Software Development Life Cycle

The **Software Development Life Cycle (SDLC)** refers to the process followed by software developers to design, develop, test, and maintain software applications. It ensures that the software is of high quality, meets the client's requirements, and is delivered on time and within budget. SDLC provides a structured approach to software development and includes various phases.

1. **Planning (Requirement Gathering and Analysis)**

Objective: Understand the project's requirements, scope, and purpose. Establish project goals, timeline, and resources needed.

In this phase, you need to outline the project's requirements and functionality.

- **User Stories**:
    - **Student**: Registers, logs in, views exam scores.
    - **Staff**: Registers, logs in, uploads marks for students.
    - **Admin**: Registers, logs in, manages staff, students, and exam data.
- **Features**:
    - Registration and login for all users.
    - Students can see their scores.
    - Staff can upload scores for students.
    - Admin has the ability to manage users (students and staff) and exam data.
- **Tech Stack**:
    - **Frontend**: HTML, CSS, JavaScript (for dynamic behavior)
    - **Backend**: Node.js, Express
    - **Database**: MySQL

2. **Feasibility Study or Requirements**

Objective: Evaluate the technical, operational, and financial feasibility of the project.

Example:

- **Technical Feasibility**: The technologies like Node.js for the backend, MySQL for the database, and HTML/CSS/JS for frontend are suitable for this type of web application.
- **Operational Feasibility**: Ensure the application will be easy for students, staff, and admin to use.
- **Financial Feasibility**: Estimate the time and resources required to complete this project.

3. **Design (System Design)**

Objective: Design the architecture and components of the system. This phase focuses on the overall system architecture, user interface (UI) design, and database design.

- **System Architecture**:
  - Frontend (HTML, CSS, JS) to handle user interactions and display.
  - Backend (Node.js with Express) to handle API routes and server-side logic.
  - MySQL database to store data related to students, staff, exam scores, and user roles.
- **UI/UX Design**:
  - Create mockups for each page (login, registration, dashboard, etc.)
  - Use a simple layout with login forms, score display, and buttons for uploading data.
- **Database Design**:
  - **users** table: `id, name, email, password, role (student, staff, admin)`
  - **exams** table: `id, exam_name, exam_date`
  - **scores** table: `student_id, exam_id, marks`

4. **Development (Implementation)**

Objective: This is where the actual coding happens. The frontend and backend are developed based on the design created in the previous step.

Example:

- **Frontend (HTML, CSS, JS)**: Write the HTML for pages (login, register, product list), style them using CSS, and add dynamic behavior with JavaScript.
- **Backend (Node.js & Express)**: Implement routes for user login, registration, and fetching product data.
- **Database (MySQL)**: Create the database and tables, and implement SQL queries for retrieving and inserting data.

5. **Testing**

Objective: Verify that the application works as expected and is free from defects.

Example:

- **Unit Testing**: Test individual components or functions (e.g., JavaScript functions, API routes in Node.js).
- **Integration Testing**: Test the interaction between frontend and backend (e.g., submitting a form and receiving a response from the backend).
- **Database Testing**: Ensure that MySQL queries are returning correct data and that data is correctly inserted into the database.
- **User Acceptance Testing (UAT)**: Have users test the application to ensure it meets their needs and expectations.

Example:

- Test if the login functionality works correctly.
- Test if the product data is fetched from MySQL and displayed on the frontend.

6. **Deployment**

Objective: Deploy the application to a live server so it can be accessed by users.

Example:

- **Frontend Deployment**: Upload your HTML, CSS, and JavaScript files to a web server or hosting platform (e.g., Netlify, Vercel).
- **Backend Deployment**: Deploy the Node.js application to a server (e.g., Heroku, AWS).
- **Database Deployment**: Set up a live MySQL database, either on your own server or a cloud service like Amazon RDS or DigitalOcean.

7. **Maintenance**

Objective: Ensure the application continues to run smoothly after deployment. Address bugs, performance issues, or add new features based on user feedback.

Example:

- Monitor the server for any issues (e.g., downtime, performance bottlenecks).
- Regularly update dependencies (e.g., Node.js, libraries).
- Add new features like user profile management, product reviews, or an admin panel.

