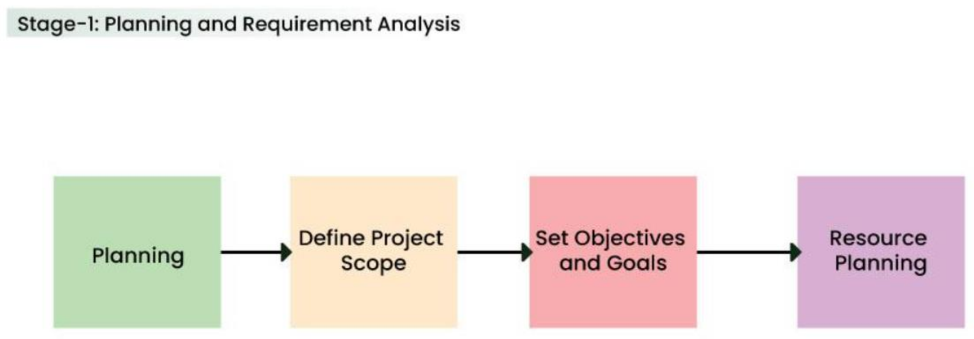
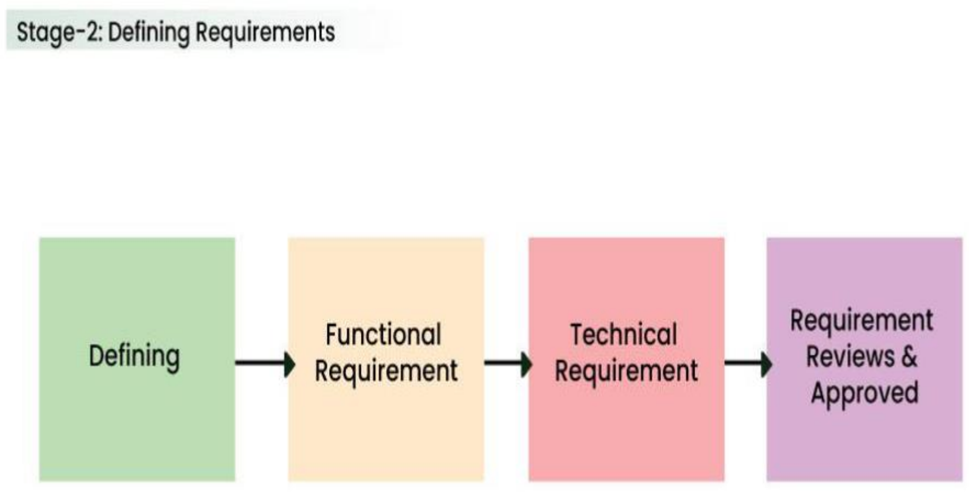
Day 2 Assignment

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.

Software Development Life Cycle (SDLC) Overview

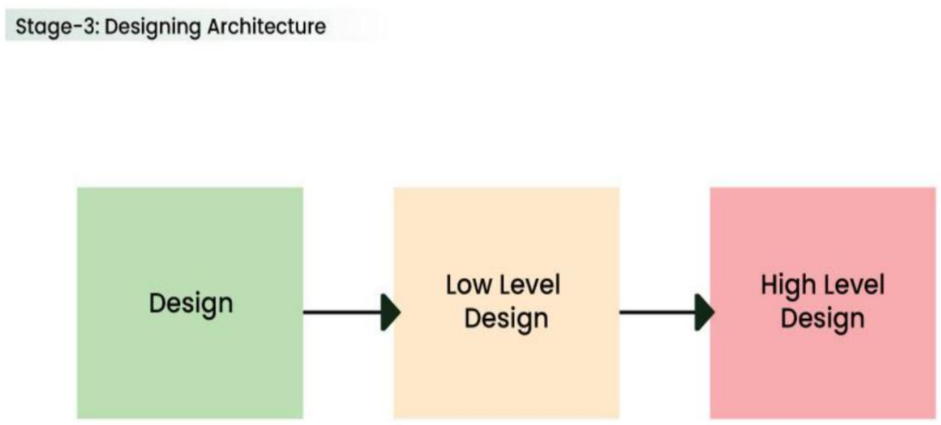
1. **Requirements Phase**:
    - **Importance**: This phase involves gathering and documenting the requirements of the software from stakeholders. It sets the foundation for the entire development process, ensuring that the end product meets user needs.
    - **Interconnection**: Requirements gathered here drive the design and development phases. Clear understanding of user needs reduces rework and ensures alignment with expectations.




