SDLC Assignment Questions

Ans1:

The Software Development Life Cycle, or SDLC, is a process used by software developers and teams to design, develop, test, and maintain software systems. It provides a structured way of building software and ensures that every part of the process is well planned and executed. The main goal of SDLC is to produce high-quality software that meets or exceeds customer expectations and is completed within time and cost limits.

SDLC is important because it helps avoid chaos in software projects. Without a clear process, developers might miss requirements, make design mistakes, or deliver buggy software. SDLC gives a step-by-step approach so that everyone in the team knows what to do at each stage. It improves communication among developers, testers, project managers, and clients. It also helps reduce errors, manage time better, and keep costs under control.

By following SDLC, companies can deliver software in a more predictable and efficient way. It reduces risks of failure and increases the chances of customer satisfaction. Overall, SDLC provides a foundation that helps turn an idea into a fully working software product.

Ans2:

SDLC has several phases, and each one has its own purpose and value. Together, these phases ensure that the software is developed in a systematic and reliable way.

1. **Requirement Gathering** – In this phase, the team talks to clients or stakeholders to understand what the software should do. This step is important because if the requirements are not clear, the entire project can go in the wrong direction.

2. **Design** – Here, the team plans how the software will work. High-level design focuses on system architecture, while low-level design looks at the detailed logic of modules. This helps avoid technical problems later.

3. **Development (or Coding)** – Developers write code based on the design. This is where the actual software is created.

4. **Testing** – The software is tested to find bugs or issues. Different types of testing like unit, integration, and system testing are done to ensure everything

works properly.

5. **Deployment** – Once the software passes testing, it is installed or delivered to users. This phase includes setting up the software in the real environment.

6. **Maintenance** – After deployment, users may face issues or request changes. The software needs to be updated or fixed regularly. This phase keeps the software useful and up-to-date.

Each of these phases is necessary for building software that works well, is reliable, and is delivered on time.

Ans3:

The Waterfall Model is a step-by-step approach where each phase is completed before the next begins. It's good for projects with clear, unchanging requirements. The Agile Model is flexible and focuses on quick, iterative development. It allows changes even in later stages and involves constant customer feedback. The V-Model is like Waterfall but with testing connected to every development stage. Waterfall suits small, well-defined projects. Agile fits fast-paced environments where requirements change often. V-Model works well for projects that need strict testing, like medical or aerospace software.

Ans4:

Requirement Gathering is the first phase where the team finds out what the client wants the software to do. This includes functional and non-functional needs. Methods used include interviews, questionnaires, group discussions, and observation. Sometimes, prototypes or use cases are created to better understand complex requirements. Clear requirements help avoid confusion and reduce rework later in the project.

Ans5:

The Design phase focuses on planning how the software will be built. High-level design covers the overall structure like architecture, technologies used, and system components. Low-level design breaks things down further into individual modules, data structures, and algorithms. Activities in this phase include creating diagrams, flowcharts, and selecting databases or APIs. A solid design helps developers write clean and efficient code.

Ans6:

In the Coding or Development phase, developers start writing the actual code based on the design documents. They use programming languages, frameworks, and tools

like IDEs, version control (like Git), and debugging tools. Techniques like modular coding and clean code practices are followed to make the code understandable and reusable. This phase turns ideas and plans into working software.

Ans7:

The Testing phase is where the team checks if the software works correctly and meets the requirements. Unit testing checks small pieces of code, integration testing ensures different modules work together, and system testing verifies the entire application. This phase helps catch bugs before the software reaches users and ensures reliability, safety, and performance.

Ans8:

In the Deployment phase, the software is moved from development to the live environment where users can access it. Key things to consider include server setup, data migration, user training, and rollback plans in case of errors. Deployment can be done all at once or in stages. Smooth deployment ensures users can start using the software without problems.

Ans9:

Maintenance happens after the software is deployed. It includes fixing bugs, updating the software, and making improvements based on user feedback. This phase is important because software needs to stay secure, fast, and useful over time. Without regular maintenance, software can become outdated or vulnerable.

Ans10:

The Waterfall Model is a simple, linear approach. Its advantages include clear structure and easy management. But it’s not good at handling changes after development starts. It’s best for small or stable projects where requirements are fully known from the start. Disadvantages include late testing and difficulty adapting to changes.

