1. Basics of SDLC

1. What is SDLC? Why is it important?
   The Software Development Life Cycle (SDLC) is a framework that defines the stages involved in the creation of software, from its initial conception to its final deployment and maintenance. It's a structured approach that outlines the entire process, including planning, analysis, design, development, testing, and deployment.
   Importance:
   - **Structure and Control:** SDLC provides a clear roadmap, ensuring that all team members understand their roles and responsibilities. This structure helps in managing complex projects and maintaining control over the process.
   - **Improved Quality:** By following a systematic process, teams can identify and fix issues early, leading to higher-quality software that meets user requirements.
   - **Predictability:** It helps in estimating costs, resources, and timelines more accurately, which is crucial for project management and stakeholder communication.
   - **Risk Management:** SDLC models often incorporate phases for risk assessment and mitigation, allowing teams to proactively address potential problems.
   - **Increased Efficiency:** A well-defined process reduces rework and streamlines communication, leading to more efficient development cycles.
2. What are the different phases of SDLC?
   The phases of a typical SDLC are:
   - **1. Requirement Gathering & Analysis:** The first step is to collect and analyze requirements from stakeholders to understand what the software needs to do. This results in documents like the Business Requirement Document (BRD) and Software Requirement Specification (SRS).
     - **Example:** For an e-commerce website, requirements would include "users must be able to add items to a cart," "the system should process payments securely," and "the site must handle 10,000 concurrent users."
   - **2. Design:** This phase focuses on creating the architecture and overall design of the system. This includes defining modules, databases, user interfaces, and system integrations. It's divided into High-Level Design (HLD) and Low-Level Design (LLD).
     - **Example:** Creating a database schema for the e-commerce site, designing the checkout page's layout, and defining the API endpoints for payment processing.
   - **3. Development (or Implementation):** In this phase, developers write the code for the software based on the design documents. This is where the actual product is built.
     - **Example:** Writing Python code to handle user authentication or developing the frontend using React for the product catalog.
   - **4. Testing:** The software is rigorously tested to identify defects and ensure it meets all the specified requirements. This includes various types of testing, such as unit testing, integration testing, system testing, and user acceptance testing (UAT).

        - **Example:** Testers clicking through the e-commerce site, trying to break the payment process, and verifying that search functionality works as expected.
    - **5. Deployment:** The final software is released to the production environment, making it available to end-users. This phase includes planning the release, installing the software, and configuring it.
        - **Example:** Deploying the e-commerce application to cloud servers like AWS or Google Cloud Platform, and making the domain (e.g., www.my-store.com) live.
    - **6. Maintenance:** After deployment, the software needs ongoing support. This phase involves addressing new issues, adding new features, and making improvements.
        - **Example:** Releasing a patch to fix a bug in the payment gateway or adding a "wishlist" feature based on user feedback.
3. **Difference between SDLC, STLC, and Agile.**
    - **SDLC (Software Development Life Cycle):** The overarching process for developing software from start to finish. It's the "what" and "how" of building software. It's a broad term that encompasses all phases.
    - **STLC (Software Testing Life Cycle):** A subset of the SDLC. It's a specific, structured process for performing software testing. It outlines the testing phases, from requirement analysis to test closure, and is focused on quality assurance. STLC happens within the SDLC's testing phase.
    - **Agile:** A methodology or a mindset for managing projects, not a rigid, linear process like traditional SDLC models. Agile focuses on iterative and incremental development, frequent collaboration, and adapting to change. It's a way of implementing the SDLC, emphasizing flexibility over strict planning.
4. **What are the most common SDLC models?**
    - **Waterfall Model:** A linear, sequential model where each phase must be completed before the next one begins.
    - **V-Model:** An extension of the Waterfall Model that emphasizes the relationship between each development phase and its corresponding testing phase.
    - **Iterative Model:** A model that involves repeating a cycle of development to produce a new version of the software with each iteration.
    - **Spiral Model:** A risk-driven model that combines the iterative approach of prototyping with the systematic aspects of the Waterfall model.
    - **Agile Model:** A group of methodologies (like Scrum, Kanban) that focus on continuous iteration, flexible planning, and customer collaboration.
5. Which SDLC model is best for your project?
   There is no single "best" model; the choice depends on the project's characteristics:
    - **Waterfall:** Best for small, simple projects with well-understood and stable requirements, where there is a low risk of change.
    - **Agile/Scrum:** Best for projects with unclear or changing requirements, where a fast time-to-market is crucial, and where collaboration is key. It's widely used today for complex, large-scale projects.
    - **V-Model:** Ideal for projects where quality is a top priority, such as safety-critical systems (e.g., medical devices, aerospace software), because of its emphasis on rigorous testing.

- **Spiral:** Best for large, complex, and high-risk projects where risks need to be identified and managed early.

