

University of Liberal Arts Bangladesh

Assignment

**Course Code: CSE\_404**

**Course Title: Software Engineering**

**Submitted to:**

**Suravi Akhter**

**Submitted by:**

Student Name: Rezowana Zayed Zahin

Student ID: 201014096

Date of Submission:

21.12.23

Department of CSE

**Software Design Best Practices**

**Introduction to software design:**

Software design refers to the process of creating a plan or blueprint for developing a software application. It involves making design decisions, defining the architecture, components, interfaces and interactions of the software system. Software design is called the foundation for building a system through proper maintainance and efficient software solutions. The motive of designing a software is to find the most effective and efficient way to meet the desired requirements.

Steps that are generally followed in a software design are:

**Requirements Analysis:** In this stage, Gathering and analyzing the requirements for the software are done. Also, the needs of users, stakeholders and the problem domain are tried to understand here.

**System Architecture Design:** In this stage, the high-level structure and components of the software system are identified. Also, tries to define the overall system architecture and how different modules interact with each other.

**User Interface Design:** In this stage, the user interface elements and interactions are kept on designing. Also, focuses on usability, accessibility, and user experience to create wireframes or prototypes to visualize the interface.

**Database Design:** In this stage, when a software requires a database, the data model is designed as well as tables, relationships, and access mechanisms for ensuring the proper organization, integrity, and efficiency of data storage.

**Integration Design:** In this stage, the integration of the software is planned with external systems or services. Also, APIs, protocols, and data formats for smooth communication and interoperability are defined here.

**Security Design:** In this stage , potential security risks and vulnerabilities are addressed. Security measures like authentication, authorization, encryption, and secure data transmission are also designed here.

**Testing and Validation:** In this stage ,the testing strategies and methodologies are maintained to validate the design. Also, the types of testing (unit testing, integration testing, etc.) are determined and defined test cases and expected outcomes.

**Documentation:** In this stage, the design decisions, architectural diagrams, and other relevant information are documented for future reference, maintenance and updates.

**Design Review:** In this stage, a design review with the development team, stakeholders, experts is conducted to get feedback and refine the design.

The software design process is iterative and feedback loops are essential for continuous improvement. Adjustments and refinements may need at each step for new insights and developments.

**Importance of Modular Design in software development:**

Modular design is important in software development for several reasons:

**Code Reusability:** Modular design is necessary for the creation of independent and reusable modules. These modules can easily reuse in different parts of the software system or in future projects. Also, this saves time and effort by avoiding the need to rewrite code and promotes consistency.

**Ease of Maintenance:** Each module focuses on a specific functionality or feature by modular design. This makes it easier to locate and modify specific parts for proper updates.Also, it reduces the risk of introducing unintended changes or dependencies on other components.

**Scalability:** Modular design allows for scalability by enabling the system to be easily modified. New features or functionalities are added by developing new modules as well as modifying existing ones without impacting the entire system. Also, this promotes adaptability to changing requirements or business needs.

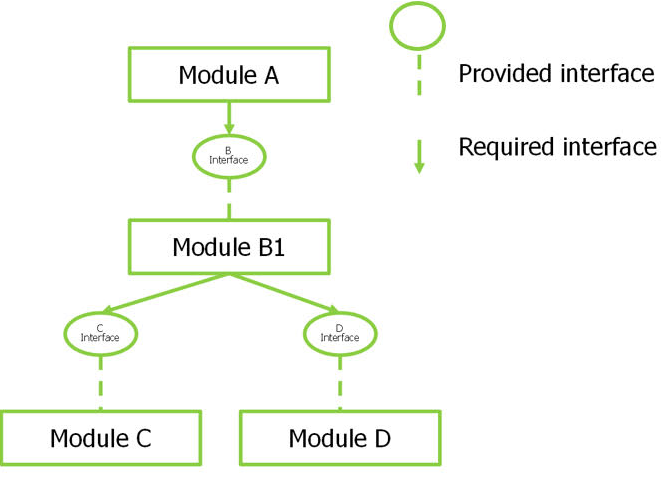
**Collaborative Development:** Modular design encourages collaboration among developers as well as allows for parallel development. Here, each module can be developed, tested, and integrated independently, enhancing productivity and speeding up the development process.

**Testing and Debugging:** Modular design simplifies testing and debugging efforts. Modules can be isolated and tested independently to enable effective unit testing where each module can be tested in isolation to ensure the quality and reliability of the code.

