1. **List down all the Models of SDLC.**

The Software Development Life Cycle (SDLC) consists of several models that represent different approaches to software development. Here are some of the commonly known SDLC models:

1. Waterfall Model: This model follows a sequential flow, where each phase (requirements, design, implementation, testing, deployment, maintenance) is completed before moving on to the next one.
2. Iterative Model: This model involves the development and delivery of software in small increments or iterations. Each iteration goes through the phases of requirements, design, development, and testing.
3. Spiral Model: The spiral model combines elements of both the waterfall and iterative models. It emphasizes risk analysis and allows for incremental development while incorporating feedback loops.
4. Agile Model: Agile methodologies, such as Scrum and Kanban, focus on iterative and incremental development. They prioritize flexibility, collaboration, and continuous improvement throughout the project.
5. V-Model: The V-model is an extension of the waterfall model, where each development phase has a corresponding testing phase. The testing activities are planned and executed in parallel with the development phases.
6. Rapid Application Development (RAD) Model: This model emphasizes rapid prototyping and iterative development. It involves user feedback and early user involvement to accelerate the development process.
7. Prototype Model: In this model, a working prototype of the software is developed to gather user feedback and refine requirements before proceeding with the full development.
8. Incremental Model: The incremental model involves dividing the project into small increments or modules that are developed and delivered in stages. Each increment builds upon the previous one.
9. DevOps Model: DevOps is a combination of development and operations. It emphasizes collaboration and integration between software development and IT operations teams, aiming for continuous delivery and deployment.
10. Continuous Integration and Continuous Delivery (CI/CD) Model: CI/CD is an approach that focuses on automating the build, integration, testing, and deployment processes to ensure frequent and reliable software releases.
11. **What is SLTC? Also, Explain all stages of STLC.**

**Software Testing Life Cycle (STLC)** is a sequence of specific activities conducted during the testing process to ensure software quality goals are met. STLC involves both verification and validation activities. Contrary to popular belief, Software Testing is not just a single/isolate activity, i.e. testing. It consists of a series of activities carried out methodologically to help certify your software product. STLC stands for Software Testing Life Cycle.

1. Requirement Analysis: In this stage, the testing team analyzes the requirements and specifications of the software to gain a clear understanding of what needs to be tested. They identify testable features and potential risks.
2. Test Planning: Test planning involves creating a detailed test plan that outlines the testing objectives, scope, test strategy, test schedule, resource allocation, and test environment setup requirements. The plan serves as a roadmap for the testing activities.
3. Test Case Development: Test cases are designed in this stage based on the requirements and test plan. Test cases are step-by-step instructions that specify the inputs, expected outputs, and the conditions under which the test should be executed. Test data and test environments are also prepared.
4. Test Environment Setup: The testing team sets up the required test environment, including hardware, software, network configurations, and test tools. This ensures that the testing environment mirrors the production environment as closely as possible.
5. Test Execution: In this stage, the test cases are executed as per the test plan. The software is tested for defects, errors, and deviations from expected behavior. Testers record the results and report any issues encountered during testing.
6. Test Reporting: Test reporting involves documenting and communicating the test results. Testers prepare test reports, defect reports, and other relevant documentation. They provide detailed information on test coverage, test execution status, and any defects found during testing.
7. Defect Tracking and Management: Defects or bugs identified during testing are logged, tracked, and managed in a defect tracking system. The defects are assigned to the development team for fixing. Testers verify the fixes in subsequent testing cycles.
8. Test Closure: The final stage of STLC involves evaluating the test cycle, summarizing the testing activities, and gathering test metrics. Lessons learned and recommendations for future testing efforts are documented. Test closure also includes preparing final test reports and archiving test assets.
9. **As a test lead for a web-based application, your manager has asked you to identify and explain the different risk factors that should be included in the test plan. Can you provide a list of the potential risks and their explanations that you would include in the test plan?**
10. Compatibility Risks: This risk involves the compatibility of the web application with different browsers, operating systems, devices, and screen resolutions. It can lead to inconsistent behavior or rendering issues across various platforms.
11. Performance Risks: Performance risks relate to the application's ability to handle the expected user load and response times. Factors such as high traffic, data volume, and concurrent users can impact the application's performance and cause slowdowns, timeouts, or crashes.
12. Security Risks: Security risks refer to vulnerabilities that could lead to unauthorized access, data breaches, or malicious attacks. It is essential to identify potential security risks, such as weak authentication mechanisms, data leakage, or inadequate encryption, and ensure appropriate testing is conducted to address them.
13. Usability Risks: Usability risks involve the ease of use and user experience of the web application. Factors like confusing navigation, inconsistent interface, poor accessibility, or lack of responsiveness can negatively impact user satisfaction and adoption.
14. Integration Risks: Integration risks occur when the web application interacts with external systems or APIs. These risks include issues with data synchronization, compatibility, communication protocols, or errors during data exchange, which can lead to functional or performance problems.
15. Data Integrity Risks: Data integrity risks involve the accuracy, completeness, and consistency of data in the application. These risks can arise from incorrect data input, data corruption, data loss, or improper data validation and handling.
16. Scalability Risks: Scalability risks pertain to the application's ability to handle an increased workload, user base, or data volume over time. Inadequate scalability can result in performance degradation or system failures when the application experiences growth.
17. Recovery Risks: Recovery risks relate to the application's ability to recover from failures or unexpected events, such as system crashes, power outages, or network failures. It is crucial to test and ensure the application's resilience and data recovery mechanisms.
18. Regulatory and Compliance Risks: Regulatory and compliance risks involve adhering to industry standards, legal requirements, and data protection regulations. Non-compliance can lead to legal issues, financial penalties, or reputational damage.
19. Maintenance Risks: Maintenance risks encompass the challenges associated with maintaining and updating the application. Risks may include compatibility issues with new technologies or software updates, documentation gaps, or inadequate support.
20. **Your TL (Team Lead)has asked you to explain the difference between quality assurance (QA) and quality control (QC) responsibilities. While QC activities aim to identify defects in actual products, your TLis interested in processes that can prevent defects. How would you explain the distinction between QA and QC responsibilities to your boss?**