---

2. SDLC Models

6. Explain the Waterfall Model in SDLC.
   The Waterfall Model is the oldest and most straightforward SDLC model. It is a linear, sequential approach where the project flows downwards (like a waterfall) through a series of distinct phases:
   - **Requirement Gathering and Analysis:** All requirements are gathered at the beginning.
   - **System Design:** The system's architecture and design are created.
   - **Implementation:** The code is written.
   - **Testing:** The software is tested.
   - **Deployment:** The software is released.
   - **Maintenance:** Ongoing support and bug fixes.
7. **Example:** Imagine building a simple internal HR tool for a company. The requirements are clearly defined from the start: employees need to log in, view their pay stubs, and update contact information. Since the requirements are unlikely to change, a Waterfall approach can be effective. The team completes requirements, moves to design, then coding, and so on, without going back.
8. **What are the disadvantages of the Waterfall Model?**
   - **Inflexibility:** It's very difficult to go back to a previous phase once it's complete. If a requirement changes midway through development, it can cause significant delays and costs.
   - **Late Testing:** Testing is done only after all development is complete, which means bugs are often discovered late in the cycle when they are more expensive and difficult to fix.
   - **Lack of Customer Involvement:** The customer is involved mainly in the requirements phase. Their feedback is not integrated during development, which can lead to the final product not meeting their actual needs.
   - **High Risk:** The model is not suitable for complex or large projects where requirements are not fully known upfront, as the risk of failure is high.
9. Explain the V-Model (Verification and Validation Model).
   The V-Model is an extension of the Waterfall Model. It's called the "V" model because of its V-shape, which illustrates the relationship between each phase of the development lifecycle and its corresponding testing phase. The left side of the 'V' represents the Verification phases (creating the product), and the right side represents the Validation phases (testing the product).
   - **Verification (Left side):**
     - Requirement Analysis leads to Acceptance Testing.
     - System Design leads to System Testing.
     - Architectural Design leads to Integration Testing.
     - Module Design leads to Unit Testing.
   - **Validation (Right side):**
     - Unit Testing

            - Integration Testing
            - System Testing
            - Acceptance Testing

10. **Example:** In developing an autonomous vehicle's software, every development step has a corresponding testing step. When the requirements for the emergency braking system are finalized, the plan for acceptance testing is also drafted. When the code for a specific module is written, a unit test is also created to verify it works as intended. This ensures every component is thoroughly tested.
11. What is the Incremental Model in SDLC?
    The Incremental Model involves breaking down the project into multiple, independent development cycles called "increments." In each increment, a new version of the software is produced that adds new functionalities to the previous version. The process is a series of mini-waterfalls for each increment.
    Example: An email service is developed incrementally.
    - **Increment 1:** A basic version is released with only "send" and "receive" functions.
    - **Increment 2:** The "drafts" and "trash" folders are added.
    - Increment 3: An address book and search functionality are integrated.
      The customer gets a working product early and can provide feedback for future increments.
12. What is the Spiral Model in SDLC?
    The Spiral Model is a risk-driven model that combines the iterative nature of prototyping with the systematic, controlled aspects of the Waterfall model. It's represented as a spiral, where each loop or revolution of the spiral represents a new phase of the project. Each loop consists of four main activities:
    - **1. Planning:** Determining objectives, alternatives, and constraints.
    - **2. Risk Analysis:** Identifying and mitigating potential risks.
    - **3. Engineering:** Developing and testing a new version of the software.
    - **4. Evaluation:** Reviewing the results and planning the next spiral.
13. **Example:** A complex, AI-powered weather prediction system. The first loop of the spiral might be to develop a simple prototype to see if the core algorithms are feasible. Based on the evaluation and risk analysis of this prototype, the team might decide to proceed with a more robust system in the next spiral, addressing new risks like data-handling capacity and model accuracy.
14. **Difference between Spiral Model and Iterative Model.**
    - **Focus:** The **Spiral Model** is primarily **risk-driven**. Each phase starts with a risk analysis to determine the next step. The **Iterative Model** is **requirement-driven**. Each iteration adds a new set of functionalities.
    - **Flexibility:** The Spiral Model is more flexible and can handle high-risk projects better due to its built-in risk analysis. The Iterative Model is also flexible but doesn't have a formal risk management phase in each iteration.
    - **Application:** Spiral is ideal for large, complex, and high-risk projects. Iterative is great for projects where some requirements are known but can be refined over time.
15. Explain the Big Bang Model in SDLC.
    The Big Bang Model is a non-linear SDLC model where there is very little to no planning. All resources are thrown into development, and the project is built with little to no formal process. This model is generally not recommended for professional

projects.
Example: A small team of two developers wants to create a simple mobile game for a weekend hackathon. They have a general idea and start coding immediately without any formal design, documentation, or testing. The entire project is developed with minimal structure. This can work for very small, experimental projects, but is highly risky for commercial software.

16. What is the Agile Model in SDLC?
    Agile is not a single model but a group of methodologies that embrace a mindset based on the Agile Manifesto. It focuses on:
    - **Individuals and interactions** over processes and tools.
    - **Working software** over comprehensive documentation.
    - **Customer collaboration** over contract negotiation.
    - **Responding to change** over following a plan.
17. Agile breaks down a project into small, manageable chunks called **sprints** (in Scrum) or uses a continuous flow (in Kanban). The goal is to deliver working software frequently and respond quickly to feedback.
18. **What are Scrum and Kanban in Agile?**
    - **Scrum:** A lightweight, iterative framework for managing complex projects. It's based on **sprints** (time-boxed iterations, typically 2-4 weeks). Key elements include:
        - **Roles:** Product Owner, Scrum Master, Development Team.
        - **Events:** Sprint Planning, Daily Scrum (standup), Sprint Review, Sprint Retrospective.
        - **Artifacts:** Product Backlog, Sprint Backlog, Increment.
        - **Example:** A team building a new mobile app works in 2-week sprints. In each sprint, they plan the features to be built, code and test them, and at the end of the sprint, they have a new, functional version of the app.
    - **Kanban:** A method for visualizing work, limiting work in progress (WIP), and maximizing efficiency. It uses a **Kanban board** with columns like "To Do," "In Progress," and "Done." The focus is on a continuous flow of work rather than fixed-time sprints.
        - **Example:** A support team uses a Kanban board to manage incoming bug reports. A new bug report is added to the "To Do" column. When a developer starts working on it, they move the card to "In Progress" and then to "Done" when the fix is released.
19. **Difference between Agile and Waterfall models.**
    - **Approach: Waterfall** is linear and sequential, while **Agile** is iterative and incremental.
    - **Flexibility: Waterfall** is rigid and resistant to change. **Agile** is highly flexible and welcomes change.
    - **Customer Involvement:** In **Waterfall**, the customer is involved at the beginning and end. In **Agile**, the customer is continuously involved through regular feedback loops.
    - **Risk: Waterfall** has a higher risk of project failure because issues are found late. **Agile** mitigates risk by releasing working software frequently and addressing problems early.

