Assignment: Software Engineering Fundamentals - Midterm Exam
Model Answer Sheet
Course: CS-301 Software Engineering
Instructor: Prof. Shubh

==

Question 1: Software Development Life Cycle (20 marks)

a) Definition of SDLC (5 marks)

Model Answer:
The Software Development Life Cycle (SDLC) is a structured process that defines the phases and activities involved in developing software systems from conception to deployment and maintenance. It provides a systematic approach to software development that ensures quality, efficiency, and predictability.

Importance of SDLC:
- Provides structure and organization to development process
- Ensures quality through defined checkpoints and reviews
- Reduces development risks and costs
- Improves communication among team members
- Enables better project management and timeline estimation
- Facilitates maintenance and future enhancements

b) Waterfall vs Agile Comparison (10 marks)

Model Answer:

Waterfall Model:
Advantages:
- Clear, linear progression through phases
- Well-documented requirements and design
- Easy to manage and track progress
- Suitable for projects with stable requirements
- Clear deliverables and milestones

Disadvantages:
- Inflexible to changing requirements
- Testing occurs late in the process
- No working software until end of project
- High risk if requirements are misunderstood

- Customer feedback comes very late

Agile Methodology:
Advantages:
- Flexible and adaptable to change
- Early and continuous delivery of working software
- Regular customer feedback and collaboration
- Reduced risk through iterative development
- Better team collaboration and communication

Disadvantages:
- Requires active customer involvement
- Can lead to scope creep
- Less predictable timelines and costs
- Requires experienced team members
- Documentation may be inadequate

c) Mobile Banking Application Choice (5 marks)

Model Answer:
For a mobile banking application, I would choose the Agile methodology for the following reasons:

1. Security Requirements Evolution: Banking security requirements evolve rapidly, and Agile allows for quick adaptation to new security standards and threats.

2. Regulatory Compliance: Financial regulations change frequently, and Agile enables quick incorporation of new compliance requirements.

3. User Experience Focus: Banking apps require excellent UX, and Agile's iterative approach allows for continuous user feedback and interface refinement.

4. Market Competition: The banking sector is highly competitive, requiring rapid feature delivery to stay competitive.

5. Technology Integration: Banking apps integrate with multiple systems, and Agile allows for better handling of integration challenges through iterations.

==

Question 2: Object-Oriented Programming Principles (25 marks)

a) Four OOP Principles (12 marks)

Model Answer:

1. Encapsulation:
Definition: Bundling data and methods that operate on that data within a single unit (class), and controlling access to internal details.
Real-world analogy: A car's engine - you can start the car using the key (public interface), but you cannot directly manipulate the internal combustion process (private implementation).

2. Inheritance:
Definition: Mechanism where a new class acquires properties and behaviors from an existing class.
Real-world analogy: A family tree - children inherit characteristics from their parents, but may also have unique features.

3. Polymorphism:
Definition: Ability of objects of different classes to be treated as objects of a common base class, while maintaining their specific behaviors.
Real-world analogy: Different animals (dog, cat, bird) can all make sounds, but each makes a different sound when asked to "speak."

4. Abstraction:
Definition: Hiding complex implementation details and showing only essential features to the user.
Real-world analogy: Using a television remote - you press buttons to change channels without needing to understand the internal circuitry.

b) Vehicle Hierarchy Implementation (13 marks)

Model Answer:

UML Class Diagram:
```
Vehicle
-------
- brand: String
- year: int
-------
+ startEngine(): void
+ stopEngine(): void
+ getInfo(): String
```

```
^
|
---------+---------
|                 |
Car              Motorcycle
---             ----------
- doors: int    - hasSidecar: boolean
---             ----------
+ openDoors()   + wheelie(): void
+ getInfo()     + getInfo(): String
```

Java Implementation:

```java
// Parent class
public class Vehicle {
protected String brand;
protected int year;

public Vehicle(String brand, int year) {
this.brand = brand;
this.year = year;
}

public void startEngine() {
System.out.println("Vehicle engine started");
}

public void stopEngine() {
System.out.println("Vehicle engine stopped");
}

public String getInfo() {
return brand + " " + year;
}
}

// Child class 1
public class Car extends Vehicle {
private int doors;
public Car(String brand, int year, int doors) {
super(brand, year);
this.doors = doors;
}

public void openDoors() {
System.out.println("Opening " + doors + " doors");
}

@Override
public String getInfo() {
return super.getInfo() + " - " + doors + " door car";
}
}

// Child class 2
public class Motorcycle extends Vehicle {
private boolean hasSidecar;

public Motorcycle(String brand, int year, boolean hasSidecar) {
super(brand, year);
this.hasSidecar = hasSidecar;
}

public void wheelie() {
System.out.println("Performing wheelie!");
}

@Override
public String getInfo() {
return super.getInfo() + " - Motorcycle" +
(hasSidecar ? " with sidecar" : "");
}
}
```

==

Question 3: Software Design Patterns (20 marks)

a) Design Patterns Definition (5 marks)

Model Answer:
Design patterns are reusable solutions to commonly occurring problems in software design. They represent best practices and proven solutions that have evolved over time through the experience of skilled developers.

