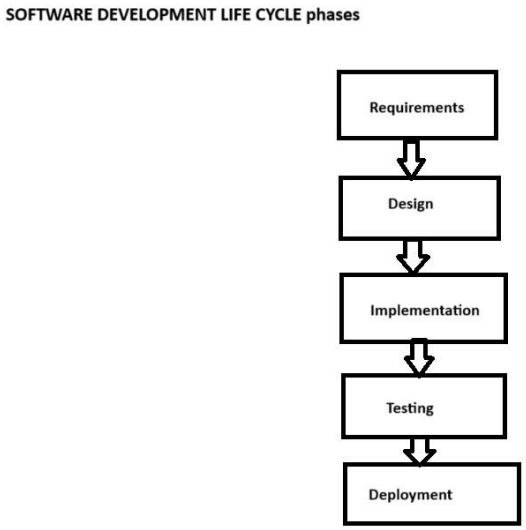
Day-2 04-06-2024

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



Requirements:

- Gathering and documenting what the users need from the system.
- Ensures a clear understanding of what needs to be built, helping to avoid scope creep, and ensuring all stakeholders are aligned.
- Key activities like Stakeholder Interviews, Requirement Analysis, Requirement Documentation, Approval.

Design:

- Creating the architecture of the system and detailed design specifications.
- Provides a blueprint for the development team, ensuring the system meets the requirements and is scalable, maintainable, and efficient.
- Key Activities like System Architecture Design, Database Design, User Interface Design, Detailed Design Documentation.

Implementation:

- Coding and converting the design into an actual software system.
- Actual development where the designs are turned into functional software, crucial for building a working product.
- Key Activities like Writing Code, Integration of Components, Version Control Management.

Testing:

- Verifying that the software works as intended and is free of defects.
- Ensures the quality and functionality of the software, identifying and fixing bugs before deployment.
- Key Activities like Unit Testing, Integration Testing, System Testing, User Acceptance Testing.

Deployment:

- Releasing the software to users and ensuring it is operational in the live environment.
- Making the software available to users, ensuring it performs correctly in the real-world environment.
- Key Activities like Deployment Planning, Release Management, Monitoring and Support, User Training.

Interconnectivity of Phases:

- Each phase is interconnected, with outputs from one phase feeding into the next.
- Requirements drive design decisions, design informs implementation choices, implementation results in testable software, and testing validates adherence to requirements.
- Iterative feedback loops ensure continuous improvement and alignment with stakeholder needs throughout the SDLC.

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.

Imagine a company developing an e-commerce platform. Here's how SDLC phases impact the project:

- **Planning:** The project team defines goals, budget, and timelines. They plan for scalability to handle increasing user traffic during peak seasons.
- **Requirement Gathering:** Stakeholders provide input on features, payment gateways, and security requirements. The SRS captures these details.
- **Design:** Architects create a modular system with separate components for inventory management, order processing, and user authentication.
- **Implementation:** Developers write code for each module, ensuring adherence to coding standards.
- **Testing:** Testers verify that product search, checkout, and payment processing work flawlessly. Any issues are addressed.
- **Deployment:** The e-commerce platform is deployed to production servers, and users can start shopping.

Then we should maintain Regular updates address security vulnerabilities, add new features, and optimize performance.

The implementation of SDLC phases contributing significantly to the project outcomes:

Planning: Clear goals (e.g., fast page load times, mobile responsiveness) drive development efforts.

Requirements: Accurate requirements capture features like product categorization, reviews, and wish lists.

Design: A modular architecture ensures scalability during seasonal sales.

Implementation: Clean code enables features like secure user authentication and seamless checkout.

Testing: Rigorous testing catches issues like broken links or slow loading times.

Deployment: Successful deployment allows users to shop online.

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

1.Waterfall model: The Waterfall model is a sequential approach where each phase flows downward like a waterfall. It progresses linearly through stages: requirements, design, implementation, testing, deployment, and maintenance.

Advantages:

- Well-defined phases ensure clarity and predictability.
- Detailed documentation aids in understanding.
- Ideal when requirements are stable, and changes are minimal.

Disadvantages:

- It is difficult to accommodate changes after a phase is complete.
- Testing occurs late, risking late defect detection.
- Not suitable for dynamic or evolving projects.

2.Agile model: The Agile model emphasizes flexibility, collaboration, and iterative development. It breaks work into short cycles (sprints) and adapts based on feedback.

Advantages:

- Responds well to changing requirements.
- Regular releases provide value to users.
- Close interaction among team members.

Disadvantages:

- Requires skilled team members and active client involvement.
- Minimal formal documentation.
- Frequent changes can lead to scope expansion.

3.Spiral Model: The Spiral model focuses on risk management. It combines elements of the Waterfall and Iterative approaches. Each iteration includes planning, risk analysis, engineering, and evaluation.

Advantages:

- Identifies and mitigates risks early.
- Allows gradual development and refinement.
- Adapts to changing requirements.

Disadvantages:

- Requires experienced project managers.
- Multiple iterations extend the timeline.
- Extensive risk analysis can be resource intensive.

4.V-Model: The V-Model extends the Waterfall model by emphasizing testing at each stage. It aligns testing phases with development phases.

Advantages:

- Rigorous testing ensures quality.
- Clear mapping between requirements and tests.
- Well-structured process.

Disadvantages:

- Limited flexibility for changes.
- Testing occurs after development.
- Requires detailed documentation.