SDLC

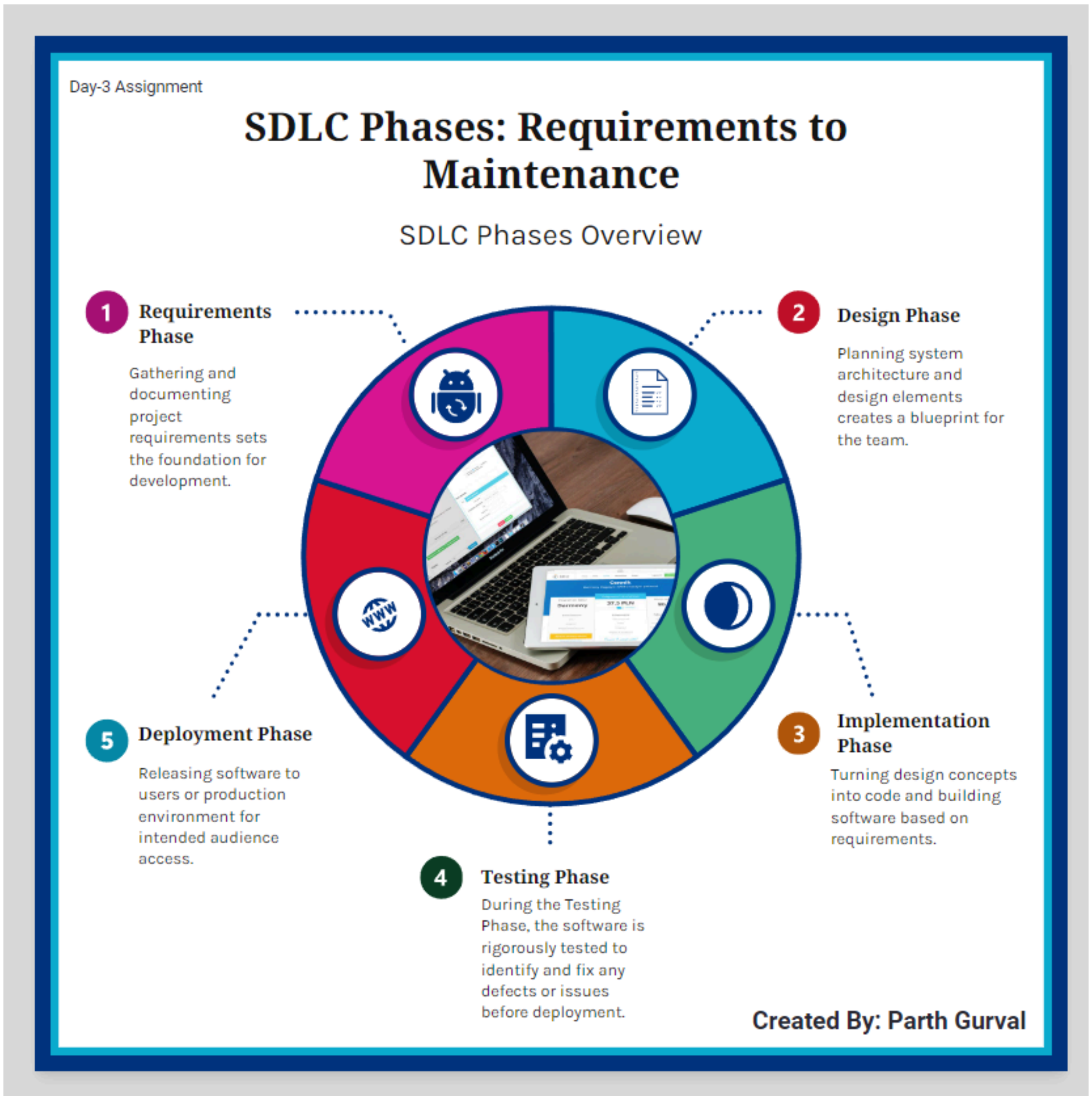
Day-3

Assignments by Parth Gurval, sohamgurval2001@gmail.com

Assignment 1: SDLC Overview - Create a one-page infographic that outlines the SDLC phases (Requirements, Design, Implementation, Testing, and Deployment), highlighting the importance of each phase and how they interconnect.