- **Deliverables: Waterfall** delivers the entire product at once at the end. **Agile** delivers small, functional parts of the product frequently.

---

3. Requirements & Analysis

16. What is the Requirement Gathering phase in SDLC?
    The Requirement Gathering phase is the initial and one of the most critical steps in the SDLC. It involves collecting and documenting the needs and expectations of all stakeholders to define what the software system must do. This phase ensures that the final product addresses the real problems of the users.
    Example: Before building a social media app, the team would conduct interviews with potential users, send out surveys, and hold workshops with product managers to understand their needs. They would find out that users want to post photos, connect with friends, and send private messages.
17. **Difference between functional and non-functional requirements.**
    - **Functional Requirements:** These describe **what** the software system must **do**. They define specific actions or behaviors of the system.
        - **Example:** "The system shall allow users to log in with an email and password." "The system shall process a credit card payment."
    - **Non-Functional Requirements (NFRs):** These describe **how** the system performs. They define quality attributes and constraints of the system, such as performance, security, usability, and reliability.
        - **Example:** "The login page must load within 2 seconds." "The system must be available 99.9% of the time." "User passwords must be encrypted."
18. What is a BRD (Business Requirement Document)?
    A Business Requirement Document (BRD) is a formal document that describes the business goals and needs of a project from the customer's perspective. It explains the "what" and "why" of the project, focusing on the business problem that the software will solve. It is written in simple, non-technical language so that business stakeholders can understand it.
    Example: A BRD for a new mobile banking app would state the business need to attract a younger demographic, the features required (like mobile check deposit), and the key performance indicators (KPIs) like user adoption rate.
19. What is an SRS (Software Requirement Specification)?
    A Software Requirement Specification (SRS) is a technical document that details the specific functional and non-functional requirements of the software from a technical perspective. It is created by the development team and serves as a blueprint for the design and development phases. It elaborates on the BRD and provides detailed information for developers and testers.
    Example: An SRS for the mobile banking app would specify the technical details: "The mobile check deposit feature will use the device's camera API to capture images." "The app must support Android and iOS versions 12 and above." "The encryption standard for passwords will be AES-256."
20. **Difference between BRD and SRS.**

    - **BRD:** Written by the business team (e.g., product managers). Describes the **business goals** and **user needs** in non-technical terms. Focuses on **what** the business wants to achieve.
    - **SRS:** Written by the technical team (e.g., business analysts, system architects). Describes the **technical requirements** for the software in detail. Focuses on **how** the system will function to meet the business needs.
21. Who are stakeholders in SDLC?
    Stakeholders are anyone with an interest in the outcome of the project. They can be individuals, groups, or organizations. Key stakeholders include:
    - **Customers/End-Users:** The people who will use the software.
    - **Project Managers:** Responsible for overseeing the project.
    - **Developers/Engineers:** The team building the software.
    - **Quality Assurance (QA) Testers:** The team ensuring the quality of the software.
    - **Business Analysts:** The bridge between the business and technical teams.
    - **Senior Management:** Executives who approve and fund the project.
22. What is feasibility analysis in SDLC?
    Feasibility analysis is a study conducted during the requirements phase to determine if a project is viable and practical. It helps to decide whether to proceed with the project or not. It typically assesses feasibility in several areas:
    - **Technical Feasibility:** Do we have the technology and expertise to build it?
    - **Economic Feasibility:** Is the project financially viable? Are the benefits worth the costs?
    - **Operational Feasibility:** Will the new system fit with the organization's existing processes and culture?
    - **Schedule Feasibility:** Can the project be completed within the required timeframe?
23. What is use case modeling?
    Use case modeling is a technique used in the requirements and analysis phase to describe the functionality of a system from the perspective of an end-user. A use case describes a sequence of actions that a system performs to yield an observable result of value to a particular user (actor). UML Use Case diagrams are used to visualize this.
    Example: For a library management system, a use case might be "Borrow Book." The actor is the "Library Patron," and the system's actions would be "Patron presents library card," "System checks patron's status," "System updates book's status to borrowed."
24. **Tools used in requirement gathering and analysis.**
    - **Jira:** For managing requirements, user stories, and tasks.
    - **Confluence:** For creating and sharing documentation (e.g., SRS, BRD).
    - **Microsoft Office Suite (Word, Excel):** For creating documents and tracking requirements.
    - **Miro/Lucidchart:** For creating flowcharts, use case diagrams, and other visual representations.
    - **Google Forms/SurveyMonkey:** For conducting surveys to gather user feedback.
25. What is requirement traceability matrix (RTM)?
    A Requirement Traceability Matrix (RTM) is a document that maps and traces user

requirements with corresponding test cases. It is used to ensure that every requirement is tested and verified. The RTM shows the relationship between a requirement, the design element that implements it, the code that builds it, and the test cases that verify it.
Example: A simple RTM would have columns for: Requirement ID, Requirement Description, Design Module, Test Case ID, Test Case Status. This ensures that a requirement like "The user must be able to log in" is linked to a specific design element and to test cases that verify the login functionality.

