SDLC Assignment Questions

1. Introduction to SDLC:

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

  Answer:

  The **Software Development Life Cycle (SDLC)** is a systematic process used by software engineers and project managers to design, develop, test, deploy, and maintain software. It provides a structured approach to building software applications by dividing the overall process into specific, manageable phases.

  Each phase has its own set of goals and deliverables that contribute to the successful creation of a high-quality software product.

  Key Phases of SDLC:

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

   These phases ensure that the development process remains organized, traceable, and efficient.

Importance of SDLC in Software Development

1. **Structured Process:**
   SDLC offers a clear roadmap, making software development more predictable and manageable.
2. **Improved Planning and Scheduling:**
   Each phase has defined activities and timelines, which helps in proper project estimation and resource allocation.
3. **Better Quality and Reduced Risk:**
   Since testing and feedback are part of the cycle, issues are identified and resolved early, minimizing the risk of project failure.
4. **Enhanced Communication:**
   It ensures better collaboration among team members, clients, testers, and developers by aligning expectations at every stage.

5. **Cost Efficiency:**
   By catching issues early in the cycle, SDLC helps avoid costly post-deployment fixes.
6. **Documentation and Traceability:**
   Every phase is well-documented, allowing teams to track progress and revisit decisions when needed.

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

  Answer:

  The SDLC is divided into key phases, each contributing to the development of reliable and efficient software.

  1. Requirement Gathering and Analysis

  In this phase, all necessary requirements from the stakeholders are collected and analyzed.

  - *Contribution:* Defines the project scope and aligns software goals with user needs.

  2. System Design

  This phase involves creating the architecture and detailed design of the software.

  - *Contribution:* Provides a clear plan for development and reduces risks during coding.

  3. Development (Coding)

  Developers write code based on the design using programming languages and tools.

  - *Contribution:* Converts design into a functional product.

  4. Testing

  The software is tested to find and fix bugs or mismatches with requirements.

  - *Contribution:* Ensures the software is reliable, bug-free, and ready for use.

  5. Deployment

  The application is released to the production environment for actual use.

  - *Contribution:* Makes the software available to users in a real-world setting.

  6. Maintenance

  Post-deployment, updates and fixes are applied as needed.

  - *Contribution:* Keeps the software running smoothly and relevant over time.

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

  Answer:

  ### 1. Waterfall Model

  Definition:

  A linear and sequential model where each phase i.e.
  (Requirement → Design → Development → Testing → Deployment)
  must be completed before the next begins.

  Use Case:

  Best suited for **small or well-defined projects** with fixed requirements (e.g., building internal tools or static websites).

  Key Point:

  Easy to manage, but **not flexible** to changes once the process starts.

  ### 2. Agile Model

  Definition:

  An iterative and incremental model where development is done in short cycles called **sprints**. Requirements and solutions evolve through collaboration.

  Use Case:

  Ideal for **projects with changing requirements**, such as mobile apps or startups where customer feedback is frequent.

  Key Point:

  Highly flexible and encourages continuous improvement and fast delivery.

  ### 3. V-Model (Validation and Verification Model)

  Definition:

  An extension of the Waterfall model where each development phase has a corresponding testing phase. Testing is planned in parallel with development.

  Use Case:

  Best for **critical systems** like healthcare, aviation, or banking where **thorough testing** is mandatory.

  Key Point:

  Ensures **early detection of defects** through validation at every stage.

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:

  The **Requirement Gathering** phase is the **first and one of the most crucial steps** in the Software Development Life Cycle (SDLC). During this phase, the development team interacts with stakeholders to **understand what the software should do**.

  It focuses on identifying:

- Functional requirements (features, workflows)
- Non-functional requirements (performance, security, usability)

Common Methods to Gather Requirements:

1. **Stakeholder Interviews:**
   One-on-one discussions with clients or end-users to understand their needs.
2. **Questionnaires and Surveys:**
   Distribute forms to gather inputs from a large group of users.
3. **Workshops and Brainstorming:**
   Interactive sessions where developers, users, and business analysts discuss requirements.
4. **Observation:**
   Study how users perform tasks in their current system to identify improvements.
5. **Document Analysis:**
   Review existing manuals, software, or reports to understand current processes.
6. **Prototyping:**
   Build a rough version of the system to help stakeholders visualize and refine their needs.

