SDLC Assignment Solutions

1. Introduction to SDLC:

- **Q1:** What is the Software Development Life Cycle (SDLC)? Explain why SDLC is important in software development.

  Answer:

  ### Software Development Life Cycle (SDLC)

  SDLC is a process followed for software building within a software organization. SDLC consists of a precise plan that describes how to develop, maintain, replace, and enhance specific software. The life cycle defines a method for improving the quality of software and the all-around development process.

  ### Importance of SDLC in software development

  1. **Organized Development**: Breaks the project into clear phases (like planning, design, coding, testing), making the process structured and manageable.
  2. **Better Planning**: Helps estimate time, cost, and resources accurately before development begins.
  3. **Clear Requirements**: Ensures developers fully understand what the client wants, reducing confusion later.
  4. **Early Error Detection**: Testing is done in phases, so bugs are caught early—saving time and effort.
  5. **Improved Communication**: Provides a common process for developers, testers, and clients to stay on the same page.
  6. **High-Quality Output**: Each step includes checks, which improves the final quality of the software.
  7. **Easier Maintenance**: Makes future updates and bug fixes more manageable after the software is delivered.
  8. **Risk Reduction**: Identifies risks early and allows time to fix them before they become bigger issues.

- **Q2:** List and describe the different phases of the SDLC. How does each phase contribute to the overall software development process?

  Answer:

Phases of SDLC :

1. Requirement Gathering and Analysis

- **Goal:** Understand what the client or user needs from the software.
- **Activities:**
  - Meet with stakeholders
  - Document requirements
  - Analyze feasibility (technical, financial, legal)
- **Output:** Software Requirement Specification (SRS) document
- Helps set a clear goal and avoid confusion later.

2. System Design

- **Goal:** Plan how the software will be built.
- **Activities:**
  - Design system architecture
  - Choose technologies (e.g., database, frameworks)
  - Create data flow diagrams, UI mockups
- **Output:** Design Document (includes High-level and Low-level design)
- Acts as a blueprint for developers to follow.

3. Implementation (Coding)

- **Goal:** Convert the design into actual working code.
- **Activities:**
  - Developers write code based on design
  - Follow coding standards and use version control
- **Output:** Source code of the software
- Brings the planned system to life.

4. Testing

- **Goal:** Ensure the software works as expected and is free of bugs.
- **Activities:**
  - Unit testing, Integration testing, System testing
  - Find and fix defects

- **Output:** Tested and verified software
- Ensures reliability, security, and quality.

5. Deployment

- **Goal:** Release the software to users or clients.
- **Activities:**
    - Install the software on production servers
    - Provide training or documentation if needed
- **Output:** Live/working software for users
- Makes the software available for actual use.

6. Maintenance

- **Goal:** Keep the software running smoothly after release.
- **Activities:**
    - Fix bugs
    - Add new features
    - Update software for new requirements or environments
- **Output:** Updated, improved versions of the software
- Ensures long-term success and user satisfaction.

Their Contributions:

Phase	Key Role in Development
Requirement Analysis	Defines what to build
Design	Plans how to build it
Implementation	Builds the software
Testing	Ensures it works correctly
Deployment	Delivers it to users
Maintenance	Improves and supports it post-launch

- **Q3:** Explain the difference between **Waterfall Model**, **Agile Model**, and **V-Model**. In which situations would each model be most appropriate?

  Answer:

Waterfall Model

Definition:
A **linear and sequential** model where each phase must be completed before moving to the next.

Key Features:

- Rigid structure
- No going back to previous phases
- Easy to manage and document

Best Suited For:

- Small or simple projects
- Projects with **clear and fixed requirements**
- When **client is not involved much** during development

Agile Model

Definition:
An **iterative and flexible** model that delivers software in small, usable parts (sprints).

Key Features:

- Frequent changes are welcome
- Regular client feedback
- Teams work in cycles (sprints)

Best Suited For:

- **Complex or large projects**
- Projects where **requirements change frequently**
- **Active client involvement** is possible

V-Model (Validation & Verification Model)

Definition:
An extension of the Waterfall model where **each development phase has a corresponding testing phase**.

Key Features:

- Testing is planned alongside development
- Emphasizes **quality and validation**
- More structured than Waterfall

Best Suited For:

- Critical systems (e.g., healthcare, aerospace)
- Projects needing strict testing and validation
- Where zero defects are important

---

2. SDLC Phases and their Importance:

- **Q4:** Describe the **Requirement Gathering** phase of the SDLC. What methods are used to gather requirements from stakeholders?

  Answer:

  ### Requirement Gathering Phase of SDLC –

  It is the **first phase** of the Software Development Life Cycle (SDLC) where the development team works with stakeholders to **understand what the software must do**.

  ### Main Goal:

  To collect **clear, complete, and detailed requirements** from clients and users to ensure the final product meets their needs.