Ans -->

Infographic Diagram:


Day-3 Assignment
SDLC Phases: Requirements to Maintenance
SDLC Phases Overview
1 Requirements Phase
Gathering and documenting project requirements sets the foundation for development.
2 Design Phase
Planning system architecture and design elements creates a blueprint for the team.
3 Implementation Phase
Turning design concepts into code and building software based on requirements.
4 Testing Phase
During the Testing Phase, the software is rigorously tested to identify and fix any defects or issues before deployment.
5 Deployment Phase
Releasing software to users or production environment for intended audience access.
Created By: Parth Gurval

SDLC Definition:

The Software Development Life Cycle (SDLC) is a structured process used by software development teams to ensure the quality and effectiveness of their software products. It includes several distinct phases, each with specific tasks and goals. Let's break down each phase in simple terms:

Phases of SDLC as Follows:

1. Requirements Phase

2. Design Phase

3. Implementation Phase

4. Testing Phase

5. Deployment Phase

SDLC Phases with Details:

1. Requirements Phase

- **Icon:**
- **Description:** This phase involves collecting and documenting what the users and stakeholders need from the software. It sets the groundwork for the entire project by defining the objectives and scope.
- **Importance:** Ensuring that everyone understands what the software should do is crucial for its success.
- **Interconnection:** The gathered requirements guide the Design phase, ensuring that the architecture aligns with user needs.

2. Design Phase

- **Icon:**
- **Description:** Here, the team creates a blueprint for the software. They plan out the system's architecture, user interfaces, and data flow.
- **Importance:** A well-thought-out design ensures that the software will be structured correctly and meet the project's requirements.
- **Interconnection:** The design serves as a roadmap for the Implementation phase, detailing what needs to be built.

3. Implementation Phase

- **Icon:**
- **Description:** This is the coding phase where developers write the actual software based on the design specifications.
- **Importance:** Translating the design into a working product is essential for creating functional software.
- **Interconnection:** Once the software is built, it moves to the Testing phase to verify that it works as intended.

4. Testing Phase

- **Icon:**
- **Description:** During this phase, the software is rigorously tested to identify and fix any bugs or issues.
- **Importance:** Testing ensures that the software is reliable and meets quality standards before it is released.
- **Interconnection:** After testing, the software is ready for Deployment, assuming all issues are resolved.

