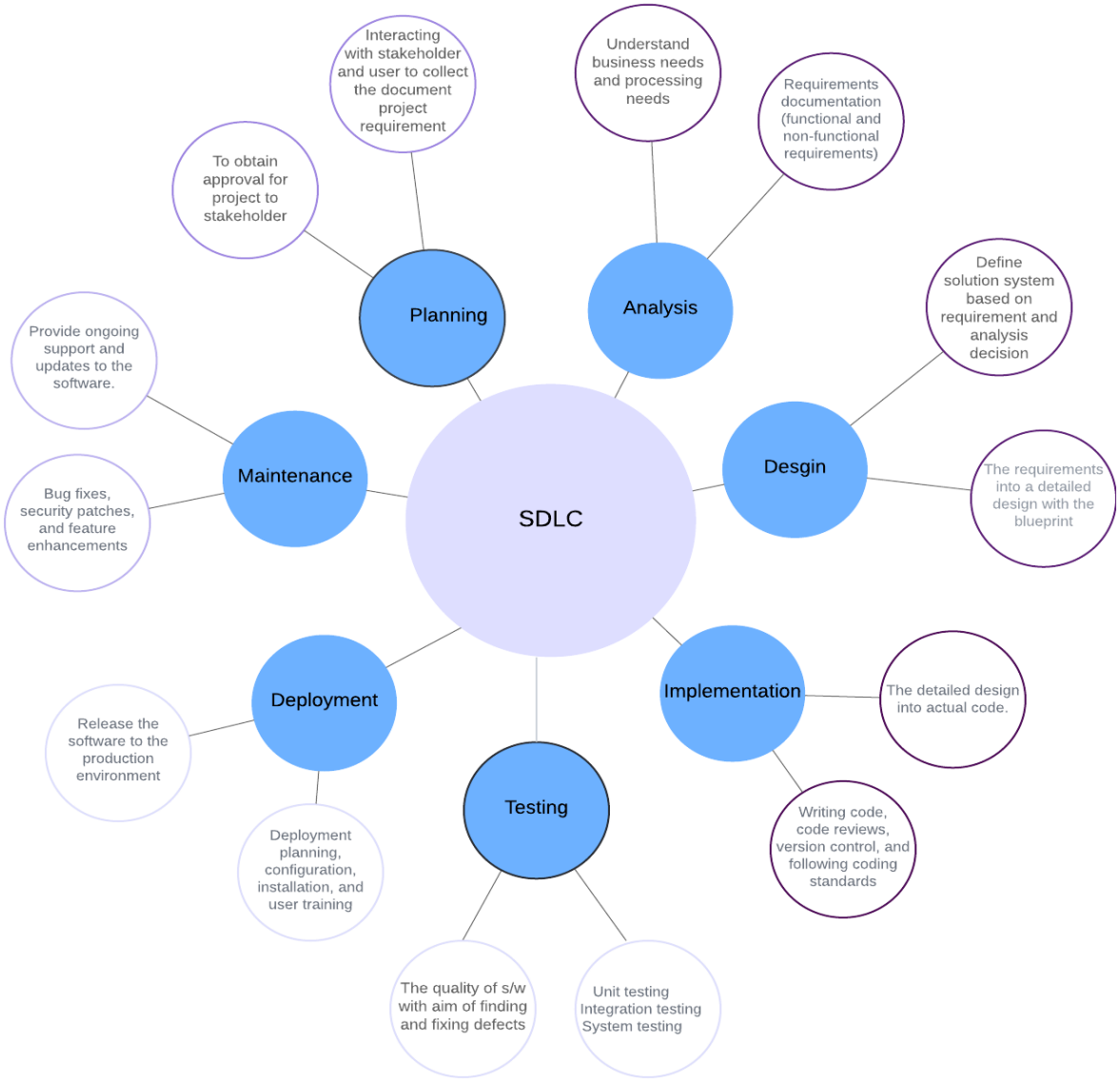
Assignment-1

Assignment 1: SDLC Overview - 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.



- **Planning**:
    - Define the project's scope, objectives, and feasibility.
    - Requirement gathering, feasibility analysis, resource allocation, and project scheduling.
- **Analysis**:
    - Understand and document what the software needs to achieve.
    - Detailed requirement analysis, documentation, and obtaining stakeholder approval.
- **Design**:
    - Architect the system to meet the specified requirements.
    - System design, including hardware and system requirements, detailed design of components, user interfaces, and data structures.
- **Implementation :**
    - Translate design specifications into a functional software application.
    - Writing code, developing software components, and integrating them.
- **Testing**:
    - Ensure the software is bug-free and meets the requirements.
    - Unit testing, integration testing, system testing, and acceptance testing.
- **Deployment**:
    - Make the software available for use.
    - Installation, configuration, and deployment in the production environment.
- **Maintenance**:
    - Provide ongoing support and enhancement.
    - Bug fixing, updates, upgrades, and adding new features as needed.

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.

Case Study: Election Survey System

Introduction :

This case study analyzes the Software Development Life Cycle (SDLC) phases in the context of developing an Election Survey System (ESS). The Election Survey System (ESS) is designed to collect and analyze voter opinions before elections.