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

  Answer:

  The **Design Phase** of SDLC focuses on converting the gathered requirements into a blueprint or plan for building the software. It defines how the system will look and work internally and externally.

  Key Activities Involved:

1. **System Architecture Design:**
   Decide on the architecture style (e.g., client-server, microservices).

2. **Technology Stack Selection:**
   Choose the programming languages, frameworks, databases, and tools.
3. **Database Design:**
   Design entity-relationship (ER) diagrams and schema.
4. **UI/UX Design:**
   Create mockups or wireframes for screens and user flows.
5. **Security and Performance Planning:**
   Define how the system will handle sensitive data and scale under load.

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

Aspect	High-Level Design (HLD)	Low-Level Design (LLD)
Focus	System architecture and overall structure	Internal logic and detailed components
Output	Module diagrams, technology stack, APIs	Class diagrams, method details, pseudocode
Audience	Architects, Project Managers	Developers
Detail Level	Abstract, general overview	Detailed, developer-focused

- **Q6:** Explain the **Coding** or **Development** phase of the SDLC. What tools and techniques are typically used by developers during this phase?
  Answer:
  The **Coding or Development phase** is the stage where actual software is built based on the finalized design documents. Developers write code to implement the features and functionalities specified during the design phase.
  This phase transforms design into a **working product** using appropriate programming languages, frameworks, and tools.
  Key Activities:
- Writing code for modules and components
- Integrating different parts of the application
- Following coding standards and best practices

- Conducting peer reviews and version control
- Preparing documentation for the written code

Common Tools and Techniques Used:

1. **Integrated Development Environments (IDEs):**
    - VS Code, IntelliJ IDEA, Eclipse, PyCharm
2. **Version Control Systems:**
    - Git, GitHub, GitLab, Bitbucket (for tracking code changes)
3. **Programming Languages & Frameworks:**
    - Java, Python, C++, JavaScript, React, Angular, Node.js, etc.
4. **Code Quality & Linting Tools:**
    - ESLint, SonarQube (for maintaining clean and readable code)
5. **Build Tools:**
    - Maven, Gradle, Webpack (to compile and bundle code)

- **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:

  The Testing phase is crucial in SDLC as it ensures the software works as intended, is free of defects, and meets the specified requirements. It helps detect bugs early, improves product quality, and builds confidence before deployment.

  Without proper testing, software may fail in real-world use, leading to poor user experience, data loss, or system crashes.

  Types of Testing:

1. **Unit Testing:**
    - Tests individual components or functions.
    - Done by developers.
    - Ensures each unit works correctly in isolation.
2. **Integration Testing:**
    - Verifies how different modules interact.
    - Detects issues in data flow between components.
3. **System Testing:**
    - Tests the complete system as a whole.
    - Checks functionality, performance, and reliability.
4. **Acceptance Testing:**
    - Validates the software against business requirements.
    - Usually performed by end-users or stakeholders.

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

  Answer:

  The **Deployment phase** is where the fully tested software is released and made available to the end-users or moved into a live/production environment. It involves installing, configuring, and enabling the application for real-world use.

  Key Considerations for Successful Deployment:

1. **Environment Readiness:**
   - Ensure production infrastructure (servers, databases) is properly set up and matches staging/test environments.
2. **Deployment Strategy:**
   - Choose the right method:
     - *Blue-Green Deployment*
     - *Rolling Updates*
     - *Canary Releases*
3. **Backup and Rollback Plan:**
   - Always have backups and a rollback strategy in case something goes wrong post-deployment.
4. **Monitoring and Logging:**
   - Use tools (like Prometheus, ELK Stack) to track system behavior and errors after deployment.
5. **User Communication:**
   - Notify users of scheduled downtime, new features, or any major changes.
6. **Security Measures:**
   - Secure the environment with proper authentication, access control, and encrypted data transfers.

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

  Answer:

  The **Maintenance phase** begins after the software is deployed and used by real users. It involves:

1. **Bug Fixes:**
   - Correcting issues reported by users or discovered post-deployment.
2. **Updates and Enhancements:**
   - Adding new features, improving usability, or adapting to user feedback.
3. **Performance Optimization:**

- Enhancing speed, efficiency, and scalability of the software.