5. Deployment Phase

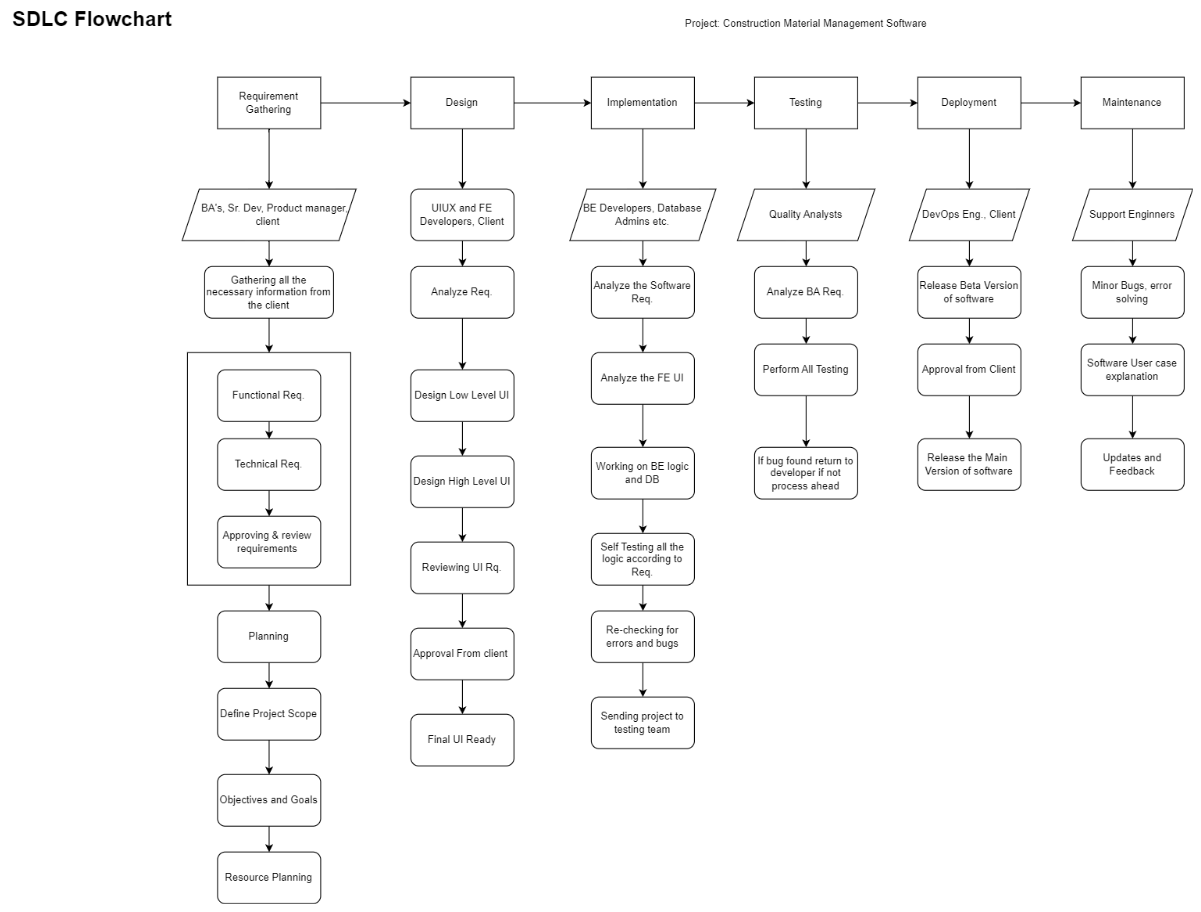
- **Icon:**

- **Description:** The software is released to the users or the production environment where it will be used.

- **Importance:** Making the software available to users is the goal of the development process.

- **Interconnection:** Once deployed, the software enters the Maintenance phase, where it will be supported and updated as needed.

Interconnections:



SDLC Flowchart
Project: Construction Material Management Software
Requirement Gathering
Design
Implementation
Testing
Deployment
Maintenance
BA's, Sr. Dev, Product manager, client
Gathering all the necessary information from the client
Functional Req.
Technical Req.
Approving & review requirements
Planning
Define Project Scope
Objectives and Goals
Resource Planning
UIUX and FE Developers, Client
Analyze Req.
Design Low Level UI
Design High Level UI
Reviewing UI Rq.
Approval From client
Final UI Ready
BE Developers, Database Admins etc.
Analyze the Software Req.
Analyze the FE UI
Working on BE logic and DB
Self Testing all the logic according to Req.
Re-checking for errors and bugs
Sending project to testing team
Quality Analysts
Analyze BA Req.
Perform All Testing
If bug found return to developer if not process ahead
DevOps Eng., Client
Release Beta Version of software
Approval from Client
Release the Main Version of software
Support Enginners
Minor Bugs, error solving
Software User case explanation
Updates and Feedback

Assignment 2: Develop a case study analyzing the implementation of SDLC phases in a real-world engineering project.

Evaluate how Requirement Gathering, Design, Implementation, Testing, Deployment, and Maintenance contribute to project outcomes.

Ans -->

As we have already seen what SDLC is and What are its phases, we are going to start directly with a project introduction and overview

Name of Project: Construction Material Management Software

Project introduction and aim:

- The project involves developing a comprehensive software solution for a construction material management business.
- This system will handle the sale of various construction materials (sand, steel, cement, bricks, etc.) for both small and large-scale projects.
- The stakeholders include owners, managers, workers, and customers. Key features of the application include billing software and live order tracking.