---

4. Design Phase

26. What happens in the Design Phase of SDLC?
The Design Phase translates the requirements gathered in the previous phase into a technical blueprint or architecture for the software system. This phase outlines how the system will be built to meet the specified requirements. It includes creating detailed plans for the system's architecture, data models, user interfaces, and component interactions.
Example: After gathering requirements for a new mobile payment app, the design phase would involve designing the database schema (how user data, transaction history, and balances are stored), creating wireframes for the user interface, and defining the APIs needed to connect to banking services.
27. **Difference between High-Level Design (HLD) and Low-Level Design (LLD).**
    - **High-Level Design (HLD):** This is a macro-level design that describes the overall system architecture. It outlines the main components of the system, their functions, and how they interact with each other. It focuses on the **what** and **how** of the system, but at a high, non-detailed level.
        - **Example:** An HLD for a social media platform would show a diagram with boxes for "User Service," "Post Service," and "Analytics Service," showing how they communicate.
    - **Low-Level Design (LLD):** This is a micro-level design that provides a detailed, component-level view of the system. It specifies the detailed design of each component, including class diagrams, module specifications, and algorithms. It provides a blueprint for the developers to start coding.
        - **Example:** An LLD for the "User Service" would include detailed class diagrams, database table schemas, and a list of functions and their parameters for handling user registration and profile updates.
28. What is system architecture in SDLC?
System architecture is the conceptual model that defines the structure, behavior, and more views of a system. It is a high-level representation of the system's components, their relationships, and the principles and guidelines governing its design and evolution over time. It is a critical part of the HLD phase.
Example: Choosing a **microservices architecture** for an e-commerce platform means designing the system as a collection of independent, small services (e.g., a service for payments, a service for user accounts, and a service for product catalog) that communicate via APIs.

29. What are UML diagrams? Why are they used?
    UML (Unified Modeling Language) is a standard, graphical notation used to visualize, specify, construct, and document the artifacts of a software system. UML diagrams provide a common, visual language for developers and stakeholders to understand the system's design.
    Types and Uses:
    - **Use Case Diagram:** Shows system functionality from a user's perspective.
    - **Class Diagram:** Illustrates the static structure of the system, showing classes, their attributes, and relationships.
    - **Sequence Diagram:** Shows the dynamic interaction of objects in a time-ordered sequence.
    - **Activity Diagram:** Models the flow of control within a system.
    - State Machine Diagram: Describes the behavior of an object in response to external events.
      They are used to communicate complex designs effectively, ensuring everyone is on the same page before coding begins.
30. **Tools used for design in SDLC (like Lucidchart, Enterprise Architect).**
    - **Lucidchart/Miro:** Excellent for creating various diagrams like flowcharts, UML diagrams, and wireframes in a collaborative environment.
    - **Enterprise Architect:** A powerful, comprehensive tool for designing and modeling complex software systems using UML and other standards.
    - **Figma/Sketch/Adobe XD:** Tools for designing user interfaces (UI) and user experiences (UX), creating wireframes, and mockups.
    - ** draw.io (now diagrams.net):** A free and open-source tool for creating flowcharts and diagrams.

---

5. Development Phase

31. What happens in the Development Phase of SDLC?
    The Development Phase is where the software is actually built. Developers write the code based on the design documents (HLD and LLD). This phase is also known as the Implementation or Coding phase. It is not just about writing code; it also includes activities like unit testing, code reviews, and building the software.
    Example: For a new feature on a ride-sharing app, the development team would write the code to handle real-time location tracking, integrate with a payment gateway, and build the user interface for booking a ride. They would also write unit tests to ensure that each individual function (e.g., calculating the fare) works correctly.
32. What is coding standard and why is it important?
    A coding standard is a set of guidelines and best practices for programming languages. It specifies a uniform style for writing code, including naming conventions, indentation, commenting, and code organization.
    Importance:
    - **Maintainability:** Consistent code is easier for other developers (or your future self) to read, understand, and maintain.

- **Readability:** It improves the clarity and readability of the code, making it easier to debug and fix errors.
- **Collaboration:** When multiple developers work on the same project, a consistent standard ensures a cohesive codebase.
- **Reduced Errors:** Adhering to standards can help prevent certain types of bugs and improve overall code quality.

33. What are code reviews?
    Code review is a systematic examination of source code. It is a practice where one or more developers check a piece of code written by another developer. The purpose is to identify bugs, improve code quality, and share knowledge among the team. This can be done manually in a meeting or more commonly today using tools like GitHub, GitLab, or Bitbucket.
    Example: Developer A writes a new feature and submits a "pull request." Developer B reviews the code, provides comments, and suggests improvements, such as making a variable name clearer or fixing a logical error. Once the comments are addressed, the code is merged into the main codebase.
34. **Difference between unit testing and integration testing during development.**
    - **Unit Testing:** A testing method where individual components or "units" of a software application are tested in isolation. The goal is to validate that each unit of the code performs as expected. Unit tests are typically written by the developers themselves.
        - **Example:** A developer writes a function to calculate the total price of a shopping cart. A unit test would check if `calculate_total(items)` returns the correct value for different item lists.
    - **Integration Testing:** A type of testing where multiple individual software modules are combined and tested as a group. The purpose is to expose defects in the interfaces and interactions between these integrated modules.
        - **Example:** After the "shopping cart" module and the "payment" module are developed, an integration test would check if the total price calculated by the first module is correctly passed to and processed by the second module.
35. **Best practices in software development during SDLC.**
    - **Version Control:** Use a version control system like **Git** to track changes, collaborate, and manage the codebase.
    - **Continuous Integration (CI):** Automate the process of integrating code changes from multiple developers into a single software repository.
    - **Automated Testing:** Write automated unit, integration, and end-to-end tests to ensure quality and prevent regressions.
    - **Code Reviews:** Implement a formal process for code reviews to catch errors and improve code quality.
    - **Documentation:** Maintain clear and up-to-date documentation for the code, design, and API.
    - **Agile Mindset:** Embrace an agile mindset with frequent feedback loops, and adapt to changing requirements.