4. **Security Patches:**
   - Addressing new vulnerabilities to protect user data and system integrity.
5. **Adaptation to Changes:**
   - Adjusting the software for new hardware, OS updates, or regulatory compliance.

Importance of Maintenance Phase:

- **Ensures software remains functional and relevant** in a changing environment.
- **Improves user satisfaction** by fixing issues quickly.
- **Extends software life**, reducing the need for complete redevelopment.
- **Supports business growth** by adapting to new needs and technologies.

3. Models in SDLC:

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

  Answer:

  The **Waterfall Model** is a traditional linear and sequential approach to software development. Each phase (Requirement → Design → Development → Testing → Deployment → Maintenance) flows in a fixed order, with little to no overlap.

  Advantages:

- **Simple and Easy to Understand**
- **Structured Approach** – clear documentation and milestones
- **Good for Small Projects** with fixed requirements
- **Easier to Manage** due to predictable process

  Disadvantages:

- **Rigid and Inflexible** – difficult to go back to a previous phase
- **Late Testing** – bugs found after full development
- **Poor for Evolving Requirements**
- **High Risk** in long-term projects

  When is the Waterfall Model Most Effective?

- When **requirements are clearly defined** and unlikely to change
- For **short-term projects** with low complexity

- In **regulated industries** where documentation and compliance are critical (e.g., construction, defense).

- **Q11:** Explain the **Agile Model** in SDLC. How does it differ from the Waterfall model, and what are its key principles?
  Answer:
  The **Agile Model** is an iterative and flexible approach to software development where the project is divided into small, manageable units called **sprints**. Continuous feedback, collaboration, and adaptability are key to Agile development.

  Difference between Waterfall and Agile Model

Feature	Waterfall Model	Agile Model
Approach	Sequential	Iterative & Incremental
Flexibility	Rigid – changes are hard	Highly flexible – welcomes changes
Customer Involvement	Limited after requirements phase	Continuous involvement throughout
Testing	Done after development	Done in every sprint
Delivery	Single final product	Frequent working builds (iterations)

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:
  Here's how each phase would be followed:
  1. Requirement Gathering
- Conduct meetings with stakeholders (bank managers, compliance officers, users)
- Identify key features: account login, money transfer, transaction history, loan management

- Gather **security requirements** like two-factor authentication (2FA), encryption, and audit logging

2. System Design

- **High-Level Design:** Define architecture (e.g., 3-tier architecture: frontend, backend, database), select tech stack like Java + Spring Boot + React
- **Low-Level Design:** Create detailed workflows for fund transfer, login validation, and error handling; prepare database schema for accounts and transactions

3. Development (Coding)

- Developers write backend APIs and frontend UI
- Follow secure coding practices to prevent SQL injection, cross-site scripting, etc.
- Use Git for version control and CI tools for automatic builds

4. Testing

- **Unit Testing** for functions like balance check or interest calculation
- **Integration Testing** to ensure frontend-backend coordination
- **System Testing** for complete banking workflows
- **Security Testing** for vulnerabilities and compliance

5. Deployment

- Deploy the application to a secure cloud environment (e.g., AWS with HTTPS and firewall)
- Conduct final user acceptance testing (UAT) with real users
- Use tools like Docker, Kubernetes for scalable deployment

6. Maintenance

- Monitor the system for downtime, bugs, or suspicious activity
- Roll out patches for security vulnerabilities
- Add new features like investment tracking or chatbot support based on user feedback

- **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:

  Here's a brief plan for each phase:

  ### 1. Requirement Gathering

  - Meet with the fitness company to understand goals
  - Identify features: step counter, calorie tracker, workout log, user goals

- Collect both functional (step count, workout upload) and non-functional (performance, battery optimization) requirements
- Use surveys, competitor analysis, and user personas.

