SOFTWARE ENGINEERING & PROJECT MANAGEMENT (SEPM)

MODULE 3 – Design Engineering

Based on Mumbai University PYQs (2023–2025)

This document contains detailed descriptive answers for ALL Module 3 questions from the uploaded SEPM question bank.

Covered Questions:

1. Explain User Interface Design in detail with example.
2. Explain Golden Rules for User Interface Design.
3. Explain Software Design Concepts.
4. Explain Design Concepts and Architectural Design Styles.
5. Explain Cohesion and Coupling.
6. Draw CFG and Calculate Cyclomatic Complexity.
7. Short Note on Software Design Patterns.
8. Architectural Design Styles.
9. Functional Independence.
10. UI Design Principles.

Q1. Explain User Interface Design in Detail with Example.

Introduction

The User Interface (UI) is one of the most important components of software systems because it acts as a communication bridge between the user and the software.

A good user interface improves:

- User satisfaction
- Productivity
- Efficiency
- Ease of use
- Software quality

Poor interface design may confuse users and reduce system efficiency.

Definition of User Interface Design

User Interface Design is the process of designing screens, controls, menus, forms, buttons, and interactions between the user and software.

It focuses on:

- User friendliness
- Simplicity
- Consistency
- Accessibility
- User experience

Objectives of User Interface Design

1. Improve usability.
2. Reduce user effort.
3. Increase efficiency.
4. Provide better user experience.
5. Minimize user errors.
6. Make software visually attractive.

Characteristics of Good User Interface

1. Simple and easy to use
2. Consistent design
3. Fast interaction
4. Error prevention
5. Clear navigation
6. Attractive appearance
7. User-friendly layout

Importance of User Interface Design

1. Improves User Satisfaction

A good UI makes software easier to use.

2. Reduces Training Time

Users can quickly learn the system.

3. Reduces Errors

Clear design prevents mistakes.

4. Improves Productivity

Efficient UI increases work speed.

5. Increases Software Acceptance

Users prefer software with better UI.

UI Design Process

The UI design process consists of several stages:

```
User Analysis → Interface Design → Prototype → Evaluation → Implementation
```

---

1. User Analysis

Understanding:

- User behavior
- User requirements
- Technical knowledge of users

---

2. Interface Design

Designing:

- Menus
- Forms
- Navigation
- Buttons
- Layouts

---

3. Prototype Development

A sample interface is developed for testing.

---

4. Evaluation

The interface is tested using user feedback.

---

5. Implementation

Final interface is integrated into the software.

UI Design Principles

1. Consistency

Same layout and controls should be used throughout the system.

2. Simplicity

Interface should be easy to understand.

3. Feedback

System should provide immediate response.

4. Error Handling

Errors should be detected and corrected easily.

5. User Control

Users should feel in control of the system.

Example of User Interface Design

Example: ATM Machine Interface

Features:

- Simple menu options
- Touchscreen interface
- Secure PIN entry
- Clear instructions
- Error messages

ATM Interface Diagram

```
---------------------------------
|           ATM SYSTEM          |
---------------------------------
| 1. Withdraw Cash              |
| 2. Deposit Money              |
| 3. Balance Inquiry            |
| 4. Mini Statement             |
| 5. Exit                       |
---------------------------------
```

Advantages of Good UI Design

1. Better user experience.
2. Increased productivity.
3. Reduced user errors.
4. Improved software usability.

5. Faster task completion.

Disadvantages of Poor UI Design

1. User confusion.
2. Increased training cost.
3. Reduced efficiency.
4. Higher chances of errors.
5. User dissatisfaction.

Conclusion

User Interface Design is a critical part of software engineering. A well-designed interface improves software usability, customer satisfaction, and overall system efficiency.

Q2. Explain Golden Rules for User Interface Design.

Introduction

Golden Rules for User Interface Design were proposed by Theo Mandel.

These rules help developers design user-friendly and efficient interfaces.

The three major golden rules are:

1. Place users in control.
2. Reduce user memory load.
3. Make the interface consistent.

---

1. Place Users in Control

Users should feel that they control the software and not vice versa.

Guidelines

a) Flexible Interaction

Allow users to customize operations.

b) Interruptible Operations

Users should be able to stop unwanted operations.

c) Easy Navigation

Provide simple navigation mechanisms.

d) Direct Interaction

Users should interact directly with objects.

e) Hide Technical Details

Avoid displaying unnecessary technical information.

Advantages

- Improves user confidence
- Better user experience
- Reduces frustration

---

2. Reduce User Memory Load

The interface should minimize the amount of information users need to remember.