Waterfall Model in SDLC

![[Pasted image 20250110010049.png]]

What is the Waterfall Model?

The **Waterfall Model** is one of the oldest and simplest approaches to software development. It is a **linear and sequential** methodology where the project flows in one direction—downwards, like a waterfall. In this model, each phase must be completed before the next phase can begin, and there is no going back to a previous phase once it's finished.

The Waterfall Model is often referred to as a **traditional** approach because it is structured, easy to manage, and ideal for small, well-defined projects where requirements are clear from the beginning.

Phases of the Waterfall Model

The Waterfall Model is divided into distinct phases, each having specific deliverables and activities. The main phases of the Waterfall Model are:

1. Requirement Gathering and Analysis

- **Goal**: Define and document all the system requirements.
- This phase involves gathering requirements from stakeholders (clients, users, etc.), analyzing them, and documenting them clearly and thoroughly.
- The output is a **requirements specification** document, which serves as the baseline for the entire project.
- The requirement analysis phase is critical because the design and development will be based on the requirements gathered here.

2. System Design

- **Goal**: Design the system architecture and software specifications.
- The design phase is divided into two sub-phases:
  - **High-Level Design (HLD)**: Describes the overall system architecture and components, focusing on how the system will meet the requirements.
  - **Low-Level Design (LLD)**: Details the design of individual components, including data structures, algorithms, and interface designs.
- In this phase, all decisions regarding system structure, technology, and tools to be used are made.

3. Implementation (Coding)

- **Goal**: Convert the design into actual code.
- Once the design is completed, the actual coding or programming begins. Developers write the source code based on the designs.
- The software is developed in smaller units or modules, and each module is tested individually.
- This phase is often the most time-consuming part of the Waterfall Model because it involves translating the design into working software.

4. Integration and Testing (Verification)

- **Goal**: Test the system to ensure it meets the requirements and is free of defects.
- After the coding phase, the software is integrated, and thorough testing is performed to find defects and ensure that the system works as expected.
- This includes various levels of testing:
  - **Unit Testing**: Tests individual components.
  - **Integration Testing**: Ensures different modules work together.
  - **System Testing**: Validates the entire system's functionality.
  - **Acceptance Testing**: Ensures the system meets the user's requirements.
- Any defects found during testing are fixed, and the system is verified against the requirements.

5. Deployment (Installation)

- **Goal**: Deploy the system to the production environment where it can be used by end-users.
- Once the system is fully tested and defects are resolved, it is deployed to the live environment.
- Users are trained, and documentation is provided to ensure proper use.
- This phase can include several activities like configuration, installation, and user training.

6. Maintenance

- **Goal**: Fix issues, implement updates, and provide ongoing support.
- After the system is deployed, it enters the maintenance phase. Maintenance includes bug fixes, system updates, and user support.
- Over time, changes or new features may be added, but the Waterfall Model is not flexible when it comes to making major changes after the deployment phase.

Characteristics of the Waterfall Model

1. **Linear and Sequential**:
    - The Waterfall Model follows a strict sequence of phases where each phase must be completed before moving to the next. Once a phase is finished, it's difficult to go back to make changes.
2. **Clear Documentation**:
    - Every phase results in clear, detailed documentation. These documents serve as a reference throughout the development process.
3. **Rigidity**:
    - The Waterfall Model assumes that requirements are well-understood and fixed from the beginning of the project. There is minimal room for changes once a phase is completed.
4. **Easy to Manage**:
    - Because the model follows a structured approach with well-defined phases, it is easy to manage and track progress, especially in smaller projects where requirements are unlikely to change.
5. **Limited User Involvement**:
    - In Waterfall, user feedback typically occurs only at the beginning (during requirement gathering) and the end (during acceptance testing). As a result, users may not be involved much during the development phase, which can lead to discrepancies between the final product and user expectations.

Advantages of the Waterfall Model

1. **Simple and Easy to Understand**:
    - The Waterfall Model's clear, step-by-step approach makes it easy to understand and follow.
2. **Structured Approach**:
    - It has well-defined phases, making project management easier and providing a clear roadmap for developers.
3. **Documentation**:
    - Since detailed documentation is created at each phase, it's easier to maintain and update the system in the future.
4. **Clear Milestones**:
    - Each phase has specific deliverables, so it's easier to track progress and ensure that requirements are met.
5. **Best for Small Projects**:
    - For smaller, well-defined projects with little expected change in requirements, Waterfall is a good fit because it focuses on clarity and predictability.