6. Testing Phase

36. What is STLC? How is it different from SDLC?
    STLC (Software Testing Life Cycle) is a sequence of activities performed by the QA team to test the software. It is a subset of the SDLC and focuses solely on the quality assurance aspects.
    - **SDLC:** The **macro-level** process for the entire software development, from concept to retirement.
    - **STLC:** The **micro-level** process for the testing phase within the SDLC.
37. The STLC phases are:
    - Requirement Analysis.
    - Test Planning.
    - Test Case Development.
    - Environment Setup.
    - Test Execution.
    - Test Cycle Closure.
38. **What is Verification vs Validation?**
    - **Verification:** The process of ensuring that the software product is built **correctly**. It is a static analysis of the documents, design, and code to confirm they meet the specified requirements. It answers the question: **"Are we building the product right?"**
        - **Example:** Reviewing the SRS document to ensure all requirements are clear and unambiguous.
    - **Validation:** The process of ensuring that the software product is what the user actually wants. It is a dynamic process that involves executing the software to check if it meets the customer's expectations and requirements. It answers the question: **"Are we building the right product?"**
        - **Example:** User Acceptance Testing (UAT) where end-users try out the software to see if it meets their needs.
39. **Types of testing done in SDLC (Unit, Integration, System, UAT).**
    - **Unit Testing:** Tests individual components in isolation.
    - **Integration Testing:** Tests the combined units to ensure they work together.
    - **System Testing:** Tests the complete, integrated software system as a whole to verify that it meets the specified requirements. It evaluates the entire system's compliance with the functional and non-functional requirements.
    - **User Acceptance Testing (UAT):** The final stage of testing where end-users or clients test the system to ensure it meets their business needs and is ready for deployment.
40. **Difference between manual and automated testing.**
    - **Manual Testing:** The process of manually testing the software by a human tester. The tester interacts with the application as a user would, without any automation tools.
        - **Pros:** Good for exploratory testing, usability testing, and for one-off tests.
        - **Cons:** Time-consuming, prone to human error, and not scalable for repetitive tasks.
    - **Automated Testing:** The process of writing scripts and using software tools to test the application automatically.

        - **Pros:** Fast, repeatable, and scalable. Ideal for regression testing and repetitive, data-driven tests.
        - **Cons:** Requires a significant upfront investment in tools and scripting, not ideal for exploratory testing.

41. What is regression testing?
    Regression testing is a type of software testing that verifies that a new change (e.g., a bug fix, new feature, or code modification) has not introduced new bugs or caused existing functionalities to break. It ensures that the software still works correctly after modifications.
    Example: A bug fix is implemented for the "login" feature. Regression testing would involve running a set of tests not only on the login feature but also on other related features (e.g., user profile page, password reset) to ensure they are still working correctly after the change.
42. **What is smoke and sanity testing?**
    - **Smoke Testing:** A quick, high-level test run on a new software build to ensure that the core functionalities are working and that the build is stable enough to proceed with further testing. It's a "pass/fail" test.
        - **Example:** For a new banking app build, a smoke test would check if the app launches, if a user can log in, and if the account balance is displayed. If any of these fail, the build is rejected.
    - **Sanity Testing:** A narrow, focused test on a specific new feature or a bug fix to ensure that the new functionality works as intended. It's a subset of regression testing.
        - **Example:** After a bug fix is released for the "transfer funds" feature, a sanity test would check only that specific functionality to ensure the fix worked and didn't break anything critical in that area.
43. **Difference between white-box and black-box testing.**
    - **Black-Box Testing:** A testing method where the internal workings or code of the application are **not** known to the tester. The tester only interacts with the software's external interface and provides inputs to check the outputs, based on requirements.
        - **Example:** A tester for a calculator app enters "2 + 2" and verifies that the output is "4," without knowing the underlying code that performs the addition.
    - **White-Box Testing:** A testing method where the internal workings, structure, and code of the application **are** known to the tester. The tester uses this knowledge to design test cases that cover all branches of the code and internal logic.
        - **Example:** A developer writes a test case to ensure that a specific `if/else` block in the login function is executed correctly for both valid and invalid passwords.
44. What is performance testing?
    Performance testing is a non-functional testing type that evaluates how a system performs in terms of stability, scalability, and resource utilization under a specific workload. It answers the question: "How well does the system perform under load?"
    Types:
    - **Load Testing:** Simulating expected user load on the system.
    - **Stress Testing:** Pushing the system beyond its limits to see where it breaks.