Stage-2: Defining Requirements
Defining
Functional Requirement
Technical Requirement
Requirement Reviews & Approved

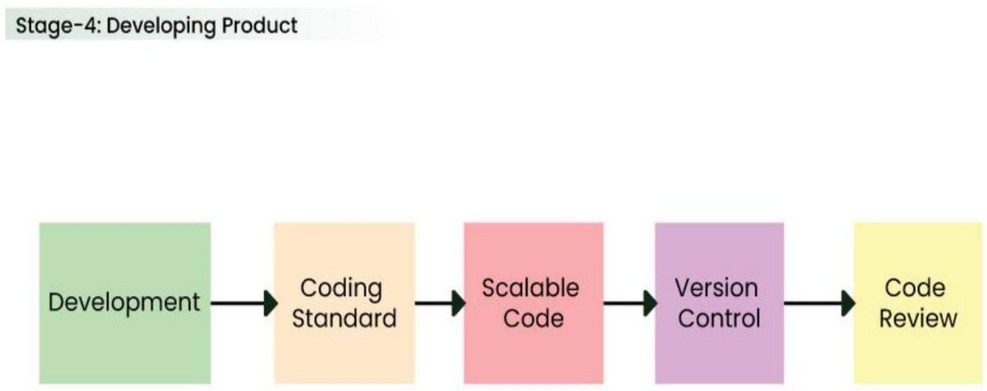
2. **Design Phase**:

- **Importance**: In this phase, system architecture, software design, and user interface design are created based on the requirements gathered earlier. It lays out the blueprint for the development team to follow.
- **Interconnection**: Designs created here provide a roadmap for implementation and testing. They ensure that the software solution is well-structured and scalable.



3. **Implementation Phase**:

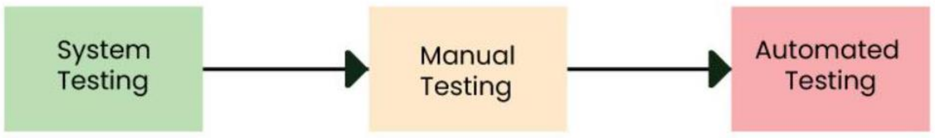
- **Importance**: This is where the actual coding of the software takes place based on the design specifications. Developers write code according to the design, bringing the software to life.
- **Interconnection**: Implementation is directly linked to the design phase. Developers refer to design documents to write code that aligns with the intended functionality and architecture.



4. **Testing Phase**:

   - **Importance**: Testing verifies that the software functions correctly and meets the specified requirements. It ensures quality by identifying and fixing defects before deployment.
   - **Interconnection**: Testing occurs throughout the SDLC but is particularly crucial after implementation. Test cases are derived from requirements and design documents to validate the software against expected outcomes.

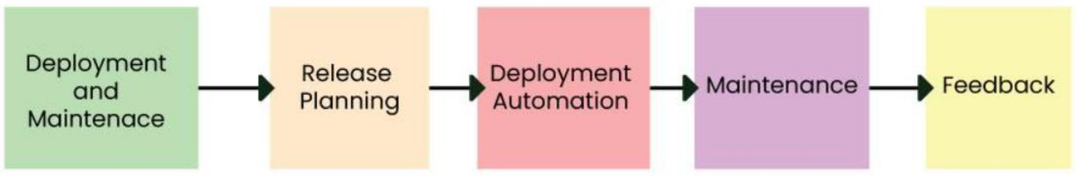
Stage-5: Product Testing and Integration



5. **Deployment Phase**:

   - **Importance**: This phase involves deploying the software to the production environment for end-users. It marks the culmination of the development process and the beginning of the software's operational life.
   - **Interconnection**: Deployment relies on successful completion of testing to ensure that the software is stable and meets quality standards. Feedback from deployment may loop back into requirements for future updates.

Stage 6: Deployment and Maintenance of Products



2. Develop a case study analysing 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: Implementation of SDLC Phases in an Online Banking System

Introduction: In this case study, we'll examine the implementation of Software Development Life Cycle (SDLC) phases in the development of an online banking system. This system aims to provide customers with secure and convenient access to their bank accounts, enabling them to perform various transactions online.

Requirement Gathering Phase:

- **Objective**: Understand the needs and expectations of bank customers and stakeholders.
- **Implementation**: Business analysts conduct interviews, surveys, and workshops with stakeholders to gather requirements. They prioritize features such as account management, fund transfers, bill payments, and security measures.
- **Outcome Contribution**: Clear requirements ensure that the online banking system aligns with customer needs, enhancing user satisfaction and adoption rates.