Disadvantages of the Waterfall Model

1. **Inflexibility**:
    - Once a phase is completed, it is difficult to go back and make changes. If requirements change after the initial stages, it can be costly and time-consuming to adapt the project.
2. **Assumes Fixed Requirements**:
    - Waterfall assumes that all requirements are understood upfront and remain fixed. If the requirements evolve during development, the model becomes inefficient.
3. **Late Testing**:
    - Testing happens after the development is complete, which means defects may not be discovered until the end. This can lead to expensive rework if problems are found later.
4. **Not Ideal for Complex Projects**:
    - For larger or more complex projects, Waterfall may not be ideal because requirements are likely to change as development progresses.
5. **User Feedback is Limited**:
    - Users are involved primarily at the beginning and end of the project. This can result in misunderstandings or misalignment with user needs, as there's limited opportunity for feedback throughout the development process.

When to Use the Waterfall Model

While the Waterfall Model has limitations, it works best in situations where:

1. **Requirements Are Well-Understood**:
    - If the project has clear, stable, and fixed requirements from the start, the Waterfall Model can be effective because it focuses on detailed planning and documentation.
2. **Small-Scale Projects**:
    - For small, straightforward projects with minimal complexity, Waterfall provides a structured and simple approach to development.
3. **Regulatory or Compliance Requirements**:
    - In industries where strict documentation and adherence to a set process are required (e.g., healthcare, aerospace), the Waterfall Model's emphasis on documentation and well-defined phases is beneficial.
4. **Shorter Development Cycles**:
    - When a project has a clear deadline and limited scope, Waterfall helps ensure that all phases are completed in a structured manner, leading to timely delivery.

Waterfall Model Example: Building a Simple Web Application

Let's say you're developing a **basic student score management system** where:

- **Students** can view their exam scores.
- **Staff** can upload scores.
- **Admins** can manage users.

Here's how the Waterfall Model would apply:

1. **Requirement Gathering**:
    - Define all the system features: user registration, login, score management, etc.
    - Document requirements: How students, staff, and admins interact with the system.
2. **System Design**:
    - Design the system's architecture: database schema (tables for users, scores), interface designs (login page, dashboard).
    - Determine the tech stack: HTML/CSS for the frontend, Node.js for the backend, MySQL for the database.
3. **Implementation**:
    - Write code based on the design: implement user authentication, score upload, and score viewing features.
4. **Testing**:
    - Test the application: verify that students can view their scores, staff can upload scores, and admins can manage users.
    - Perform integration testing to ensure all components work together.
5. **Deployment**:
    - Deploy the application to a server, making it live for users.
6. **Maintenance**:
    - After deployment, address any bugs or updates based on feedback from users.

Agile Methodology

What is Agile?

Agile is a methodology for software development that emphasizes **flexibility**, **collaboration**, **continuous improvement**, and **customer feedback**. Unlike traditional software development methods like the **Waterfall Model**, where each phase is completed sequentially, **Agile** allows for iterative and incremental development, where a project is divided into small, manageable units (called **iterations** or **sprints**), and the software is developed, tested, and delivered in short cycles.

Agile focuses on delivering a **working product** in small, incremental pieces, making it easier to adapt to changes and incorporate user feedback throughout the development process.

Key Principles of Agile

Agile is guided by the Agile Manifesto, which outlines four key values and twelve principles.

The manifesto emphasizes:

- **Individuals and Interactions over Processes and Tools**: Focus on the people working on the project and their interactions rather than the processes they follow or the tools they use.
- **Working Software over Comprehensive Documentation**: Deliver functional software that meets user needs rather than producing extensive documentation.
- **Customer Collaboration over Contract Negotiation**: Work closely with customers to understand their needs and adapt to changes rather than strictly adhering to initial contract terms.
- **Responding to Change over Following a Plan**: Embrace change and be flexible in adapting plans to new information and requirements rather than rigidly sticking to a fixed plan.