SDLC Phases Overview:

1. Requirements Gathering

Objective: To gather detailed requirements from all stakeholders to ensure the software meets their needs.

Activities:

- Conduct interviews and surveys with owners, managers, workers, and customers.
- Define the functionalities needed, such as material management, billing, order tracking, and user roles.
- Document user stories and use cases to outline interactions with the system.

Outcome:

- A comprehensive requirements document detailing all the features and functionalities.
- A clear understanding of user roles and their specific needs.

Importance: Establishes a solid foundation for the project, ensuring that all stakeholder needs are understood and will be addressed.

2. Design Phase

Objective: To create a blueprint for the software system, detailing the architecture and design.

Activities:

- Develop system architecture diagrams showing how different components will interact.
- Design user interfaces (UI) for owners, managers, workers, and customers.
- Create data models for managing inventory, orders, and billing.

Outcome:

- System architecture diagram.
- Detailed UI.
- Data models and database schema.

Importance: Ensures that the system is well-structured and all components work together seamlessly, facilitating smooth implementation.

3. Implementation Phase

Objective: To build the software based on the design specifications.

Activities:

- Set up the development environment with the necessary tools and technologies.
- Code the core modules: inventory management, billing system, user management, and order tracking.
- Implement the UI for different user roles.
- Integrate a live tracking feature for orders using GPS or similar technologies.

Outcome:

- Functional software with all specified features.
- The initial version is ready for testing.

Importance: Translates design into a working system, turning theoretical plans into practical functionality.

4. Testing Phase

Objective: To verify and validate the software, ensuring it is free of defects and meets the requirements.

Activities:

- Conduct unit tests on individual components.
- Perform integration testing to ensure different modules work together.
- Execute system testing to validate the complete system against requirements.
- Carry out user acceptance testing (UAT) with stakeholders.

Outcome:

- Identified and fixed bugs.
- Validated software ready for deployment.

Importance: Ensures the quality and reliability of the software, minimizing the risk of issues during deployment.

5. Deployment Phase

Objective: To release the software to the production environment for use by the stakeholders.

Activities:

- Prepare the production environment.
- Deploy the software and migrate data.
- Conduct a final round of testing in the live environment.
- Train users on how to use the system.

Outcome:

- Software will be deployed and operational in the production environment.
- Users are trained and ready to use the system.

Importance: Makes the software available to users, allowing them to benefit from its features.

6. Maintenance Phase

Objective: To provide ongoing support and enhancements to the software post-deployment.

Activities:

- Monitor the system for any issues.
- Provide support and resolve any bugs or problems.
- Implement updates and new features based on user feedback.
- Perform regular maintenance to ensure system performance and security.

Outcome:

- Stable and continuously improving software.
- Satisfied users and stakeholders.

Importance: Ensures long-term success and adaptability of the software, keeping it relevant and functional.

Evaluation of Project Outcomes

Requirements Gathering:

- Clearly defined requirements resulted in a system that met all stakeholder needs.
- User stories and use cases provided a clear roadmap for development.

Design:

- Detailed architecture and UI designs facilitated smooth implementation.
- Data models ensured efficient handling of large volumes of data.

Implementation:

- Efficient coding practices led to a robust and functional system.
- Integration of live tracking features added significant value to the users.

Testing:

- Rigorous testing identified and resolved critical issues, ensuring a reliable system.
- User acceptance testing ensured the system was user-friendly and met business needs.

Deployment:

- Successful deployment enabled the business to start using the system with minimal downtime.
- The training ensured users could effectively utilize the new system.

Maintenance:

- Ongoing support and updates kept the system running smoothly and allowed for continuous improvement.
- Regular feedback loops with users helped in identifying areas for enhancement.

Assignment 3: Research and compare SDLC models suitable for engineering projects. Present findings on Waterfall, Agile, Spiral, and V-Model approach, emphasizing their advantages, disadvantages, and applicability in different engineering contexts

Ans -->

- The Software Development Life Cycle (SDLC) is a framework used to plan, design, develop, test, and deliver software projects.

- There are several SDLC models, each with its strengths and weaknesses, suitable for different engineering projects.