  ### Common Methods Used to Gather Requirements:

  1. **Interviews**
     One-on-one discussions with stakeholders to understand their needs and expectations.

2. **Questionnaires/Surveys**
   Written sets of questions to gather input from a large number of users quickly.
3. **Workshops**
   Group sessions with clients, developers, and users to collaboratively define requirements.
4. **Observation**
   Watching how users perform tasks to understand their problems and needs.
5. **Document Analysis**
   Reviewing existing documents like manuals, reports, or current systems for reference.
6. **Prototyping**
   Creating simple mockups or models of the system to clarify what the user wants.
7. **Brainstorming**
   Generating ideas with stakeholders to discover hidden or innovative requirements.

Output of This Phase:

- **Software Requirements Specification (SRS)** document
  – A detailed and formal document listing **functional** and **non-functional** requirements.

Why This Phase is Important:

- Prevents misunderstandings later
- Reduces costly changes during development
- Ensures client satisfaction

- **Q5:** In the **Design** phase, what are the key activities involved? Differentiate between high-level design and low-level design.

  Answer:

Design Phase of SDLC –

The **Design Phase** is the **second phase** of the Software Development Life Cycle (SDLC), where the **system's structure and plan** are created based on the requirements gathered in the first phase.

Key Activities in the Design Phase:

1. **System Architecture Design**
   Define the overall structure of the system (modules, components, interactions).
2. **Database Design**
   Decide how data will be stored, accessed, and managed (ER diagrams, schema design).
3. **Interface Design**
   Plan how users will interact with the system (UI/UX mockups, screens).
4. **Technology Selection**
   Choose programming languages, frameworks, tools, and platforms.
5. **Security Design**
   Plan security measures like authentication, data protection, etc.
6. **Design Document Creation**
   Document both **High-Level Design (HLD)** and **Low-Level Design (LLD)**.

High-Level Design (HLD) vs Low-Level Design (LLD)

Feature	High-Level Design (HLD)	Low-Level Design (LLD)
Focus	Overall system architecture	Internal logic of each module/component
Details	Broad structure and major components	Detailed algorithms, functions, and logic
Audience	Project managers, system architects	Developers
Examples	Module diagrams, data flow diagrams	Class diagrams, pseudocode, database tables
Purpose	Shows **what** each module does	Shows **how** each module works internally

- **Q6:** Explain the **Coding** or **Development** phase of the SDLC. What tools and techniques are typically used by developers during this phase?

  Answer:

  ## What is the Coding Phase?

  The **Coding phase** (also called **Implementation phase**) is the stage where **actual programming** begins. Developers use the design documents (HLD & LLD) to write the software's source code.

  ## Main Goal:

  To **convert design into a working software** by writing clean, efficient, and bug-free code.

  ## Key Activities in This Phase:

  1. **Writing Code:**
     Developers write the actual program using suitable programming languages.
  2. **Version Control:**
     Code is managed using tools like **Git** to track changes and collaborate.
  3. **Unit Testing:**
     Each module is tested by the developer to ensure it works properly.
  4. **Code Reviews:**
     Peer reviews are done to improve code quality and catch bugs early.
  5. **Integration:**
     Combine different modules to form a complete system.

  ## Common Tools & Techniques Used:

  ### *Programming Languages*

  - Java, Python, C++, JavaScript, etc.

  ### *IDEs (Integrated Development Environments)*

  - **VS Code**, IntelliJ IDEA, Eclipse, PyCharm

  ### *Version Control Systems*

  - **Git**, GitHub, GitLab, Bitbucket

Code Review Tools

- GitHub Pull Requests, Crucible, Gerrit

Testing Tools

- JUnit, PyTest, Selenium (for automation testing)

Build Tools

- Maven, Gradle, Webpack, Make

Security & Static Code Analysis

- SonarQube, Checkmarx, ESLint

Importance of the Coding Phase:

- It's the **core** of the project — turning plans into a **working product**.
- Clean and well-structured code makes future maintenance **easier and faster**.
- Quality in this phase affects **overall performance, reliability, and scalability** of the software.

- **Q7:** What is the importance of the **Testing** phase in SDLC? Explain the different types of testing that are performed during this phase (e.g., unit testing, integration testing, system testing).

  Answer:

Why is Testing Important?

1. **Detects and fixes bugs early** to avoid costly fixes later.
2. **Ensures software meets user requirements** and functions as expected.
3. **Improves quality** and **builds user trust**.
4. **Verifies performance, security, and compatibility**.
5. **Reduces risk of failure** after deployment.