Design Phase:

- **Objective**: Develop a comprehensive design blueprint for the online banking system.
- **Implementation**: Architects and designers create system architecture, database schemas, user interface designs, and security protocols based on gathered requirements. They focus on usability, scalability, and security.
- **Outcome Contribution**: Well-designed system architecture and user interfaces facilitate efficient development and usage. Security measures protect sensitive customer data, fostering trust and compliance.

Implementation Phase:

- **Objective**: Translate design specifications into functional software components.

- **Implementation**: Developers write code using appropriate programming languages and frameworks. They follow coding standards, version control, and collaborative practices to ensure consistency and maintainability.
- **Outcome Contribution**: Quality code implementation lays the foundation for a stable and reliable online banking system. Adherence to coding standards simplifies maintenance and future enhancements.

Testing Phase:

- **Objective**: Validate the functionality, performance, and security of the online banking system.
- **Implementation**: Testers create and execute test cases, including functional testing, integration testing, performance testing, and security testing. They identify and report defects for resolution.
- **Outcome Contribution**: Thorough testing ensures that the online banking system meets quality standards and regulatory requirements. Early detection and resolution of defects prevent issues in production, enhancing reliability and user experience.

Deployment Phase:

- **Objective**: Release the online banking system to production environment for public access.
- **Implementation**: System administrators deploy the software on servers, configure network settings, and monitor performance during initial rollout. They coordinate with stakeholders to manage user access and provide support.
- **Outcome Contribution**: Successful deployment enables customers to access the online banking system securely and reliably. Monitoring and support ensure smooth operations and prompt resolution of any issues.

Maintenance Phase:

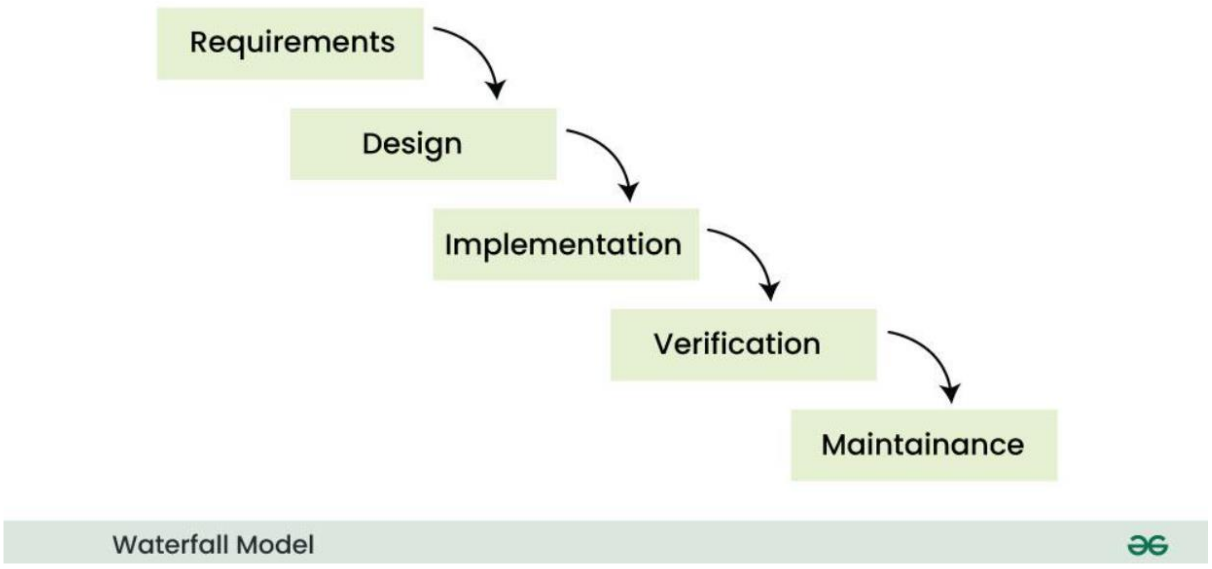
- **Objective**: Sustain the performance, security, and usability of the online banking system over time.
- **Implementation**: Maintenance teams monitor system health, apply software updates, and address user feedback and emerging security threats. They implement enhancements and optimizations to keep the system competitive.

- **Outcome Contribution**: Ongoing maintenance ensures that the online banking system remains functional, secure, and responsive to evolving customer needs and technological advancements.