Importance:
- Provide tested, proven development paradigms
- Improve code readability and maintainability
- Enable better communication among developers
- Speed up development process
- Reduce errors by using proven solutions
- Make code more flexible and reusable

b) Singleton Pattern (10 marks)

Model Answer:

Definition: Singleton pattern ensures a class has only one instance and provides global access to that instance.

Practical Example: Database connection manager - you want only one connection pool for the entire application.

UML Diagram:
```
DatabaseManager
---------------
- instance: DatabaseManager
- connection: Connection
---------------
- DatabaseManager()
+ getInstance(): DatabaseManager
+ getConnection(): Connection
```

Java Implementation:
```java
public class DatabaseManager {
private static DatabaseManager instance = null;
private Connection connection;

private DatabaseManager() {
```

```
// Private constructor prevents instantiation
connection = createConnection();
}

public static synchronized DatabaseManager getInstance() {
if (instance == null) {
instance = new DatabaseManager();
}
return instance;
}

public Connection getConnection() {
return connection;
}

private Connection createConnection() {
// Create database connection
return DriverManager.getConnection("jdbc:mysql://localhost/db");
}
}
```

c) Observer vs Factory Pattern (5 marks)

Model Answer:

Observer Pattern:
- Purpose: Defines one-to-many dependency between objects
- Use when: You need to notify multiple objects about state changes
- Example: Newsletter subscription system, GUI event handling

Factory Pattern:
- Purpose: Creates objects without specifying their exact class
- Use when: You need to create objects based on certain conditions
- Example: Creating different types of documents (PDF, Word, Excel) based on file type

When to use Observer: When you have a subject that changes state and multiple observers that need to be notified of these changes.

When to use Factory: When you need to create objects but the exact type depends on runtime conditions or configuration.

==

Question 4: Software Testing and Quality Assurance (20 marks)

a) Types of Testing (8 marks)

Model Answer:

Unit Testing:
- Tests individual components or modules in isolation
- Focuses on smallest testable parts of an application
- Example: Testing a single function that calculates tax amount

Integration Testing:
- Tests interaction between integrated modules
- Verifies that combined components work correctly together
- Example: Testing data flow between login module and user profile module

System Testing:
- Tests complete integrated system
- Verifies system meets specified requirements
- Example: Testing entire e-commerce website end-to-end, from product browsing to payment processing

b) Test-Driven Development (7 marks)

Model Answer:

Test-Driven Development (TDD):
TDD is a software development approach where tests are written before the actual code. The process follows Red-Green-Refactor cycle:
1. Red: Write a failing test
2. Green: Write minimal code to make test pass
3. Refactor: Improve code while keeping tests passing

Advantages:
- Better code quality and design
- Comprehensive test coverage
- Easier debugging and maintenance
- Documentation through tests
- Reduced regression bugs
- Encourages simple design

Challenges:
- Initial learning curve
- Time-intensive upfront
- Requires discipline and practice
- May slow down initial development
- Difficult for complex UI testing

c) Login Test Cases (5 marks)

Model Answer:

Positive Test Cases:
1. Valid username and valid password !’ Login successful
2. Valid email and valid password !’ Login successful
3. Remember me checkbox functionality !’ Session maintained

Negative Test Cases:
1. Invalid username and valid password !’ Login failed
2. Valid username and invalid password !’ Login failed
3. Empty username field !’ Error message displayed
4. Empty password field !’ Error message displayed
5. SQL injection attempt !’ System remains secure
6. Brute force attack !’ Account locked after attempts
7. Case sensitivity test !’ Appropriate behavior
8. Special characters in password !’ Handled correctly

==

Question 5: Software Requirements Engineering (15 marks)

a) Functional vs Non-functional Requirements (8 marks)

Model Answer:

Functional Requirements (what the system should do):
For e-commerce website:
1. User registration and login capability
2. Product search and filtering functionality
3. Shopping cart and checkout process
4. Order tracking and history
5. Payment processing integration

Non-functional Requirements (how the system should perform):
For e-commerce website:
1. Performance: Page load time under 3 seconds
2. Security: SSL encryption for all transactions
3. Availability: 99.9% uptime requirement
4. Scalability: Support 10,000 concurrent users
5. Usability: Intuitive interface for all user types

b) Requirements Validation and Verification (4 marks)

Model Answer:

Requirements Validation:
- Ensures requirements meet customer needs and expectations
- Answers: "Are we building the right product?"
- Involves stakeholder reviews and approval

Requirements Verification:
- Ensures requirements are complete, consistent, and testable
- Answers: "Are we building the product right?"
- Involves technical review and analysis

Importance:
- Prevents costly changes later in development
- Ensures customer satisfaction
- Reduces project risks
- Improves project success rate

c) Requirements Gathering Techniques (3 marks)

Model Answer:

1. Interviews:
- One-on-one discussions with stakeholders
- Allows deep exploration of requirements
- Best for understanding complex business processes

2. Surveys/Questionnaires:
- Collect information from large groups
- Standardized questions for consistency
- Efficient for gathering broad feedback

3. Workshops/Focus Groups:
- Collaborative sessions with multiple stakeholders
- Encourages discussion and consensus building
- Effective for resolving conflicting requirements

==

Grading Criteria:
- Accuracy of technical concepts
- Depth of explanation
- Use of appropriate examples
- Clarity of communication
- Demonstration of understanding vs memorization