Types of Testing Performed in This Phase

Type	Description
Unit Testing	Tests **individual functions or modules**. Done by developers.
Integration Testing	Tests how **multiple modules work together**. Checks data flow and interaction.
System Testing	Tests the **entire system as a whole**. Verifies overall functionality and performance.
User Acceptance Testing (UAT)	Done by the **end users** to confirm the system meets their needs.
Regression Testing	Ensures **new changes don't break** existing features.
Performance Testing	Tests **speed, scalability, and responsiveness** under load.
Security Testing	Checks for **vulnerabilities and data protection**.
Compatibility Testing	Ensures software works on **different devices, browsers, or OS**.

- **Q8:** Describe the **Deployment** phase in the SDLC. What are the key considerations for successfully deploying software into a live environment?

  Answer:

What is the Deployment Phase?

The **Deployment Phase** is the stage in the SDLC where the **tested software is released into the live/production environment** so that end users can use it.

Main Goal:

To **deliver the final working product** to the users with minimal issues and smooth transition.

Key Activities in Deployment Phase:

1. **Install the software** on client systems or servers.

2. **Migrate data** from old systems (if applicable).
3. **Configure environments** (production settings, user access, etc.).
4. **Provide documentation and training** to users.
5. **Monitor the system** to ensure it's running smoothly.

Key Considerations for Successful Deployment:

Consideration	**Why It's Important**
Proper Testing	Ensure software is bug-free before release
Security Setup	Secure access, data encryption, and authentication
Backup & Recovery Plans	Have backup in case deployment fails or data is lost
Environment Setup	Make sure production environment is properly configured
User Training	Train end-users to use the system effectively
Rollout Strategy	Use methods like **phased**, **big bang**, or **blue-green deployment**
Post-deployment Testing	Check the system once live to ensure everything works
Support Plan	Prepare for quick fixes if users face issues after deployment

Deployment Types:

- **Phased Deployment:** Release to a few users first, then expand.
- **Big Bang Deployment:** Full release to all users at once.
- **Blue-Green Deployment:** Switch traffic from old version to new without downtime.

- **Q9:** What happens during the **Maintenance** phase? Why is it important for the long-term success of the software?

  Answer:

What is the Maintenance Phase?

The **Maintenance Phase** is the **final phase** of the SDLC, where the software is **monitored, updated, and improved** after it has been deployed.

What Happens During Maintenance?

1. **Bug Fixing**
   - Resolving issues or errors reported by users after deployment.
2. **Upgrades & Enhancements**
   - Adding new features or improving existing ones based on user feedback.
3. **Performance Optimization**
   - Improving speed, efficiency, and stability of the software.
4. **Security Updates**
   - Fixing vulnerabilities to protect against cyber threats.
5. **Adaptation to Environment Changes**
   - Updating software to work with new hardware, operating systems, or other systems.

Why Maintenance is Important:

Reason	Explanation
Keeps Software Relevant	Adapts to user needs and technology changes
Ensures Reliability	Fixes bugs and reduces system failures
Maintains Security	Applies patches to prevent data breaches
Supports Business Growth	Enables scaling, adding features, and improving user experience
Protects Investment	Keeps the software useful for a long time, avoiding the need to rebuild it

---

3. Models in SDLC:

- **Q10:** What is the **Waterfall Model**? List its advantages and disadvantages. In which scenarios is it most effective?

  Answer:

What is the Waterfall Model?

The **Waterfall Model** is a **linear and sequential** software development model where each phase (like requirements, design, coding, testing, etc.) flows **downward** to the next — like a waterfall.

Each phase must be **fully completed** before the next begins, and there's **little to no going back** once a phase is done.

Phases in Waterfall Model:

1. Requirement Gathering
2. System Design
3. Implementation (Coding)
4. Testing
5. Deployment
6. Maintenance

Advantages of the Waterfall Model:

Advantage	Description
Simple & Easy to Understand	Clear structure; each phase has specific deliverables
Well-Documented Process	Everything is properly documented at every stage
Easy to Manage	Progress is measurable phase by phase
Ideal for Fixed Requirements	Works well when requirements are clear and won't change
Better for Small Projects	Especially effective for short-term, low-risk projects

Disadvantages of the Waterfall Model:

Disadvantage	Description
No Flexibility	Difficult to go back once a phase is completed
Late Testing	Bugs found only after the coding phase

Disadvantage	Description
Not Suitable for Changing Needs	Doesn’t handle requirement changes well
High Risk of Failure	If early stages are wrong, the whole project may collapse
Poor User Feedback	User sees the final product only after development is complete

When is the Waterfall Model Most Effective?

Use the Waterfall Model when:

- Requirements are **clearly defined and fixed**
- The project is **short and simple**
- The client **won’t change** their mind often
- Strict **regulatory or documentation** is required (e.g., government, banking)

- **Q11:** Explain the **Agile Model** in SDLC. How does it differ from the Waterfall model, and what are its key principles?

Answer:

What is the Agile Model?