**Flexibility and Interoperability:** Modular design promotes flexibility by allowing the use of different technologies, languages and frameworks for each module. As long as the modules communicate through well-defined interfaces, they can be developed using different tools. Also, this enhances interoperability with other systems and components.

**Concurrent Development:** Modular design allows multiple teams or individuals to work concurrently on different modules without conflicts.Besides,it can maximize resource utilization, accelerates development timelines as well as facilitates the delivery of complex software projects.

**Understanding and Maintainability:** Modular design improves the overall understanding of the system. Each module represents a specific functionality that makes it easier to comprehend the system's structure and behavior. It also promotes maintainability by encapsulating functionality within modules.



So, modular design brings benefits such as code reusability, ease of maintenance, scalability, collaboration, testing efficiency, flexibility, and maintainability to software development.

**Various Design Patterns and their usage in software design:**

Design patterns are established for providing solutions of common software design problems.They give reusable templates which guide developers in solving specific design challenges.Some commonly used design patterns are given below:

**Singleton Pattern:** The Singleton pattern states that a class has only one instance that provides global access to that instance. This belongs to the creational design pattern. It is used when there should be only one instance of a class in the application.

**Factory Pattern:** The Factory pattern states a way to create objects without specifying their exact classes. This belongs to the creational design pattern.It is useful when there is a need to create multiple objects of a similar type and when the creation varies on specific conditions.

**Builder Pattern:** This belongs to the creational design pattern.The Builder pattern is used to comake complex objects step by step. It is also used to separate the creation of an object from its representation. This pattern is commonly employed to create objects with many optional parameters.

**Observer Pattern:** The Observer pattern makes a one-to-many dependency between objects when one object changes its state and all its dependencies are notified and updated automatically. This belongs to the structural design pattern. This pattern is helpful where objects need to be coupled. Changes in one object should be propagated to other dependent objects.

**Adapter Pattern:** The Adapter pattern allows objects with incompatible interfaces to work together by providing a bridge between them. It is useful when an existing class or interface needs to be reused but does not match the required interface of the system.

**Decorator Pattern:** The Decorator pattern dynamically adds new responsibilities to an object by wrapping it in an object of a decorator class. This belongs to the structural design pattern.It also provides a flexible alternative to subclassing for extending functionality. This pattern is commonly used to add features to an existing class without modifying its structure.

**Strategy Pattern:** The Strategy pattern defines a family of interchangeable algorithms that encapsulates each one as a separate class. This belongs to the behavioral design pattern.It also allows the algorithm to vary independently from the clients. It is useful when different variants of an algorithm need to be implemented and selected at runtime.

**Iterator Pattern:** The Iterator pattern allows a way to access the elements of an aggregate object sequentially without exposing its underlying representation. This belongs to the creational design pattern This pattern decouples the algorithm for traversing the elements from the aggregate structure itself that allows for flexible iteration strategies.

**MVC (Model-View-Controller) Pattern:** The MVC pattern divides an application into three interconnected components.Those are- the Model (data and business logic), the View (user interface representation), and the Controller (handles user input and updates the Model and View). This pattern promotes modularity and separation of concerns in the design of user interfaces.

**Facade Pattern:** The Facade pattern converts a simplified interface to a complex subsystem. This belongs to the structural design pattern. It also encapsulates a set of interfaces of a subsystem into a higher-level interface, making it easier to use. It is commonly used to provide a simple, unified API for a more complex underlying system.

It's necessary to choose and apply the appropriate patterns based on the specific requirements and constraints of the software being developed.

**SOLID Principles in software design:**

The SOLID principles refer to a set of guidelines that help in designing software that is easy to maintain, understand, and extend. Some principles are given below:

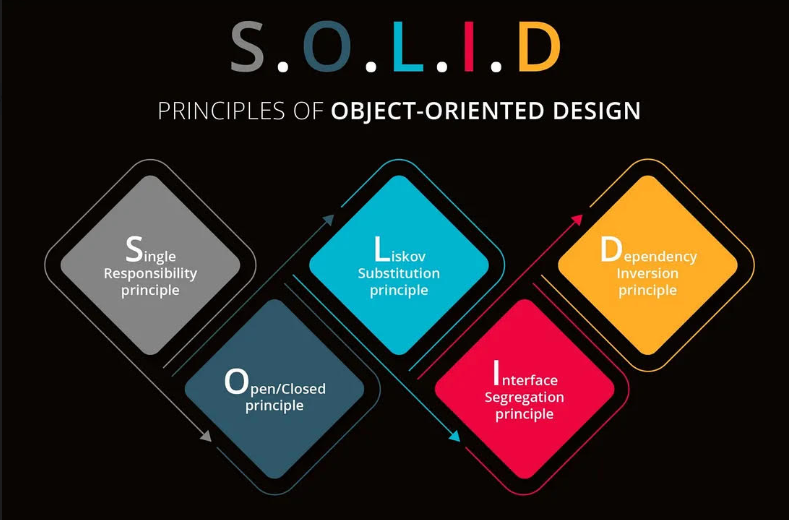
**Single Responsibility Principle (SRP):** A module should have only one reason to change. It states that a class should have only one responsibility. This principle helps in making classes focused that makes easier to understand, test and maintain.