![[Pasted image 20250110005501.png]]

Agile Methodology in Practice

Agile development is implemented in **short iterations** or **sprints** (typically lasting 2-4 weeks), with each sprint producing a **deliverable** piece of software. The methodology relies on continuous collaboration, frequent reviews, and adjustments throughout the development process.

The following are some of the popular Agile frameworks that help organize and structure Agile development:

Key Agile Frameworks

1. **Scrum**:
    - **Scrum** is one of the most widely used Agile frameworks. It organizes development into **sprints** (short development cycles), with each sprint typically lasting 2-4 weeks.
    - A **Scrum team** typically consists of:
        - **Product Owner**: Responsible for defining and prioritizing features or requirements.
        - **Scrum Master**: Facilitates the Scrum process, ensures the team follows Scrum practices, and removes obstacles.
        - **Development Team**: A cross-functional group responsible for delivering the product increment.
    - Scrum involves regular events such as:
        - **Daily Standups**: Short meetings for team members to discuss their progress and any obstacles.

        - **Sprint Planning**: A meeting where the team plans the work for the upcoming sprint.
        - **Sprint Review**: A demonstration of the work completed during the sprint.
        - **Sprint Retrospective**: A meeting to reflect on the sprint and improve the process for the next one.

2. **Kanban**:

    - **Kanban** is another Agile methodology that emphasizes continuous delivery without the time-boxed structure of sprints.
    - Work is visualized on a **Kanban board**, and tasks flow through various stages (e.g., To Do, In Progress, Done).
    - The goal is to maintain a continuous flow of work, limit work in progress (WIP), and improve efficiency.
    - Unlike Scrum, Kanban doesn't require regular meetings or defined roles, making it a flexible approach.

3. **Extreme Programming (XP)**:

    - **XP** focuses on engineering practices, with the aim of improving the quality of the software. It encourages:
        - **Pair Programming**: Two developers work together on the same code.
        - **Test-Driven Development (TDD)**: Writing automated tests before writing the code.
        - **Continuous Integration**: Regularly integrating code changes into the codebase to avoid integration problems.
        - **Refactoring**: Continuously improving the code structure without changing its functionality.

Benefits of Agile

1. **Flexibility and Adaptability**:

    - Agile allows for changes to be made even after development has started. Requirements can evolve based on feedback, market changes, or customer needs, making Agile ideal for projects where changes are expected.

2. **Faster Delivery**:

    - By breaking the project into smaller pieces and delivering work in iterative cycles, Agile teams can deliver working software more frequently, allowing clients to get value sooner.

3. **Improved Collaboration**:

    - Agile promotes close collaboration between stakeholders and development teams. Regular communication and feedback loops ensure that the product is aligned with customer needs.

4. **Better Product Quality**:

    - Continuous testing and feedback, along with regular code reviews, result in higher-quality software that is tested throughout the development cycle.

5. **Increased Customer Satisfaction**:
    - Since customers or stakeholders are involved throughout the development process, they can ensure that the product meets their needs, leading to higher satisfaction.
6. **Higher Team Morale**:
    - Agile promotes a culture of autonomy, self-organization, and continuous improvement, which can boost team morale and productivity.

Agile Process Example

Let's consider an example of building a **student exam management system** using Agile principles.

- **Initial Planning**:
    - Define the core features needed for the system: user login, exam score uploading, score viewing, and admin functionalities.
    - Create a product backlog that lists all features and requirements.
- **Sprint 1 (2 days)**:
    - **Goal**: Implement user registration and login.
    - The team plans and delivers user authentication features, such as:
        - User registration form
        - Login page with authentication logic (using email and password).
    - **Daily Standups**: Discuss progress and obstacles.
    - **Sprint Review**: The team presents the working user registration and login to stakeholders for feedback.
    - **Sprint Retrospective**: Reflect on what went well and what could be improved in the next sprint.
- **Sprint 2 (2 days)**:
    - **Goal**: Implement exam score management.
    - The team delivers the functionality for staff to upload scores and students to view their scores.
    - The product owner prioritizes the next feature in the backlog based on feedback and new requirements.
- **Subsequent Sprints**:
    - The team continues iterating on features, such as:
        - Admin functionality for managing users and exams.
        - Security enhancements or performance improvements.
    - Each sprint delivers a potentially shippable product increment.