The **Agile Model** is a **flexible and iterative** approach to software development where the project is broken down into **small cycles (called sprints or iterations)**. Each cycle delivers a **working piece of the software**, allowing for **continuous feedback and improvement**.

Key Features of Agile:

- **Frequent releases** of working software
- **Customer collaboration** throughout the process
- **Adaptive to changes** even late in development
- **Cross-functional teams** work together closely

Key Principles of Agile :

1. **Customer satisfaction** through early and continuous delivery
2. **Welcome changing requirements** at any stage
3. **Deliver working software frequently** (every 1–4 weeks)
4. **Close collaboration** between developers and stakeholders
5. **Build projects around motivated individuals**
6. **Face-to-face communication** is most effective
7. **Working software is the primary measure of progress**
8. **Sustainable development pace** is maintained
9. **Continuous attention to technical excellence**
10. **Simplicity** is essential
11. **Self-organizing teams** produce best results
12. **Reflect and adjust** regularly for better performance

Agile Model vs Waterfall Model

Feature	Agile Model	Waterfall Model
Development Style	Iterative & incremental	Linear & sequential
Flexibility	Highly flexible to changes	Rigid, changes are hard to implement
Customer Involvement	Continuous feedback from customer	Involved mostly at beginning and end
Testing	Continuous testing in each sprint	Testing only after development is complete
Delivery	Working software after every sprint	One final product at the end
Documentation	Lightweight, only as needed	Heavy documentation throughout

When to Use Agile:

- When **requirements may change frequently**
- For **complex or large projects**
- When **early delivery** of working features is important
- For **collaborative environments** with strong team interaction

4. Real-World Applications and Scenarios:

- **Q12:** Imagine you are working in a team developing a banking application. Discuss how you would follow the SDLC in your project, focusing on each phase.

  Answer:

  ## 1. Requirement Gathering and Analysis

  - **Goal**: Understand exactly what the bank needs.
  - **Activities**:
    - Meet with stakeholders (bank managers, users).
    - Identify features like: money transfer, balance check, login, account creation, etc.
    - Document functional (what it should do) and non-functional (security, speed) requirements.

  ## 2. Design Phase

  - **Goal**: Plan how the application will work.
  - **Activities**:
    - **High-Level Design (HLD)**: Define overall architecture — frontend, backend, database, APIs.
    - **Low-Level Design (LLD)**: Create detailed designs for modules like login, transactions, UI flow.
    - Choose tech stack (e.g., React + Java + MySQL).

  ## 3. Coding / Implementation Phase

  - **Goal**: Build the application.
  - **Activities**:
    - Developers start writing code for modules like login, account management, money transfer, etc.
    - Use version control (Git), IDEs (VS Code, IntelliJ), and follow coding standards.
    - Code is divided among frontend, backend, and database developers.

  ## 4. Testing Phase

  - **Goal**: Ensure the app works correctly and securely.

- **Activities**:
    - **Unit Testing**: Test each function (e.g., money transfer logic).
    - **Integration Testing**: Ensure modules work together (e.g., login + dashboard).
    - **System Testing**: Test the entire app for performance and accuracy.
    - **Security Testing**: Verify protection of user data and transactions.

5. Deployment Phase

- **Goal**: Launch the app for real users.
- **Activities**:
    - Deploy to a **production server**.
    - Ensure proper database setup and backup.
    - Inform and train bank staff.
    - Monitor initial usage for any unexpected bugs.

6. Maintenance Phase

- **Goal**: Keep the app running smoothly over time.
- **Activities**:
    - Fix bugs reported by users (e.g., login failure).
    - Release updates with new features (e.g., loan calculator).
    - Ensure the app remains compatible with new OS/browser versions.

- **Q13:** You are tasked with developing a mobile app for a fitness tracking company. Create a brief SDLC plan for this project, detailing each phase and the activities involved.

  Answer:

1 Requirement Gathering & Analysis

Objective: Understand what the fitness company and users need.

Activities:

- Meet with stakeholders to define app goals: track workouts, calories, steps, heart rate, etc.
- Gather functional requirements (user registration, GPS tracking, daily goals).
- Identify non-functional needs (security, performance, cross-platform compatibility).
- Analyze competitors' apps to understand best practices.

2 Design Phase

Objective: Create a visual and technical plan for the app.

Activities:

- **High-Level Design (HLD)**:
  - Decide architecture (e.g., client-server, REST API).
  - Choose tech stack (e.g., React Native, Firebase).
- **Low-Level Design (LLD)**:
  - Create wireframes and UI mockups for screens like dashboard, goals, workout log.
  - Plan database schema (e.g., user profiles, workout history).

3 Development / Coding Phase

Objective: Build the app functionality.