SDLC Phases are:

1. **Requirement Gathering**
2. **Design**
3. **Implementation**
4. **Testing**
5. **Deployment**
6. **Maintenance**

1. Requirement Gathering

Objective: To identify and document the functional and non-functional requirements of the ESS.

Process:

1. **Stakeholder Meetings:** Engaged with election commissions, political parties, and potential users to gather requirements.
2. **Surveys and Questionnaires:** Collected data on user expectations and necessary features.
3. **Requirement Analysis:** Prioritized requirements based on feasibility and importance.

2.Design

Objective: To develop a blueprint for the ESS that meets the specified requirements.

Process:

1. **High-Level Design (HLD):** Defined the system architecture, including the database, server-side components, and user interface.
2. **Low-Level Design (LLD):** Detailed the specific functionalities, module designs, and data flow diagrams.

3.Implementation

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

Process:

1. **Module Development:** Coded the individual modules (user management, survey management, analytics, etc.).
2. **Integration:** Combined modules into a cohesive system.
3. **Version Control:** Used Git for source code management to track changes and collaborate efficiently.

4.Testing

Objective: To ensure the ESS is free of defects and meets the specified requirements.

Process:

1. **Unit Testing:** Verified the functionality of individual modules.
2. **Integration Testing:** Ensured that combined modules worked together correctly.
3. **System Testing:** Conducted end-to-end testing of the entire system.
4. **User Acceptance Testing (UAT):** Involved stakeholders in testing to confirm the system met their needs.

5.Deployment

Objective: To make the ESS available to end users.

Process:

1. **Deployment Planning:** Prepared a detailed deployment plan, including rollback procedures.
2. **Environment Setup:** Configured the production environment, ensuring it mirrored the testing environment.

6.Maintenance

Objective: To ensure the ESS remains functional and up-to-date post-deployment.

Process:

1. **Monitoring:** Set up continuous monitoring to detect and address issues promptly.
2. **Bug Fixes and Updates:** Regularly released patches and updates based on user feedback and detected issues.

3. **Performance Optimization:** Made ongoing improvements to enhance system performance and user experience.

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 SDLC Model**:

   The water model is one of the oldest and most straightforward approaches to software development.

   Phases:

   - **Requirements**: Gather requirements from stakeholders and analyze them to understand the project scope and objectives.
   - **Design**: Create a detailed design document outlining software architecture, user interface, and system components.
   - **Development**: Implement the software based on design specifications, including unit testing.
   - **Testing**: Test the software as a whole to ensure it meets requirements and is defect-free.
   - **Deployment**: Deploy the tested and approved software to the production environment.
   - **Maintenance**: Fix issues post-deployment and ensure ongoing compliance with requirements.

   Advantages:

   - **Simplicity**: Linear and sequential nature makes it easy to understand and implement.

   Disadvantages:

   - **Rigidity**: Changes are difficult once a phase is completed.
   - **Limited adaptability**: Not suitable for dynamic or evolving requirements.

   Applicability: Well-suited for stable projects with clear requirements.

2. **Agile SDLC Model**:

   Agile is not a specific methodology but rather a set of principles and values outlined in the Agile Manifesto. The Agile Manifesto prioritizes individuals and interactions, working solutions, customer collaboration, and responding to change over rigid processes and documentation

Phases:

- **Sprints**: Short development cycles with continuous feedback and adjustments.

Advantages:

- **Adaptability**: Easily accommodates changing requirements.
- **Collaboration**: Frequent communication among team members.

Disadvantages:

- **Complexity**: Requires active participation and coordination.
- **Documentation**: Minimal formal documentation.

Applicability: Ideal for dynamic projects where requirements evolve.

3. **Spiral SDLC Model**:

The Spiral model combines the idea of iterative development with the systematic aspects of the Waterfall model. It is based on the concept of a spiral, with each loop representing a phase in the software development process. The model is inherently risk-driven, meaning that risks are continuously assessed and addressed throughout the development life cycle.

Phases:

- **Planning**: Define objectives, constraints, and risks.
- **Engineering**: Develop and test the software.
- **Evaluation**: Review progress and assess risks.
- **Risk Analysis**: Identify and address potential risks.

Advantages:

- **Risk Management**: Explicit focus on risk assessment.
- **Flexibility**: Iterative approach allows for adjustments.

Disadvantages:

- **Complexity**: Requires skilled project management.
- **Time-Consuming**: Multiple iterations can extend the timeline.

Applicability: Suitable for large, complex projects with high risks.

4. **V-Model (Verification and Validation Model)**:

The V-Models, also known as the Verification and Validation models, is an extension of the traditional Waterfall models. It introduces a parallel testing phase for each corresponding development stage, forming a V-shaped diagram. Let's delve into the key principles that underpin the V-Models.

Phases:

- **Requirements**: Define requirements.
- **Design**: Create system design.
- **Coding**: Implement the design.
- **Testing**: Verify and validate at each level.

Advantages:

- **Thorough Testing**: Rigorous testing throughout the process.
- **Traceability**: Clear mapping between requirements and testing.

Disadvantages:

- **Rigidity**: Similar to Waterfall in terms of inflexibility.
- **Documentation Overhead**: Detailed documentation required.

Applicability: Well-suited for critical systems with strict quality requirements.