Quality Assurance (QA) Responsibilities: QA activities focus on preventing defects and ensuring that the development and testing processes are effective, efficient, and adhere to established standards. The main responsibilities of QA include:

1. Process Definition: QA is involved in defining and establishing the processes, methodologies, and best practices to be followed throughout the software development life cycle (SDLC). This includes creating quality standards, guidelines, and templates.
2. Quality Planning: QA plans and strategizes for quality by defining the test approach, test objectives, and the selection of appropriate testing techniques. It involves identifying the scope of testing, allocating resources, and setting quality goals.
3. Standards and Compliance: QA ensures that the development team adheres to quality standards, industry best practices, and regulatory requirements. It involves conducting audits, reviews, and inspections to identify deviations and provide guidance for improvement.
4. Training and Knowledge Sharing: QA provides training and guidance to the development and testing teams on quality processes, tools, and methodologies. It helps in building a shared understanding of quality requirements and fosters continuous improvement.
5. Metrics and Analysis: QA collects and analyzes metrics and data related to quality, defects, and testing progress. This helps in identifying areas of improvement, measuring process efficiency, and making data-driven decisions to enhance quality.

Quality Control (QC) Responsibilities: QC activities, on the other hand, focus on identifying defects or issues in the actual products or deliverables. The main responsibilities of QC include:

1. Defect Identification: QC is responsible for executing various testing techniques, such as functional testing, regression testing, and performance testing, to identify defects, bugs, and inconsistencies in the software application.
2. Test Execution: QC carries out the planned testing activities, executes test cases, and performs thorough verification and validation of the software to ensure that it meets the specified requirements.
3. Defect Reporting and Management: QC tracks, documents, and reports defects found during testing. It involves providing clear and detailed information about the issues, including steps to reproduce them, so that the development team can address and fix the defects.
4. Test Environment Setup: QC ensures the availability and readiness of the test environment, test data, and test tools needed for effective testing. It includes configuring test environments to mirror the production environment as closely as possible.
5. Test Coverage Analysis: QC analyzes the coverage of the testing efforts to ensure that all identified requirements are adequately tested. It involves identifying any gaps in the test coverage and taking necessary actions to mitigate the risks.
6. **Difference between Manual and Automation Testing.**

Manual Testing:

1. Manual testing involves the process of manually executing test cases without the use of automation tools or scripts.
2. Testers interact with the software application as end-users, manually verifying the functionality, usability, and performance of the application.
3. Manual testing requires human intervention and observation to execute test cases, record test results, and identify defects.
4. It is suitable for exploratory testing, ad-hoc testing, and scenarios that require human judgment, creativity, and real-time observations.
5. Manual testing is time-consuming, especially for repetitive or complex test scenarios.
6. It is generally more flexible and adaptable to changes in requirements or user interactions.
7. Manual testing is recommended for initial testing, usability testing, and scenarios where automation may not be cost-effective.

Automation Testing:

1. Automation testing involves using specialized tools, frameworks, and scripts to automate test case execution and results verification.
2. Test scripts are created to simulate user actions, interact with the application, and validate expected results automatically.
3. Automation testing can significantly speed up the testing process, increase test coverage, and improve test accuracy.
4. It is particularly effective for regression testing, performance testing, and repetitive test scenarios.
5. Automation testing reduces human error and provides consistent results.
6. Initial setup and maintenance of automation frameworks and scripts require additional time and effort.
7. Automation testing is suitable for long-term projects with frequent releases and stable requirements.
8. It is not ideal for scenarios that require human intuition, exploratory testing, or visual verifications.