Activities:

- Set up frontend (UI/UX) using React Native or Flutter.
- Build backend APIs for data storage, user auth, and syncing (Node.js, Firebase, etc.).
- Implement device features: GPS, accelerometer, health data integration (Google Fit/Apple Health).
- Use version control (Git) and CI/CD tools for faster delivery.

4 Testing Phase

Objective: Ensure the app is stable, secure, and bug-free.

Activities:

- **Unit Testing**: Test individual features (e.g., goal calculation logic).
- **Integration Testing**: Check how modules interact (e.g., login + dashboard).
- **System Testing**: Run the complete app across devices and OS versions.
- **Beta Testing**: Allow selected users to try the app and give feedback.
- **Security Testing**: Verify personal data is safe (encrypted storage, secure login).

5 Deployment Phase

Objective: Release the app to real users.

Activities:

- Publish on **Google Play Store** and **Apple App Store**.
- Set up app analytics and crash reporting.
- Ensure servers and databases are ready for user load.
- Prepare a user guide or in-app tutorial.

6 Maintenance Phase

Objective: Keep the app updated and running smoothly.

Activities:

- Fix bugs reported by users.
- Release feature updates (e.g., challenges, progress badges).
- Monitor performance and crash reports.
- Respond to OS updates or changes in API dependencies.

- **Q14:** In a software development project, the project manager has opted to use the **Agile Model**. How will this affect the roles of the development team and the way the project is managed?

Answer:

When a **project manager chooses the Agile Model** for a software development project, it significantly changes both the **roles of the team** and the **project management approach** compared to traditional methods like Waterfall.

Impact on the Development Team Roles:

Developers

- Work in short iterations (sprints)
- Collaborate directly with testers and product owners

Testers / QA

- Involved from the beginning, not just after coding
- Perform continuous testing (unit, integration, regression)
- Often work closely with developers (test-driven development)

Business Analysts

- Take on the **Product Owner** role in Agile
- Constantly refine and prioritize product backlog based on feedback
- Work closely with both the customer and the development team

Team Members –

All roles become more **cross-functional**
- Everyone contributes to quality, documentation, and delivery
- Shared responsibility for success/failure of sprints |

Impact on Project Management Style:

Aspect	Agile Approach
Planning	Iterative planning (per sprint), flexible scope, welcomes change
Leadership Style	Project manager acts as a **facilitator** or **Scrum Master**, not a top-down controller
Progress Tracking	Uses **burn-down charts**, velocity, and sprint reviews instead of Gantt charts
Customer Involvement	Constant collaboration and feedback through sprint reviews and backlog grooming
Deliverables	Working software is delivered **incrementally** in every sprint
Feedback Loop	Continuous feedback from users and team after each iteration

Benefits of Agile for Team and Management:

- Faster response to changes in requirements
- Better visibility into progress and blockers
- Higher customer satisfaction through regular delivery
- More ownership and motivation among team members

- **Q15:** How would you approach testing in a project that uses the **Waterfall Model**? Compare this with testing in an **Agile Model** project.

  Answer:

Testing in the Waterfall Model

Approach:

- Testing is a **distinct phase** that happens **after the development phase** is fully completed.
- Testers usually have **little involvement** during earlier phases.
- Focuses on verifying if the final product meets the documented requirements.

Key Characteristics:

- **Sequential**: No testing until the full product is developed.
- **Heavy documentation**: Test plans and cases are prepared in advance.
- **Late feedback**: Bugs are found after coding is done, which can delay fixes.

Testing Types Used:

- Unit testing (by developers)
- Integration testing
- System testing
- User acceptance testing (UAT)

Waterfall vs Agile Testing –

Feature	Waterfall Testing	Agile Testing
Timing	After development is complete	Alongside development (in every sprint)
Tester Involvement	Late in the process	Involved from the start
Feedback Loop	Slow	Fast & continuous
Documentation	Heavy, fixed test plans	Light, evolving test scenarios
Test Automation	Less common	Common and encouraged
Change Handling	Difficult and costly	Easy to adapt and retest

- **Q16:** Discuss the challenges you might face in the **Deployment** phase of the SDLC when moving from a development environment to a production environment. How would you overcome these challenges?

  Answer:

1. Environment Differences

- **Problem**: Code works in the development environment but fails in production due to configuration, OS, database, or version mismatches.
- **Solution**:
  - Use **containerization tools** like Docker to ensure consistency.
  - Automate environment setup using tools like Ansible or Terraform.
  - Perform **staging environment testing** that mirrors production.

2. Incomplete Testing

- **Problem**: Code may have bugs not discovered during development or testing.
- **Solution**:
  - Conduct **regression, performance, and user acceptance testing (UAT)** in a pre-production environment.
  - Use **automated test suites** to cover critical paths.
  - Include real-world test data for better simulation.