2. Design

- **High-Level Design (HLD):** Decide on cross-platform approach (e.g., Flutter), database (e.g., Firebase), and backend services
- **Low-Level Design (LLD):** Create wireframes for the UI, screen navigation flow, data models for user stats, and notification system

3. Development

- Set up the development environment using Flutter or React Native
- Use Firebase or MongoDB for backend
- Break development into modules (authentication, dashboard, activity tracking)
- Integrate APIs for health data and GPS tracking

4. Testing

- **Unit Testing:** Validate individual features like step count logic
- **UI Testing:** Ensure screen responsiveness and layout consistency
- **Integration Testing:** Test syncing with wearables and backend services
- **User Acceptance Testing (UAT):** Gather feedback from real users and trainers

5. Deployment

- Prepare for release on Google Play Store and Apple App Store
- Set up CI/CD pipeline for quick future updates
- Monitor app performance and crash reports using tools like Firebase Crashlytics

6. Maintenance

- Regularly update app for OS compatibility and new features
- Fix bugs based on user reviews
- Add enhancements like AI workout suggestions or personalized plans
- Monitor user engagement and retention metrics

- **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:

Agile Model helps in development as:

1. Roles Become More Collaborative

- **Developers** don't just write code — they work closely with testers, designers, and product owners throughout each sprint.
- **Testers** test continuously instead of waiting until the end.
- **Product Owner** acts as the voice of the customer and manages the product backlog.

2. Project Management is Iterative

- The project is broken into **sprints** (typically 1–4 weeks long), each delivering a small but working piece of software.
- **Sprint Planning** helps define clear short-term goals.
- **Daily Stand-ups** ensure real-time tracking of progress and quick resolution of issues.

3. Flexibility and Customer Involvement

- Changes in requirements are welcomed even late in development.
- The project manager focuses more on **facilitation and team empowerment** than strict control.
- Customers or stakeholders are **actively involved** in reviews and prioritization.

4. Documentation is Lightweight but Ongoing

- Unlike Waterfall, Agile prefers **just enough documentation** — updated regularly but not overly detailed.

- **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:

- **Sequential Approach:** Testing begins **only after** the development phase is fully completed.
- **Predefined Test Plan:** A detailed test plan is created during the planning stage.
- **Limited Flexibility:** Changes are difficult once testing starts.
- **Types of Testing:** Mostly focuses on **system testing** and **user acceptance testing**.
- **Responsibility:** Testing is done by a **dedicated QA team**, often isolated from developers.
- **Feedback Loop:** Delayed; bugs found late may require major rework.

Comparison Summary:

Aspect	Waterfall Model	Agile Model
Timing	After development	Throughout development (each sprint)
Flexibility	Low	High
Feedback	Late	Early and continuous
Collaboration	Limited (separate teams)	High (cross-functional teams)
Bug Fixing	Costly and late	Quicker and cheaper
Documentation	Heavy test documents	Lightweight and evolving

- **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:

  Challenges in Deployment Phase:

1. **Environment Differences:**
   The development, staging, and production environments might have configuration mismatches, causing bugs that didn't appear during testing.
2. **Downtime and Service Disruption:**
   Deploying updates can lead to application downtime, which may affect users.
3. **Rollback Complexity:**
   If the new deployment fails, rolling back to a previous version can be difficult without a proper strategy.
4. **Security Issues:**
   Improper configuration or unsecured deployment can expose vulnerabilities in the production system.
5. **Data Migration Errors:**
   Moving large volumes of data or updating databases might lead to loss or corruption if not handled properly.

Following approaches are used to overcome these challenges:

1. **Use Staging Environments:**
Create a staging environment identical to production to test the final version before going live.
2. **Automated CI/CD Pipelines:**
Implement Continuous Integration/Continuous Deployment tools like Jenkins, GitHub Actions, or GitLab CI to automate deployments and reduce human errors.
3. **Blue-Green Deployment Strategy:**
Maintain two environments (blue and green); switch traffic to the new environment only after confirming it's stable.
4. **Monitor and Log:**
Use monitoring tools (e.g., Prometheus, New Relic) and logging tools (e.g., ELK Stack) to quickly detect issues after deployment.
5. **Backup and Rollback Plan:**
Always back up data and application versions before deployment. Use version control and containerization (e.g., Docker) for quick rollbacks.
6. **Deployment Checklist:**
Use a standardized checklist to ensure all steps are followed correctly, such as verifying permissions, dependencies, and network configurations.