   - **Spike Testing:** Simulating a sudden increase in users.
45. **Example:** A performance test for an e-commerce website during Black Friday would simulate 10,000 concurrent users to ensure the site doesn't slow down or crash under high traffic.
46. What is acceptance testing?
    Acceptance testing is a formal testing process to determine if a system satisfies the acceptance criteria and is ready for delivery. It is a part of validation. User Acceptance Testing (UAT) is a common form of acceptance testing where the end-user or client formally tests the system.
    Example: A bank's business team would perform UAT on a new mobile banking app, using it to perform real-world tasks like transferring money and paying bills, to verify that it meets their business needs before it is released to the public.
47. **Tools used in software testing (Selenium, JUnit, TestNG, JMeter).**
    - **Selenium:** An open-source tool for automating web browser testing. It allows testers to write scripts to automate browser actions like clicking buttons and filling forms.
    - **JUnit/TestNG:** Frameworks for unit and integration testing in Java. They provide a structure for writing and running automated tests for Java applications.
    - **JMeter:** An open-source tool for performance and load testing of web applications. It can simulate a large number of users to test the server's performance under load.
    - **Postman:** A tool for API testing. It allows testers to send requests to a server and check the responses.

---

7. Deployment & Maintenance

46. What happens during the Deployment Phase in SDLC?
    The Deployment Phase is the process of making the software system available for use by the end-users. It involves a series of coordinated activities to release the software into the production environment.
    Activities:
    - **Release Planning:** Deciding when and how to release the software.
    - **Environment Preparation:** Setting up the production servers and databases.
    - **Installation/Configuration:** Installing the software and configuring it for the live environment.
    - **Data Migration:** Moving data from the old system to the new one.
    - **Sanity Check:** A final check to ensure the deployed application is working correctly.
47. **Example:** Deploying a new mobile app to the Apple App Store and Google Play Store, or deploying a website to a cloud provider like AWS, is a part of the deployment phase.
48. What is continuous integration and continuous deployment (CI/CD)?
    CI/CD is a modern software development practice that automates the building, testing, and deployment of applications. It aims to deliver code changes frequently and reliably.

  - **Continuous Integration (CI):** Developers integrate their code into a shared repository frequently (e.g., several times a day). An automated build and test process runs on every code commit to detect integration errors early.
  - **Continuous Deployment (CD):** After the CI process is successful, the code is automatically deployed to a production environment without manual intervention.

49. **Example:** A developer commits a change to a Git repository. A CI tool (like Jenkins or GitHub Actions) automatically runs unit tests. If all tests pass, the CD tool automatically builds a new version of the application and deploys it to the production server.
50. **Difference between alpha testing and beta testing.**
    - **Alpha Testing:** A type of acceptance testing conducted by a select group of internal employees or testers. It is performed in a controlled environment at the developer's site to identify bugs before the product is released to external users.
      - **Example:** A gaming company releases a new game to its internal employees to play and find bugs before launching the beta version.
    - **Beta Testing:** A type of acceptance testing conducted by a group of external end-users (the "beta testers"). The goal is to get real-world feedback from a wider audience in a real-world environment.
      - **Example:** A company releases a new app to a limited group of external users to gather feedback on usability, functionality, and performance before the official public release.
51. What is the role of DevOps in SDLC?
    DevOps (Development + Operations) is a set of practices that combines software development (Dev) and IT operations (Ops). The goal is to shorten the SDLC and provide continuous delivery with high software quality.
    - **Role:** DevOps breaks down silos between development and operations teams, promoting a culture of collaboration. It uses automation for CI/CD, infrastructure management, monitoring, and logging to make the entire process more efficient and reliable.
52. **Example:** Instead of developers waiting for the operations team to manually set up servers, a DevOps engineer automates the server provisioning process using tools like Docker and Kubernetes.
53. What is software maintenance in SDLC? Types of maintenance (corrective, adaptive, perfective, preventive).
    Software maintenance is the ongoing process of modifying and updating software after its deployment. It's a critical part of the SDLC that ensures the software remains functional and relevant.
    - **Corrective Maintenance:** Fixing bugs and defects found in the live software.
      - **Example:** Releasing a patch to fix a bug in the payment system that was reported by a user.
    - **Adaptive Maintenance:** Modifying the software to adapt to a changing environment (e.g., new operating systems, new hardware, or new legal regulations).
      - **Example:** Updating a mobile app to be compatible with the latest version of iOS.

- **Perfective Maintenance:** Improving the software's performance, usability, or adding new features based on user feedback.
    - **Example:** Optimizing a database query to make a report load faster.
- **Preventive Maintenance:** Making changes to prevent future problems. This involves refactoring code, updating documentation, and improving code quality.
    - **Example:** Rewriting a complex and hard-to-maintain piece of code to make it cleaner and more robust.

---

8. Agile & Modern SDLC

51. What is Agile methodology?
    As mentioned before, Agile is a flexible, iterative approach to software development that prioritizes rapid delivery, continuous improvement, and customer collaboration. It's an alternative to traditional, sequential models like Waterfall.
52. Difference between Agile and SDLC.
    This is a common trick question. Agile is not an alternative to SDLC, but rather a different approach to implementing the SDLC. SDLC is the overarching concept of the software life cycle. Agile is a methodology (a way of doing things) that fits within the SDLC framework. You can follow the SDLC phases using an Agile methodology.
53. What is Scrum in Agile? Explain roles (Scrum Master, Product Owner, Team).
    Scrum is a framework within Agile for developing and sustaining complex products. It's built on sprints.
    - **Scrum Master:** A facilitator and a coach for the team. They ensure the team follows Scrum principles, remove impediments (obstacles), and help the team be as productive as possible. They do not manage people but serve the team.
    - **Product Owner:** The voice of the customer. They are responsible for defining and prioritizing the items in the **Product Backlog** and ensuring the team builds the right product. They maximize the value of the product.
    - **Development Team:** A self-organizing and cross-functional group of professionals (developers, testers, etc.) who do the work of creating a "Done" increment of the product in each sprint.
54. What are sprints in Scrum?
    A sprint is a time-boxed period, typically 1 to 4 weeks long, during which the development team works to complete a set amount of work from the Product Backlog. It is the core of the Scrum methodology. At the end of a sprint, a new, potentially shippable increment of the product is delivered.
55. What is a user story in Agile?
    A user story is an informal, natural language description of a feature from an end-user's perspective. It's a simple, high-level statement of a requirement that focuses on the value it provides. A common format is: As a [user role], I want to [goal] so that [reason/benefit].
    Example: "As a student, I want to be able to submit my homework online so that I don't have to print it out and hand it in."
56. **What is backlog in Agile?**