3. Downtime or Service Disruption

- **Problem**: Deployments can make the application temporarily unavailable.
- **Solution**:
  - Use **blue-green deployment** or **canary releases** to roll out changes gradually.
  - Deploy during low-traffic hours.
  - Set up **rollback procedures** in case of failure.

4. Security and Access Issues

- **Problem**: Sensitive credentials or permissions may be exposed or misconfigured.
- **Solution**:
  - Use **secret management tools** (like Vault, AWS Secrets Manager).
  - Apply the **principle of least privilege** for access control.
  - Conduct **security audits** before deployment.

5. Integration Failures

- **Problem**: Application might fail to interact with third-party services or APIs in production.
- **Solution**:
  - Use **mock environments** or sandbox APIs during testing.
  - Have **monitoring and logging** in place to quickly identify integration issues.

6. Poor Documentation or Handover

- **Problem**: The deployment team may lack necessary knowledge.
- **Solution**:
  - Provide **deployment runbooks** and detailed documentation.
  - Conduct **knowledge transfer sessions** between dev and ops teams.
  - Use **CI/CD pipelines** for repeatable deployments.

7. Lack of Monitoring and Alerts

- **Problem**: Bugs or performance issues go unnoticed after release.
- **Solution**:
  - Set up **real-time monitoring** (e.g., Prometheus, Grafana, New Relic).
  - Implement **log tracking** and **automated alert systems** (e.g., Splunk, ELK).

---

5. SDLC Documentation:

- **Q17:** Create a sample **Test Plan** document for a simple web application. List the key components that should be included in the plan.

  Answer:

1. Test Plan ID

- TP-002-MOBILEBANK-APP

2. Introduction

This Test Plan outlines the strategy, scope, schedule, and resources for testing the **Mobile Banking Application**, which allows users to securely access banking services such as balance checks, fund transfers, transaction history, and bill payments.

3. Objectives and Tasks

- Verify core banking features (login, balance inquiry, transfers)
- Ensure app security and data privacy
- Confirm responsiveness and usability across devices
- Validate performance under load

4. Scope of Testing

In-Scope:

- User Login & OTP Verification
- Account Overview
- Fund Transfers (within & outside bank)
- Transaction History
- Bill Payments
- Notifications & Alerts
- Security (Session timeout, encryption)

Out-of-Scope:

- Backend banking systems
- ATM or branch systems

5. Test Items

- Mobile App UI (Android & iOS)
- Backend API connectivity
- Login, logout, and timeout functionality
- Fund transfer and transaction modules
- Push notifications

6. Testing Types

- **Unit Testing** (by developers)
- **Functional Testing**
- **Security Testing**
- **Usability Testing**
- **Performance Testing**

- **Compatibility Testing** (on different devices/OS)
- **Regression Testing**

7. Roles and Responsibilities

Role	Responsibility
QA Lead	Manage testing lifecycle, schedule, reports
QA Engineers	Prepare and execute test cases
Dev Team	Support with builds and bug fixes
Security Analyst	Perform vulnerability assessments
UAT Users	Validate functionality as end users

8. Schedule / Milestones

Phase	Start Date	End Date
Requirement Review	June 16, 2025	June 17, 2025
Test Planning	June 18, 2025	June 19, 2025
Test Case Design	June 20, 2025	June 22, 2025
Test Execution	June 23, 2025	July 1, 2025
Performance Testing	July 2, 2025	July 3, 2025
UAT	July 4, 2025	July 6, 2025
Final Report & Sign-Off	July 7, 2025	July 8, 2025

9. Test Deliverables

- Test Plan
- Test Cases & Test Data
- Defect Reports
- Test Summary Report
- UAT Sign-Off

- Security Audit Report

10. Environment Requirements

- Android (v10 and above), iOS (v14 and above)
- Secure test servers (mock banking APIs)
- Tools: Appium (UI testing), Postman (API testing), Burp Suite (security), Jira (bug tracking)

11. Risks and Mitigation

Risk	Mitigation
Changing banking regulations	Regular sync with compliance team
Security vulnerabilities	Frequent pen-testing & code audits
Third-party integration failure	Use test stubs for payment gateway, OTP SMS

12. Exit Criteria

- All critical features are tested with pass status
- No critical/severe bugs are open
- All compliance and security checks passed
- UAT sign-off received from the business team

- **Q18:** As a project manager, how would you ensure proper documentation is maintained throughout the SDLC? Discuss tools that can be used for documentation management.

  Answer:

To Ensure Proper Documentation Throughout SDLC:

1. Define Documentation Requirements for Each Phase

- **Requirement Phase**: SRS, use cases, user stories
- **Design Phase**: Architecture diagrams, data flow diagrams
- **Development Phase**: Code documentation, APIs
- **Testing Phase**: Test plans, test cases, bug reports
- **Deployment Phase**: Release notes, user manuals
- **Maintenance Phase**: Change logs, updated manuals

2. Assign Clear Ownership

- Assign roles for writing and reviewing documents (e.g., BA for SRS, QA for test cases).
- Conduct **peer reviews** or documentation walkthroughs regularly.