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 Title:
Test Plan for User Authentication Module of a Web Application

2. Test Objective:
To verify the functionality, performance, and security of the login and registration features of the web application.

3. Scope of Testing:

- Login page validation
- Registration form validation
- Forgot password functionality
- Browser compatibility

- Input validations (email format, password length, etc.)

4. Test Items:

- Login form
- Sign-up form
- Password reset page
- User session handling

5. Features to be Tested:

- Correct redirection after login
- Error messages on invalid input
- Field-level validation
- Security (SQL injection, XSS prevention)
- Responsive design on multiple devices

6. Features Not to be Tested:

- Admin module
- Backend database optimization
- Load testing (for this phase)

7. Test Strategy:

- **Testing Type:** Manual and Automated Testing
- **Test Levels:** Unit, Integration, and System Testing
- **Test Design Techniques:** Equivalence partitioning, boundary value analysis

8. Test Environment:

- Browsers: Chrome, Firefox, Edge
- Operating Systems: Windows 10, Android, iOS
- Tools: Selenium for automation, Postman for API testing

9. Roles and Responsibilities:

- **Test Lead:** Prepare test plan and supervise the testing process
- **Testers:** Execute test cases, report bugs
- **Developers:** Fix reported issues and support testing

10. Entry and Exit Criteria:

- **Entry:** All components are unit tested and integrated; test environment is ready

- **Exit:** All critical bugs fixed; test cases executed successfully; test summary report approved

11. Deliverables:

- Test Plan Document
- Test Cases
- Bug Reports
- Test Summary Report

12. Schedule:

Activity	Duration	Start Date	End Date
Test Planning	2 Days	June 16	June 17
Test Case Design	3 Days	June 18	June 20
Test Execution	4 Days	June 21	June 24
Reporting & Closure	2 Days	June 25	June 26

- **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:

Proper documentation ensures that all phases of the SDLC are clearly recorded, making the development process transparent, maintainable, and traceable. It helps new team members onboard quickly and ensures stakeholders understand the system.

Proper documentation is ensured by:

1. **Define Documentation Standards:**

- Establish a consistent format and structure for all documents.
- Use templates for requirement specs, design documents, test plans, etc.

2. **Assign Documentation Responsibilities:**

- Allocate tasks to team members (e.g., developers for design docs, QA for test cases).

3. **Conduct Regular Reviews:**

- Schedule periodic documentation reviews to ensure accuracy and completeness.
- Involve stakeholders for validation and approvals.

4. **Version Control:**

- Maintain document history and change logs.

- Track revisions with date, author, and summary of changes.

5. **Centralized Repository:**

- Store all documents in one accessible platform.
- Ensure access permissions for security.

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:

  User Story:

  As a registered customer,

  I want to **add products to my shopping cart**

  So that I can **review them before making a purchase.**

  Acceptance Criteria:

- User can click "Add to Cart" from the product page.
- Cart shows a list of all selected products with prices.
- User can update quantities or remove items.
- Cart data persists during the session.

How This User Story Fits into the Agile Development Cycle:

1. **Product Backlog:**
   - This user story is added to the product backlog by the **Product Owner**.
2. **Sprint Planning:**
   - During a sprint planning meeting, the team selects this story for the next sprint.
   - The story is broken down into tasks like UI design, backend API, database update, and testing.
3. **Development:**
   - Developers implement the "Add to Cart" functionality during the sprint.
4. **Testing:**
   - Testers verify the cart behavior using the defined acceptance criteria.
5. **Sprint Review:**
   - The completed feature is demonstrated to stakeholders for feedback.
6. **Sprint Retrospective:**
   - The team reflects on what went well and what can be improved for the next sprint.

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 Details:
- **Test Case ID:** TC_Login_01
- **Module:** User Authentication
- **Priority:** High
- **Precondition:** User must have an existing registered account.
- **Test Environment:** Chrome browser, Windows OS, Internet connection
- **Open the login page** of the website.
- **Enter valid username and password**, then click "Login."
- **Enter invalid credentials** and attempt to log in.
- **Leave both fields empty** and click "Login."
- **Enter SQL injection string** in input fields and click "Login."
- **Click on the "Forgot Password"** link.