    - **Product Backlog:** A prioritized list of all the features, functions, requirements, and bug fixes that need to be done on the product. It's a living document maintained by the Product Owner.
    - **Sprint Backlog:** A subset of the Product Backlog that the development team selects to work on during a single sprint. It's a plan for the current sprint.
57. Explain daily standup meetings in Agile.
    The daily standup (or Daily Scrum) is a short, 15-minute meeting for the development team to synchronize activities and create a plan for the next 24 hours. The goal is to inspect progress toward the Sprint Goal and adjust the Sprint Backlog if needed. Each team member answers three questions:
    - What did I do yesterday?
    - What will I do today?
    - Are there any impediments in my way?
58. **Difference between Scrum and Kanban.**
    - **Scrum:** Time-boxed (sprints), uses a **pull system** based on sprint capacity, and has specific roles and ceremonies. It's better for projects with predictable work.
    - **Kanban:** Continuous flow, uses a **WIP limit** to manage throughput, and has no formal roles or time boxes. It's better for projects with unpredictable or continuous work, like a support or maintenance team.
59. What is SAFe (Scaled Agile Framework)?
    SAFe is a framework that helps large organizations apply Agile principles to their entire enterprise. It provides a blueprint for scaling Agile practices across multiple teams and departments, addressing the coordination and alignment challenges of large-scale projects.
60. **How does Agile improve over Waterfall?**
    - **Increased Flexibility:** Agile allows for changing requirements throughout the project, whereas Waterfall is rigid.
    - **Faster Time-to-Market:** Agile delivers working software in short, frequent increments.
    - **Higher Customer Satisfaction:** Customers are involved continuously and can provide feedback, ensuring the final product meets their needs.
    - **Reduced Risk:** Issues are identified and addressed early, preventing major failures at the end of the project.

---

9. Advanced SDLC Concepts

61. What is DevOps lifecycle? How is it different from SDLC?
    The DevOps lifecycle is an iterative and continuous process that focuses on integrating development and operations. It encompasses activities like continuous development, continuous testing, continuous integration, continuous deployment, and continuous monitoring.
    Difference from SDLC:
    - **SDLC** is a sequential process model for building software.
    - **DevOps** is a cultural and technical practice that **automates and accelerates** the SDLC.

    - SDLC phases often have handoffs between teams (e.g., from dev to QA). DevOps aims to remove these silos and make the process seamless.

62. What are KPIs (Key Performance Indicators) in SDLC?
    KPIs are metrics used to measure the performance and success of an SDLC process. They help teams assess their efficiency and product quality.
    - **Examples:**
        - **Lead Time:** Time from a user story being created to it being deployed.
        - **Deployment Frequency:** How often code is deployed to production.
        - **Bug Count:** Number of defects found in production.
        - **Time to Resolution:** Time it takes to fix a bug after it's reported.
        - **Customer Satisfaction Score:** Measured through surveys.
63. **What are common risks in SDLC? How to manage them?**
    - **Common Risks:**
        - Changing or unclear requirements.
        - Inaccurate project estimates (cost/time).
        - Lack of stakeholder buy-in.
        - Technical complexities.
        - Team-related issues (e.g., skill gaps, turnover).
    - **Management:**
        - Use an Agile approach to accommodate changing requirements.
        - Perform a thorough feasibility analysis and use a risk-driven model like Spiral.
        - Engage stakeholders early and often.
        - Conduct regular risk assessments and have a mitigation plan.
64. What is risk-based testing in SDLC?
    Risk-based testing is a testing approach where the testing effort is prioritized based on the level of risk associated with different features or modules of the software. High-risk areas (e.g., the payment system of an e-commerce site) get more thorough testing than low-risk areas (e.g., a "contact us" form).
65. What is change management in SDLC?
    Change management is a structured process for managing all changes to the software system. It ensures that changes are introduced in a controlled, coordinated, and orderly manner. This includes formal requests, impact analysis, approval, and communication.
66. What is version control in SDLC (Git, SVN)?
    Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. It's crucial for collaborative development.
    - **Git:** A distributed version control system. It's the most popular today. Each developer has a full copy of the repository.
    - **SVN (Subversion):** A centralized version control system. All changes are stored on a central server.
67. What is configuration management in SDLC?
    Configuration management is a process that tracks and manages changes to the software's build, libraries, code, documentation, and tools. It ensures that all components of the system are at a consistent state and can be replicated. It is a key part of DevOps.
68. Explain shift-left testing in SDLC.
    Shift-left testing is a practice that moves the testing activities to an earlier stage in the

SDLC. Instead of testing being a separate phase at the end, it is integrated into every phase of development, starting from the requirements phase. The goal is to find bugs early, when they are easier and cheaper to fix.
Example: Testers and developers collaborate to write test cases and acceptance criteria as soon as requirements are defined, not after the code is written.