3. Integrate Documentation into Workflows

- Make documentation part of the **Definition of Done** for each task or sprint.
- Include it in **agile ceremonies** (sprint reviews, retrospectives).

4. Maintain Version Control

- Use versioning for all documents to track changes.
- Set review/approval cycles to maintain accuracy.

Tools for Documentation Management

Tool	**Purpose**	**Features**
Confluence	Centralized documentation wiki	Templates, access control, page versioning
Google Docs	Collaborative document editing	Real-time editing, comments, cloud storage
Notion	Lightweight docs + task management	Flexible pages, embedded databases
Microsoft SharePoint	Enterprise document management	Permissions, workflow automation
Jira (linked to Confluence)	Requirement + issue tracking	Link issues to docs, sprint integration
GitHub / GitLab Wiki	Developer-focused documentation	Markdown support, versioning with code
DocuSign / Adobe Acrobat	For formal approvals	E-signatures, audit trail

Best Practices

- Use **templates** for consistency.
- **Automate** documentation updates where possible (e.g., generate API docs using Swagger).
- Conduct **periodic audits** to remove outdated or redundant documents.
- Store documents in a **centralized, accessible repository**.

---

6. SDLC in Agile:

- **Q19:** Create a simple **user story** for an e-commerce website project. Explain how this story fits into the **Agile** development cycle.

  Answer:

Title: User Login Functionality

As a

Registered user

I want to

log in to my account using my email and password

So that

I can view my order history and track my current orders

Acceptance Criteria

1. User can enter email and password on the login page.
2. The system verifies credentials with the backend.
3. If correct, the user is redirected to the dashboard.
4. If incorrect, an error message is shown.
5. A "Forgot Password" link is available.

How This Fits into the Agile Development Cycle

Agile Phase	How the User Story Fits
1. Backlog Grooming	Product owner adds the login user story to the product backlog and refines the details.
2. Sprint Planning	Team selects the login story for the current sprint and breaks it into smaller tasks.
3. Design & Dev	Developers design the UI and implement the login feature with backend integration.
4. Testing	QA tests the login functionality based on the acceptance criteria.
5. Review/Demo	The login feature is demoed to stakeholders at the sprint review.
6. Deployment	The login feature is deployed in a staging or production environment.
7. Retrospective	Team reflects on what went well and how the login feature was developed.

---

7. Quality Assurance and Testing in SDLC:

- **Q20:** Write a **Test Case** for a login page on a website. Include the steps, expected results, and pass/fail criteria.

  Answer:

Test Case ID

TC_001_Login_Valid_Credentials

Test Scenario

Verify that a user can log in successfully using valid email and password.

Tested By

Tanya Chaudhary

Test Date

18-06-2025

Preconditions

- User is registered on the website.
- User is on the login page (example.com/login).

Test Steps

Step No.	Action	Test Data
1	Open the login page	-
2	Enter valid email in the email input field	user@example.com
3	Enter valid password in the password field	ValidPassword123
4	Click the "Login" button	-

Expected Result

- User should be redirected to their dashboard/homepage.
- A welcome message like “Welcome, [User Name]” should be displayed.

Pass/Fail Criteria

- **Pass**: If login is successful and the user is redirected correctly.
- **Fail**: If the user is not redirected, or an error message is shown despite valid credentials.

Postconditions

- User is logged in and session is active.

Test Status

Pass / Fail (select after execution)

---

8. Risk Management in SDLC:

- **Q21:** During the **Testing** phase, your team discovers a critical bug that requires significant changes to the design. How would you handle this issue, considering the SDLC process?

  Answer:

  ### 1. Analyze and Confirm the Bug

  - Reproduce the issue and confirm it's critical (e.g., causes data loss, app crash, security breach).
  - Categorize the bug severity and impact.

  ### 2. Communicate with Stakeholders

  - Notify project stakeholders, including the **project manager**, **developers**, **designers**, and **product owner**.
  - Discuss the **impact on timelines**, scope, and cost.

  ### 3. Re-enter the SDLC – Return to the Design Phase

  - Since it involves a design-level flaw, **loop back** to the **Design phase** of SDLC.
  - Redesign the affected component(s) to address the issue properly.
  - Document the design changes (update architecture diagrams, specifications, etc.).

  ### 4. Update Development

  - Implement the new design in code.
  - Perform **unit testing** and **code review** for the changed modules.

  ### 5. Retest the Application

  - Return to the **Testing phase** to:
    - Run **regression tests**

    - Validate the bug fix
    - Ensure that changes didn’t break other parts of the system

6. Update Project Plan & Timeline

- Adjust the **project schedule**, and communicate new deadlines if necessary.
- Update the **risk register** and mitigation strategies.