**Open/Closed Principle (OCP):** Software entities should be open for extension but not for modification. It allows the practice of designing software that can be easily extended to incorporate new behaviors without modifying existing code. This principle helps in achieving code reusability and scalability.

**Liskov Substitution Principle (LSP):** Subtypes must be changeable for their base types. It also states that objects of a superclass should be replaceable with objects of its subclasses without changing the correctness of the program. Otherwise it can lead to unexpected behavior and inconsistencies in code.

**Interface Segregation Principle (ISP):** Clients should not be forced to depend on interfaces to which they do not need to use. It also suggests that multiple specific interfaces are better than a single general-purpose interface. This principle helps in avoiding interface pollution and designing.

**Dependency Inversion Principle (DIP):** High-level modules should not depend on low-level modules.It also advocates the use of abstractions (interfaces, abstract classes) to distinct higher-level modules from lower-level implementation details. This principle enables easier testing and mocking, and facilitates code reuse.



Following these SOLID principles code maintainability, modularity, testability, reusability as well as overall design quality are promoted in software development.

**Measures can be taken to increase code quality:**

There are several measures that can be taken to increase code quality. Some effective ones are given below:

• Following coding standards and style guidelines consistently. This helps in consistency, readability, and makes the code easier to understand and maintain.

• Using clear and meaningful names for variables, functions, and classes. Adding comments to explain complex logic or clarify intentions.

• Keeping functions and methods small as well as split larger functions into smaller ones that perform a single task. This helps in readability, reusability, and testability of the code.

• Avoiding code duplication to identify repetitive code segments and refactor them into reusable functions or modules. This reduces redundancy, improves maintainability, and makes the code easier to update.

• Performing code reviews for example peers or experienced developers review codes. Code reviews help in identifying bugs, vulnerabilities, and potential improvements.

• Writing automated unit tests for codes to ensure its correctness and reliability. Test both normal and boundary cases, and aim for high code coverage. Automating tests also helps catch regressions and provides a safety net during refactoring.

• Use version control: Utilize version control systems like Git to track changes and collaborate effectively. Branching, merging, and tagging practices can help manage code changes and releases efficiently.

• Refactoring codes regularly to improve its structure by eliminating code smells and enhance its maintainability. Refactoring should be done in small and incremental steps to minimize the risk of introducing bugs.

• Using design patterns can be applied to solve common problems in software development. Design patterns provide proper solutions by promoting code reusability, scalability, and maintainability.

• Employing automation tools and processes for continuous integration and deployment ensure code changes effectively by reducing the likelihood of integration issues.

By following these measures, the quality, readability, maintainability, and reliability of your code can be significantly improved.

**How testing can ensure correct software design:**

Testing plays a vital role in ensuring correct software design by validating the functionality, behavior and performance of a software system.

• By designing and executing test cases the software components, modules can be integrated properly as well as performance of a system with their intended functions can be ensured correctly.

• By identifying and fixing issues during the testing phase,the overall correctness and stability of the design can also be improved.

• Testing allows to validate the behavior of the software under different cases and boundary conditions.

• By comparing the actual behavior with the expected design, testing verifies if the implemented system meets the desired design standards and fulfills the functional and non-functional

• Testing enables to optimize the design that ensures the expected workload under user demands effectively.

• By retesting previously tested areas it is validated that they continue to function correctly or not. Also, maintains the integrity and correctness of the design throughout its lifecycle.

**Concluson:**

In conclusion, software design is a critical phase in the software development lifecycle where the requirements are converted to a well-structured and organized system design. It involves making important architectural decisions, defining module interfaces, selecting appropriate data structures and algorithms, and ensuring proper integration and interaction between different components.

Thus, software design is the foundation of a reliable and successful software system. It enables smooth implementation, effective testing, and efficient maintenance. By following design principles and incorporating feedback from users and stakeholders, software designers can create solutions that meet user requirements effectively.

**“THE END”**