readability.

**3. Implementation**

This phase involves writing the actual code based on the design decisions.

* **Current Code**:
  + The code already handles basic functionality for renting, returning, and listing movies, but it lacks validation and exception handling.
* **Improvements**:
  + Add **input validation** to ensure the year is a valid integer, movie titles are valid strings, and customers exist in the system before interacting with it.

**4. Testing**

Testing is a critical part of the SDLC to ensure the system behaves as expected in all scenarios.

* **Unit Testing**:
  + Use a testing framework like unit test or py test to write unit tests for every method in the Movie, Customer, and Rental Store classes.
  + Test for:
    - Renting an available movie.
    - Attempting to rent an unavailable movie.
    - Returning a movie that was not rented.
    - Adding and finding movies/customers.

**5. Deployment**

Once the system is thoroughly tested, it is ready for deployment.

* **Packaging**:
  + Package the system as an executable script or an installable Python package using setuptools to make it easy for others to use.
  + Ensure the system can be deployed in different environments (Linux, Windows, etc.).
* **Version Control**:
  + Use Git for version control to track changes, manage features, and collaborate with other developers.

**6. Maintenance**

After deployment, continuous maintenance ensures that the system remains functional and relevant.

* **Bug Fixes**:
  + Monitor the system for bugs or issues in production and create patches to fix them.
* **Feature Upgrades**:
  + As the business grows, add more features such as:
    - Support for multiple stores.
    - Online rentals with a user interface.
    - Handling larger numbers of movies and customers by switching to a database backend.

**Task:2**

**Mention the Software Design Principle:**

There are 10 Software design principal.

1.**KISS** : Keep it simple and, stupid

This principal demands the program to be simple and easy by making it easy for the users to access and use it in a best possible way.

2.DRY : Don’t repeat yourself

This principal demands the developer to create the code for once and try not to duplicate it which will further lead to errors.

3. Open/Closed principal

In this principal developer can extend the course according to the demand but could not modify the existing code.

4.Composition> Inheritance

This principal demands not to give preference to Inheritance, rather than Composition is considered important due to which complexity of the object behaviour is avoided by creating instance of object with individual behaviour.

5. Single Responsibility Principal.

This principal demands that the one class should perform only function at the time as it causes problems and errors and should only be responsible for one factor at a time.

6.Sepeartion of concerns

This principal is the independent principal too if said in other words, where one code is not dependent on the other code where ever there is an overlap. One code needs to be finished at a time if any other needs to take place.

7. Yagni : You ain’t gonna need it

This principal strikes on the simplicity of the code which defines that no extra things should be added to the code as it increases the chances of errors in a code. Instead of adding extra it is better to focus on the requirements.

8. Avoid Premature Optimization

This principal demands to make the code applicable for running first than it is the second concern to make it faster or better.This helps to boost upto the algorithm tendency.

9.Refactor, Refactor, Refactor

This principal requires the code to be clear and tidy any type of mess is not required and if there is any mess it could result into errors and confusion.

10.Clean Code > Clever Code

Clean code is preferred which further leads to less confusion and clever code basically is performed to flex the knowledge.

2.Examine the code and identify where the software principal codes lack.

1.The class UnstructuredCode has a method xyzzy that performs a basic addition operation, which is overly complex in its naming and purpose. Additionally, the legacy\_function seems unnecessary, as this could be part of a well-structured class.

2. The Calculator class has two methods, add and multiply, both performing operations in a similar way. This is code duplication.

3. The InflexibleShape class and its subclasses (Circle, Square) have empty methods, making it hard to extend the functionality for other shapes without modifying existing code.

4. The Circle and Square classes inherit from InflexibleShape but provide no implementation of calculate\_area. This shows unnecessary inheritance, as inheritance should only be used when behavior can be shared.

5. The UnstructuredCode class performs addition, which overlaps with the purpose of the Calculator class. It doesn’t have a single responsibility.

6. There's overlap between the functionality of the UnstructuredCode class and the Calculator class. This violates separation of concerns.

7. The InflexibleShape class defines a calculate\_area method that isn’t implemented in its subclasses. This adds unnecessary code that might not be used.

8. There is no direct premature optimization, but the extra methods like legacy\_function are not necessary until there's a clear need for optimization.

9. The current structure with repeated logic and inheritance patterns makes the code messy and harder to maintain.

10. The method xyzzy and the legacy\_function are not intuitive, which makes the code harder to read and maintain.