69. **What are SDLC deliverables at each phase?**
    - **Requirements:** Business Requirement Document (BRD), Software Requirement Specification (SRS).
    - **Design:** High-Level Design (HLD), Low-Level Design (LLD), System Architecture document, UML diagrams.
    - **Development:** Source code, developer documentation.
    - **Testing:** Test plans, test cases, test reports, defect logs.
    - **Deployment:** Deployment plan, user manuals, release notes.
    - **Maintenance:** Patches, updates, new versions.
70. **What are some challenges faced in SDLC?**
    - Scope creep (uncontrolled changes to the project scope).
    - Poor communication among teams.
    - Lack of clear requirements.
    - Inaccurate project timelines and budgets.
    - Resistance to change from stakeholders.
    - Technical debt (sub-optimal code that needs refactoring).

---

10. Real-World & Best Practices

71. **How do you choose the right SDLC model for a project?**
    - **Project Size and Complexity:** For large, complex projects, Agile or Spiral are better. For small, simple projects, Waterfall might work.
    - **Requirements Clarity:** If requirements are well-defined and stable, Waterfall is an option. If they are unclear or likely to change, Agile is the best choice.
    - **Time-to-Market:** If you need to release a product quickly and frequently, Agile is the way to go.
    - **Risk:** For high-risk projects, the Spiral model's risk analysis is invaluable.
    - **Customer Involvement:** If the customer wants to be involved throughout the process, Agile is ideal.
72. **What are best practices in requirement gathering?**
    - Involve all stakeholders from the beginning.
    - Use a variety of techniques (interviews, surveys, workshops) to gather requirements.
    - Document requirements clearly, unambiguously, and in a verifiable way.
    - Prioritize requirements to focus on the most important features first.
    - Use prototypes and mockups to get early feedback.
73. **How do you ensure quality in SDLC?**
    - **Shift-Left Testing:** Start testing early in the cycle.
    - **Automated Testing:** Use automated unit, integration, and regression tests.
    - **Code Reviews:** Implement a peer-review process.

  - **CI/CD:** Use a continuous integration and deployment pipeline to ensure a stable build.
  - **Clear Requirements:** Ensure requirements are well-defined and unambiguous.

74. **What are common mistakes in SDLC projects?**
    - **Skipping Phases:** Not spending enough time on requirements or design.
    - **Poor Communication:** Siloed teams not communicating effectively.
    - **Lack of Version Control:** Not using a proper system to track code changes.
    - **Ignoring Risk:** Not identifying and mitigating risks early.
    - **Underestimating Time:** Being overly optimistic with project timelines.
75. **How to handle changing requirements in SDLC?**
    - **Agile Approach:** The best way is to adopt an Agile methodology where changes are expected and welcomed.
    - **Change Control Process:** For non-Agile models, establish a formal change management process. All changes must be documented, analyzed for impact, approved by stakeholders, and then implemented.
76. How to ensure security in SDLC? (Secure SDLC practices)
    A Secure SDLC integrates security practices into every phase of the SDLC.
    - **Requirements:** Define security requirements (e.g., encryption, authentication).
    - **Design:** Perform a threat model to identify potential vulnerabilities.
    - **Development:** Use secure coding guidelines, static code analysis (SAST) tools.
    - **Testing:** Conduct penetration testing, vulnerability scanning (DAST).
    - **Deployment:** Use secure configurations and continuous monitoring.
77. What is Agile Testing Quadrants?
    The Agile Testing Quadrants are a model used to categorize and organize the types of testing that should be performed in an Agile project.
    - **Quadrant 1 (Internal, Technology Facing):** Unit tests, component tests.
    - **Quadrant 2 (Internal, Business Facing):** Functional tests, examples, story tests.
    - **Quadrant 3 (External, Business Facing):** Exploratory testing, usability testing, user acceptance testing.
    - **Quadrant 4 (External, Technology Facing):** Performance testing, security testing, scalability testing.
78. What is Test-Driven Development (TDD) in SDLC?
    TDD is a software development process where a developer writes an automated test case for a new function or feature before writing the code.
    - **Steps:**
      - Write a failing test.
      - Write the minimum amount of code to make the test pass.
      - Refactor the code to improve its design.
        TDD ensures a high level of code quality and test coverage.
79. What is Behavior-Driven Development (BDD)?
    BDD is a software development approach that focuses on bridging the communication gap between business and technical teams. It uses a common language (e.g., Gherkin) to write test cases that are understandable by everyone.

    - **Format:** Given [a context], When [an event occurs], Then [an outcome should be]
    - **Example:** Given I am on the login page, When I enter my username and password, Then I should be redirected to my dashboard.

80. **Real-world example of SDLC in your previous project.**
    - You can adapt this to your own experience, but here is a general example: "In my previous project, we developed a mobile application for a food delivery service. We followed a Scrum methodology within the overall SDLC.
        - **Requirements:** The Product Owner gathered user stories from stakeholders and customers, such as 'As a user, I want to be able to see a list of restaurants near me so that I can easily choose a place to order from.'
        - **Design:** We held design sessions to create wireframes for the app and a high-level architecture. We decided on a microservices architecture to handle different functions like user accounts, orders, and payments.
        - **Development & Testing:** We worked in 2-week sprints. In each sprint, we pulled user stories from the backlog, wrote the code, and wrote automated unit and integration tests. We also conducted daily standups to discuss our progress.
        - **Deployment:** At the end of each sprint, we had a potentially shippable increment. We used a **CI/CD pipeline** with tools like Jenkins to automatically build and test the code and deploy it to a staging environment for QA testing. After a final review, the new features were deployed to production.
        - **Maintenance:** Once released, we monitored the app for bugs and gathered user feedback for new features, which were then added to the product backlog for future sprints. This iterative process allowed us to deliver value to the users quickly and adapt to their feedback."