7. Maintain Documentation

- Update all related documentation: SRS, design documents, test cases, change logs.
- Ensure traceability of changes (who made them, why, and when).

---

9. Continuous Integration and Continuous Deployment (CI/CD):

- **Q22:** Implement a simple **CI/CD pipeline** for a sample web application. Explain the stages involved, from code commit to deployment.

  Answer:

What is CI/CD?

- **CI (Continuous Integration)**: Automatically builds and tests code every time a developer commits.
- **CD (Continuous Deployment/Delivery)**: Automatically delivers or deploys the app to a staging or production server.

Tech Stack Assumptions

- Frontend: React
- Backend: Node.js + Express
- Version Control: GitHub
- CI/CD Tool: GitHub Actions (or alternatives like Jenkins, GitLab CI, CircleCI)
- Deployment: Render, Netlify, Vercel, or AWS EC2

CI/CD Pipeline Stages

1 Code Commit

- Developer pushes code to a GitHub repository (e.g., main or dev branch).

2 CI: Build & Test

Pipeline Trigger: On push or pull request
Steps:

- **Checkout Code**
- **Install Dependencies** (e.g., npm install)
- **Run Linting/Code Quality Checks**
- **Run Unit Tests** (e.g., npm test)
- **Build the App** (e.g., npm run build for React)

Tools Used:

- GitHub Actions Workflow (.github/workflows/main.yml)
- npm, Jest, ESLint

Output: A tested and built artifact ready for deployment

3 CD: Deployment to Staging/Production

Trigger: Successful build from CI
Steps:

- Connect to server or cloud platform (e.g., Render, Netlify, Vercel, AWS)
- Upload build files (or use CLI/SSH to deploy backend)
- Restart the server or application

Tools/Platforms:

- For frontend: Netlify, Vercel, or GitHub Pages (auto deploy from GitHub)
- For backend: Render, Heroku, or manual deploy via SSH to EC2 or VPS

Output: Web app is live on the staging or production environment

10. SDLC Best Practices:

- **Q23:** As a developer, how can you ensure that your code is maintainable and scalable throughout the SDLC? Discuss techniques such as modular coding, commenting, and versioning.

Answer:

1. Modular Coding (Separation of Concerns)

What it means:

Break your application into small, reusable, and independent components or modules (e.g., functions, classes, services).

Benefits:

- Easier to debug and test
- Reduces code duplication
- Makes adding new features easier without breaking existing functionality

Example:

```
// BAD: Monolithic code
function handleEverything() {
  // logic for auth, DB, API, UI all in one
}

// GOOD: Modular code
authService.login()
database.saveUser()
ui.renderDashboard()
```

2. Proper Commenting & Documentation

What it means:

Write meaningful comments to explain *why* the code does something, not just *what* it does. Also, maintain README files, API docs, and inline comments.

Benefits:

- Helps other developers (and you) understand the purpose and logic
- Reduces onboarding time for new team members

- Helps in debugging and audits

Example:

python
Copy code

```
# Calculate interest for a given principal and rate
def calculate_interest(principal, rate):
    return principal * rate
```

3. Version Control with Git

What it means:

Use tools like Git to manage code changes with branches**,** commits**,** and tags**.**

Benefits:

- Keeps a history of changes
- Makes collaboration safe (via feature branches)
- Enables rollback in case of bugs

Tips:

- Use meaningful commit messages (e.g., `fix: corrected null pointer issue in login`)
- Create feature branches (e.g., `feature/payment-module`)

4. Follow Coding Standards & Naming Conventions

What it means:

Use a consistent code style throughout the project (e.g., indentation, variable naming, file structure).

Benefits:

- Makes code predictable and easier to read
- Reduces merge conflicts
- Improves team collaboration

Tools:

- ESLint, Prettier (JS)

- PEP8 (Python)
- StyleCop (C#)

5. Write Tests (Unit, Integration)

What it means:

Use automated tests to verify that each part of your application behaves correctly.

Benefits:

- Reduces bugs during future changes
- Documents how your code is supposed to work
- Gives confidence during refactoring

6. Use Design Patterns Wisely

What it means:

Apply proven solutions to common software design problems (e.g., Singleton, MVC, Factory).

Benefits:

- Makes code structure more predictable
- Encourages best practices
- Improves scalability and extensibility

7. Refactor Regularly

What it means:

Continuously improve code quality by cleaning up bad design, renaming unclear variables, and removing redundant code.

Benefits:

- Keeps codebase clean and healthy
- Improves performance and readability

- Makes future development faster

8. Use Scalable Architecture

What it means:

Choose architectures that support growth (e.g., microservices, layered architecture).

Benefits:

- Easy to add new features or services
- Better performance under load
- Allows teams to work on different modules independently