Ans11:

The Agile Model breaks development into small, repeatable cycles called sprints. It encourages team collaboration, constant feedback, and flexibility. Unlike Waterfall, Agile allows changes even in late stages and focuses on delivering working software quickly. Agile is ideal for fast-changing environments like startups or evolving products.

Ans12:

In a banking app project, I’d start by gathering requirements like user login, money transfer, and transaction history. In the design phase, we’d plan the structure,

security features, and user interface. Developers would then build the features while testers check for bugs. After successful testing, we'd deploy it to the bank's systems and continue maintaining it by adding updates and fixing issues as needed.

Ans13:

For a fitness tracking app, requirement gathering would involve features like step count, workout logging, and goals. In the design phase, we'd create wireframes and plan how the app connects to sensors or health APIs. Developers would code the front-end and back-end, while testers ensure it works on different devices. Finally, we'd deploy it to app stores and keep improving it through updates and user feedback.

Ans14:

In an Agile project, team roles become more flexible and collaborative. Instead of waiting for full requirements, developers and testers work closely with the product owner in short cycles called sprints. The project manager becomes more of a facilitator, helping remove obstacles and keeping the team on track. Tasks are divided into smaller stories, and regular meetings like stand-ups and sprint reviews keep everyone in sync. Agile makes the team more adaptable to changes and encourages faster delivery of features.

Ans15:

In the Waterfall Model, testing happens only after development is complete. Testers follow a test plan that was created early on, and any bugs found can be harder to fix since changes affect the entire structure. In Agile, testing happens continuously, often within the same sprint as development. Testers work closely with developers and provide feedback quickly. This approach catches bugs earlier and supports frequent updates, while Waterfall's method is more rigid and best for stable projects.

Ans16:

During deployment, some challenges include compatibility issues, server errors, or unexpected bugs that didn't appear in testing. To handle these, it's important to test in an environment similar to production and to automate deployment as much as possible. Having a rollback plan and backups helps in case something goes wrong. Also, good communication with users and support teams can make deployment smoother and less risky.

Ans17:

A sample test plan for a simple web app should include the following parts: objectives of testing, scope (what features to test), test environment setup, types of testing (like functional, UI, and security), test cases, expected results, and who is

responsible for what. It should also list tools used, entry and exit criteria, and a schedule. This helps the team stay organized and ensures all parts of the app are properly tested.

Ans18:

To maintain proper documentation in SDLC, I'd ensure that every phase produces a document—like requirement specs, design diagrams, test plans, and deployment steps. Tools like Confluence, Google Docs, or Microsoft SharePoint help manage and share documentation. Version control systems like Git can also track changes in code and docs. Regular reviews and updates keep the documents relevant and accurate throughout the project.

Ans19:

A simple user story for an e-commerce site could be: "As a customer, I want to add items to a shopping cart so that I can purchase multiple items in one order." This fits into Agile as it represents a small, clear feature. The team picks this story in a sprint, discusses how to build it, tests it, and gets feedback. Stories like this help the team focus on delivering usable features one by one.

Ans20:

Test Case: Login Page
Steps:

1. Open the login page
2. Enter a valid username
3. Enter a valid password
4. Click on the login button
   Expected Result:
   User should be redirected to the dashboard
   Pass/Fail Criteria:
   If the user reaches the dashboard after entering correct details, the test passes. If an error shows up or the page doesn't load, it fails. Other test cases would include checking wrong passwords, empty fields, and password masking.

Ans21:

If a major bug is found during testing that needs design changes, I would pause the current development and go back to the design phase. First, I'd gather the team to

analyze the impact, update the design documents, and inform stakeholders. Then the code and tests would be updated accordingly. Following the SDLC ensures that the changes are made in a structured way, avoiding more errors later.

Ans22:

A simple CI/CD pipeline for a web app starts when a developer commits code to GitHub or another repository. The CI server like Jenkins or GitHub Actions runs tests automatically. If tests pass, the code is merged and a build is created. In CD, the new build is deployed to a test or production environment automatically or with approval. This pipeline ensures faster delivery and less manual work.

Ans23:

To write maintainable and scalable code, I follow practices like modular coding, where each part of the code does one specific thing. I add comments to explain logic and use meaningful variable and function names. Using version control systems like Git helps track changes. Following design patterns, writing reusable functions, and keeping code DRY (Don’t Repeat Yourself) also help in making the code easier to manage over time.