Guidelines

a) Simple Interface

Use simple layouts.

b) Meaningful Icons

Use understandable icons and symbols.

c) Default Values

Provide default options.

d) Shortcuts

Provide keyboard shortcuts.

e) Progressive Disclosure

Display only necessary information.

Advantages

- Faster interaction
- Reduced confusion
- Improved efficiency

---

3. Make Interface Consistent

Consistency improves software learnability.

Guidelines

a) Consistent Commands

Same commands should behave similarly.

b) Consistent Layout

Same screen layouts should be used.

c) Consistent Colors and Fonts

Use similar design elements.

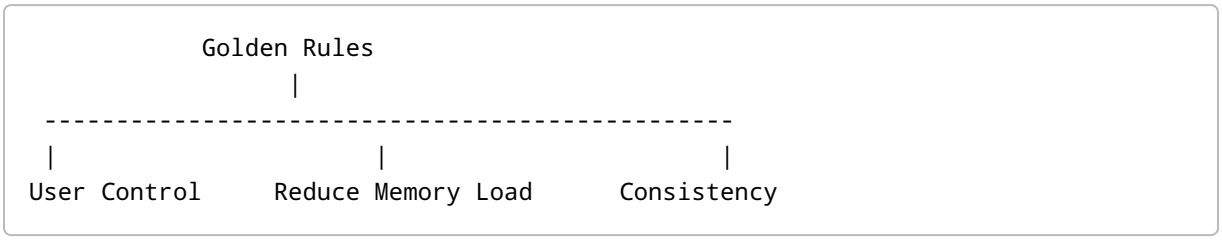
d) Standard Icons

Use commonly understood icons.

Advantages

- Easy learning
- Better navigation
- Improved usability

Diagram of Golden Rules



Applications of Golden Rules

- Web applications
- Mobile applications
- Banking software
- ATM systems
- E-commerce systems

Conclusion

Golden Rules help developers create effective, user-friendly, and efficient software interfaces.

Q3. Explain Software Design Concepts.

Introduction

Software Design is the process of converting requirements into software architecture and implementation details.

Design concepts provide guidelines for creating efficient and maintainable software.

Important Software Design Concepts

1. Abstraction
2. Refinement
3. Modularity
4. Software Architecture
5. Information Hiding
6. Functional Independence
7. Cohesion
8. Coupling
9. Patterns
10. Refactoring

---

1. Abstraction

Abstraction focuses on important details while hiding unnecessary implementation details.

Types of Abstraction

a) Procedural Abstraction

Focuses on sequence of operations.

b) Data Abstraction

Focuses on data objects.

Advantages

- Reduces complexity
- Improves understanding
- Simplifies design

---

2. Refinement

Refinement is the process of breaking large problems into smaller parts.

Also called:

Stepwise Refinement

Example

```
Login System
   ↓
Input Username
   ↓
Input Password
   ↓
Validate User
```

Advantages

- Easier implementation
- Better organization
- Reduced complexity

---

3. Modularity

Software is divided into smaller modules.

Each module performs a specific function.

Advantages

1. Easy maintenance
2. Better testing
3. Easy debugging
4. Reusability

---

4. Software Architecture

Software Architecture defines the overall structure of the software system.

It specifies:

- Components
- Relationships
- Communication mechanisms

Advantages

- Better system organization
- Improved scalability
- Easier maintenance

---

5. Information Hiding

Implementation details are hidden from other modules.

Only necessary information is exposed.

Advantages

- Improved security
- Reduced complexity
- Easier modifications

---

6. Functional Independence

A module should perform one specific task independently.

Functional independence is achieved using:

- High Cohesion
- Low Coupling

Advantages

- Easier testing
- Better maintainability
- Reduced complexity

---

7. Cohesion

Cohesion measures the relationship among elements inside a module.

High cohesion is desirable.

---

8. Coupling

Coupling measures dependency between modules.

Low coupling is desirable.

---

9. Design Patterns

Reusable solutions to common software problems.

Examples:

- Singleton Pattern
- Factory Pattern
- Observer Pattern

---

10. Refactoring

Refactoring improves software structure without changing functionality.

Advantages of Design Concepts

1. Improved software quality.
2. Better maintainability.
3. Easier debugging.
4. Improved scalability.
5. Reduced development complexity.

Conclusion

Software design concepts provide a strong foundation for developing maintainable and efficient software systems.

Q4. Explain Design Concepts and Architectural Design Styles.

Introduction

Architectural Design defines the high-level structure of software systems.

Architectural styles provide reusable system organization patterns.

Architectural Design Styles

1. Data-Centered Architecture
2. Data Flow Architecture
3. Call and Return Architecture
4. Object-Oriented Architecture
5. Layered Architecture

---

1. Data-Centered Architecture

A central repository stores data.