- In this Documentation compares and contrasts four popular SDLC models: Waterfall, Agile, Spiral, and V-Model, highlighting their advantages, disadvantages, and applicability in different engineering contexts.

As I have already explained what is SDLC and its Phases in asg1 here I have detailed information about SDLC's different Models

1. Waterfall Model

Description: The Waterfall model is a linear sequential approach where each phase must be completed before the next begins. It follows a strict order: Requirements, Design, Implementation, Testing, Deployment, and Maintenance.

Advantages:

- **Simplicity and Ease of Use:** Easy to understand and manage due to its linear structure.
- **Clear Documentation:** Well-documented phases and deliverables at each stage.
- **Defined Stages:** Each phase has specific deliverables, making project progress easy to track.

Disadvantages:

- **Inflexibility:** Changes are difficult and costly to implement once a phase is completed.

- **Late Testing:** Testing is done late in the process, leading to the risk of significant issues being discovered late.
- **Not Ideal for Complex Projects:** Not suitable for projects with unclear or evolving requirements.

Applicability:

- Best suited for projects with well-defined requirements and where changes are unlikely.
- Ideal for smaller projects with clear, stable objectives.

Best For: Small-scale engineering projects with stable requirements, such as straightforward construction projects or systems with fixed specifications.

2. Agile Model

Description: Agile is an iterative and incremental approach where requirements and solutions evolve through collaborative efforts. It focuses on delivering small, functional pieces of software frequently.

Advantages:

- **Flexibility:** Easily accommodates changes in requirements throughout the project.
- **Continuous Feedback:** Regular feedback from stakeholders ensures the project aligns with business needs.
- **Early and Frequent Delivery:** Continuous delivery of functional software components.

Disadvantages:

- **Resource Intensive:** Requires constant involvement from stakeholders and team members.
- **Lack of Documentation:** Emphasis on working software over comprehensive documentation can be challenging.
- **Less Predictable:** Project timelines and costs can be harder to predict.

Applicability:

- Suitable for projects with rapidly changing requirements or where customer feedback is essential.
- Ideal for complex projects that require frequent adjustments and improvements.

- **Best For:** Projects with evolving requirements, such as software development, where customer feedback and iterative improvements are crucial.

3. Spiral Model

Description: The Spiral model combines iterative development with systematic aspects of the Waterfall model. It focuses on risk assessment and management at each iteration.

Advantages:

- **Risk Management:** Focus on identifying and mitigating risks early in the project.
- **Iterative Refinement:** Allows for iterative refinement of requirements and solutions.
- **Flexibility:** Can accommodate changes at any stage through iterative cycles.

Disadvantages:

- **Complexity:** Managing the process can be complex and require specialized skills.
- **High Cost:** Risk analysis and management can be costly and time-consuming.
- **Documentation Intensive:** Requires extensive documentation and planning at each cycle.

Applicability:

- Best suited for large, complex projects with high-risk elements.
- Ideal for projects requiring frequent reassessment and risk management.

Best For: Large-scale, complex engineering projects with significant risk elements, such as aerospace or large infrastructure projects.

4. V-Model (Verification and Validation Model)

Description: The V-Model is an extension of the Waterfall model that emphasizes verification and validation. Each development phase has a corresponding testing phase, forming a V shape.

Advantages:

- **Emphasis on Testing:** Each development stage has a corresponding test phase, ensuring thorough validation.
- **Clear and Structured:** Provides a clear structure with well-defined phases and deliverables.
- **Early Detection of Defects:** Issues can be identified and resolved early in the process.

Disadvantages:

- **Inflexibility:** Similar to the Waterfall model, it is difficult to implement changes once a phase is completed.
- **High Cost:** Detailed testing at each stage can be costly.
- **Not Ideal for Iterative Processes:** Less suited for projects requiring iterative and incremental development.

Applicability:

- Suitable for projects with well-defined, stable requirements.
- Ideal for projects where thorough testing and validation are critical.

Best For: Projects where rigorous testing is essential, such as safety-critical systems in healthcare, automotive, or defense industries.