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. Log and Prioritize the Bug
- Record the bug in a defect tracking tool (e.g., Jira, Bugzilla)
- Assign it **critical severity and high priority**

  ### 2. Conduct Root Cause Analysis (RCA)
- Collaborate with the **development and design teams**
- Identify whether the issue stems from requirement misinterpretation, flawed logic, or architectural gaps

  ### 3. Revisit the Design Phase
- Go back to the **design documentation** and update it to address the flaw
- Make sure both **high-level** and **low-level designs** reflect the necessary changes

  ### 4. Modify the Code Accordingly

- Update the impacted modules based on revised design
- Ensure coding follows updated architecture and does not break other features

5. Retest the System Thoroughly

- Perform **regression testing** to check if the fix caused any new issues
- Re-run **unit**, **integration**, and **system tests** on the modified areas

6. Communicate with Stakeholders

- Notify the **project manager, QA lead, and client** about the impact
- Update the timeline, cost estimates, and risk register accordingly

7. Document the Changes

- Update the **design documents, test cases, and bug reports**
- Maintain version control and proper documentation trail

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:

  Stages in the CI/CD Pipeline:

1. Code Commit (Source Stage):

- Developer writes code and **pushes changes to a GitHub repository**.
- A CI/CD pipeline is triggered on events like push or pull request.

2. Build Stage:

- GitHub Actions starts a workflow to **build the application**.
- Example: For a React app, it runs npm install and npm run build.
- Ensures code compiles correctly and all dependencies are installed.

3. Test Stage:

- Automated tests are run (e.g., unit or integration tests) using Jest, Mocha, etc.
- Ensures new code doesn't break existing features.
- If tests fail, the pipeline stops here.

4. Deployment Stage:

- If tests pass, the code is **automatically deployed** to a live environment.
- For example:
    - **Heroku:** git push heroku main
    - **Netlify:** Auto-deploy from GitHub
    - **AWS/GCP:** Use CLI or pipeline integration

5. Monitoring & Notification (Optional):

- Slack, Email, or GitHub notifications alert the team about the build status.
- Integration with tools like **Sentry** or **New Relic** can monitor app performance.

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

- Break the application into **independent, reusable components** or modules.
- Follows the **Single Responsibility Principle** — each module should do one thing well.
- Easier to debug, test, and extend features without affecting the whole codebase.
  Example: In a web app, separate modules for user login, dashboard, and payment processing.

2. Meaningful Commenting

- Add **clear, concise comments** to explain *why* something is done, not just *what*.
- Helps other developers (and your future self) understand the logic quickly.
- Use **inline comments for logic** and **docstrings for functions/classes**.

3. Consistent Naming Conventions

- Use clear, **descriptive variable and function names**.
- Follow naming standards like camelCase, PascalCase, or snake_case, depending on the language.
- Makes code intuitive and reduces the need for excessive comments.

4. Version Control

- Use Git for **tracking changes**, maintaining history, and collaboration.
- Create **separate branches** for features or bug fixes and merge only after testing.

- Tag versions (v1.0, v2.1) for stable releases and easier rollbacks.

5. Code Reusability

- Write **generic utility functions** and avoid repetition (DRY principle — Don't Repeat Yourself).
- Use libraries or frameworks where applicable to avoid reinventing the wheel.

6. Design Patterns

- Apply proven **software design patterns** like MVC, Singleton, Factory, etc., for solving common problems in a scalable way.
- Helps maintain consistency and improves architecture.

7. Regular Code Reviews

- Participate in peer reviews to catch issues early and share knowledge.
- Encourages writing cleaner, more efficient code and fosters accountability.

8. Documentation

- Maintain up-to-date **README files, API docs**, and inline documentation.
- Use tools like JSDoc, Sphinx, or Swagger for structured documentation.

9. Unit and Integration Testing

- Write **automated tests** for individual units and integrations.
- Helps in safely refactoring and extending code later.

10. Scalability Planning

- Optimize for performance and memory usage.
- Keep the codebase **loosely coupled** to easily scale parts of the system (e.g., microservices or serverless functions).