All components access this repository.

Example

Database Systems

Diagram

Advantages

- Centralized data management
- Easy integration

---

2. Data Flow Architecture

Data flows through system components.

Example

Compiler Design

Diagram

```
Input → Processing → Output
```

Advantages

- Easy understanding
- Modular structure

---

3. Call and Return Architecture

Control passes from one module to another.

Example

Traditional procedural programs.

Diagram

```
Main Module
   |
Sub Modules
```

Advantages

- Easy implementation
- Structured design

---

4. Object-Oriented Architecture

System is organized using objects.

Features

- Encapsulation
- Inheritance
- Polymorphism

Advantages

- Reusability
- Scalability
- Easy maintenance

---

5. Layered Architecture

Software is divided into layers.

Example

Operating Systems

Diagram

```
Presentation Layer
Business Layer
Data Layer
```

Advantages

- Better organization
- Easy maintenance
- Improved security

Conclusion

Architectural styles help developers design scalable and maintainable software systems.

Q5. Explain Cohesion and Coupling.

Introduction

Cohesion and Coupling are important concepts used to measure software quality.

Good software design aims for:

High Cohesion and Low Coupling

Cohesion

Cohesion measures the relationship among elements within a module.

Higher cohesion means better module quality.

Types of Cohesion

1. Coincidental Cohesion

Unrelated tasks are grouped together.

2. Logical Cohesion

Tasks are logically related.

3. Temporal Cohesion

Tasks executed at same time.

4. Procedural Cohesion

Tasks follow sequence.

5. Communicational Cohesion

Tasks operate on same data.

6. Sequential Cohesion

Output of one task becomes input of another.

7. Functional Cohesion

Module performs one specific task.

Best type of cohesion.

Coupling

Coupling measures dependency between modules.

Lower coupling is preferred.

Types of Coupling

1. Content Coupling

One module modifies another module.

Worst type.

2. Common Coupling

Modules share global data.

3. External Coupling

Modules depend on external systems.

4. Control Coupling

One module controls another.

5. Stamp Coupling

Modules share data structures.

6. Data Coupling

Modules communicate through parameters.

Best type.

Difference Between Cohesion and Coupling

Cohesion	Coupling
Relationship within module	Relationship between modules
High cohesion desirable	Low coupling desirable
Improves module quality	Reduces dependency
Internal property	External property

Advantages of High Cohesion and Low Coupling

1. Better maintainability.
2. Easier debugging.
3. Improved reusability.
4. Better software quality.
5. Easier testing.

Conclusion

Cohesion and Coupling are essential concepts for achieving functional independence and high-quality software design.

Q6. Draw CFG and Calculate Cyclomatic Complexity.

Introduction

Control Flow Graph (CFG) represents program control structure.

Cyclomatic Complexity measures logical complexity of a program.

It was proposed by:

Thomas McCabe

Control Flow Graph (CFG)

A CFG consists of:

- Nodes
- Edges
- Decision points

Importance of CFG

1. Program analysis
2. Complexity calculation
3. Test case design
4. Path testing

Cyclomatic Complexity

Cyclomatic Complexity measures number of independent execution paths.

Formula

```
V(G) = E - N + 2P
```

Where:

- E = Number of edges
- N = Number of nodes
- P = Number of connected components

Alternative Formula

```
V(G) = Number of Decision Nodes + 1
```

Example

```
if(condition)
{
   statement1;
}
else
{
   statement2;
}
```

Decision nodes = 1

Therefore:

```
V(G) = 1 + 1
     = 2
```

Advantages of Cyclomatic Complexity

1. Measures software complexity.

2. Helps test case design.
3. Improves maintainability.
4. Identifies risky modules.

Conclusion

CFG and Cyclomatic Complexity help developers analyze program complexity and improve software testing.

Q7. Short Note on Software Design Patterns.

Introduction

Software Design Patterns are reusable solutions to common software design problems.

Design patterns improve software maintainability and scalability.

Categories of Design Patterns

1. Creational Patterns
2. Structural Patterns
3. Behavioral Patterns

---

1. Creational Patterns

Used for object creation.

Examples:

- Singleton
- Factory
- Builder

---

2. Structural Patterns

Used for object composition.

Examples:

- Adapter
- Bridge
- Decorator

---

3. Behavioral Patterns

Used for communication between objects.

Examples:

- Observer
- Strategy
- Command

Advantages of Design Patterns

1. Reusable solutions.
2. Reduced development time.
3. Better maintainability.
4. Improved software quality.
5. Standardized design.

Conclusion

Design patterns help developers create scalable, reusable, and maintainable software systems.

END OF MODULE 3