Conclusion: The implementation of SDLC phases in the development of an online banking system demonstrates their crucial role in achieving project outcomes. Requirement gathering ensures alignment with customer needs, design facilitates efficient development and security, implementation delivers functional software components, testing validates quality and reliability, deployment enables public access, and maintenance sustains performance and relevance over time. By following a structured SDLC approach, organizations can develop and maintain high-quality software systems that meet user expectations and industry standards.

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.
Comparison of SDLC Models for Engineering Projects

1. **Waterfall Model**:

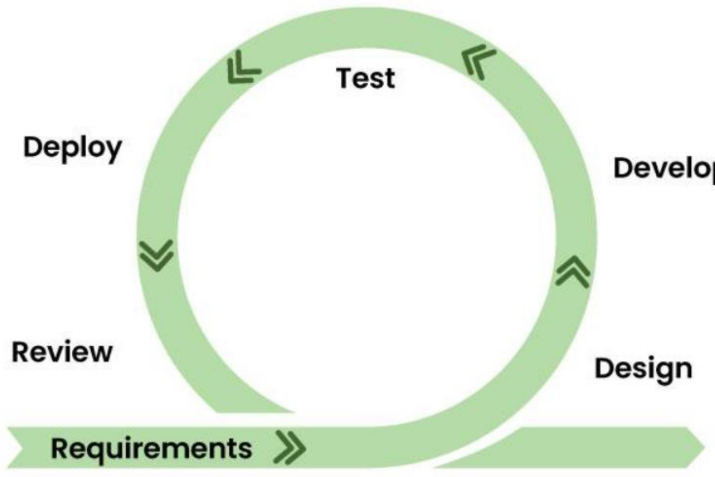


   - **Advantages**:
     - Sequential and linear approach makes it easy to understand and manage.

- Well-defined phases with distinct deliverables facilitate project planning and documentation.
- Suitable for projects with stable requirements and predictable outcomes.

- **Disadvantages**:
    - Limited flexibility for accommodating changes once a phase is completed.
    - High risk of late-stage changes being costly and time-consuming to implement.
    - Limited user involvement until the testing phase, which can lead to misunderstandings or mismatches between requirements and deliverables.
- **Applicability**: Ideal for engineering projects with well-understood requirements and low uncertainty, such as construction projects or hardware development where changes are costly.

2. **Agile Model**:

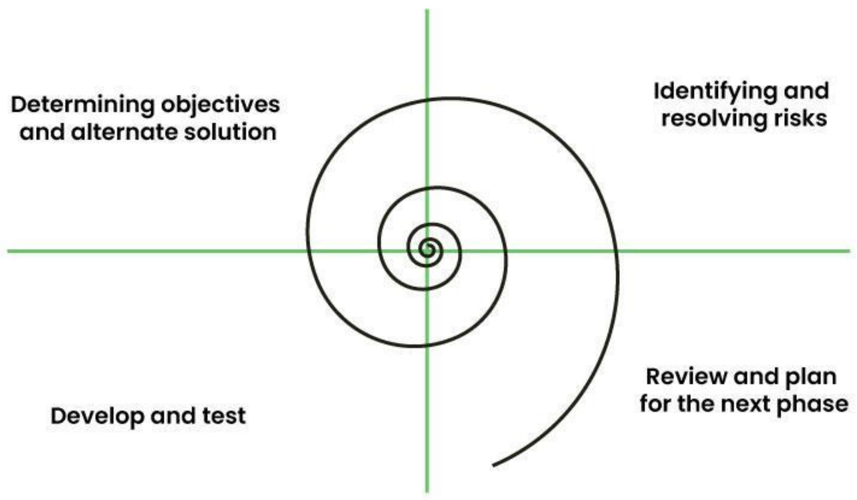


Agile Model

- **Advantages**:
    - Iterative and incremental approach allows for frequent feedback and adaptation.

        - Emphasizes collaboration between cross-functional teams and customer involvement throughout the project.
        - Flexibility to respond to changing requirements and priorities, enhancing adaptability and customer satisfaction.
    - **Disadvantages**:
        - Requires active customer involvement and continuous communication, which can be challenging for large or distributed teams.
        - May lack comprehensive documentation, leading to potential misunderstandings or difficulties in knowledge transfer.
        - Not suitable for projects with strict regulatory compliance requirements or fixed deadlines.
    - **Applicability**: Well-suited for software development projects with evolving requirements, research and development initiatives, and projects where rapid iteration and feedback are critical.

3. **Spiral Model**:


Determining objectives and alternate solution
Identifying and resolving risks
Develop and test
Review and plan for the next phase

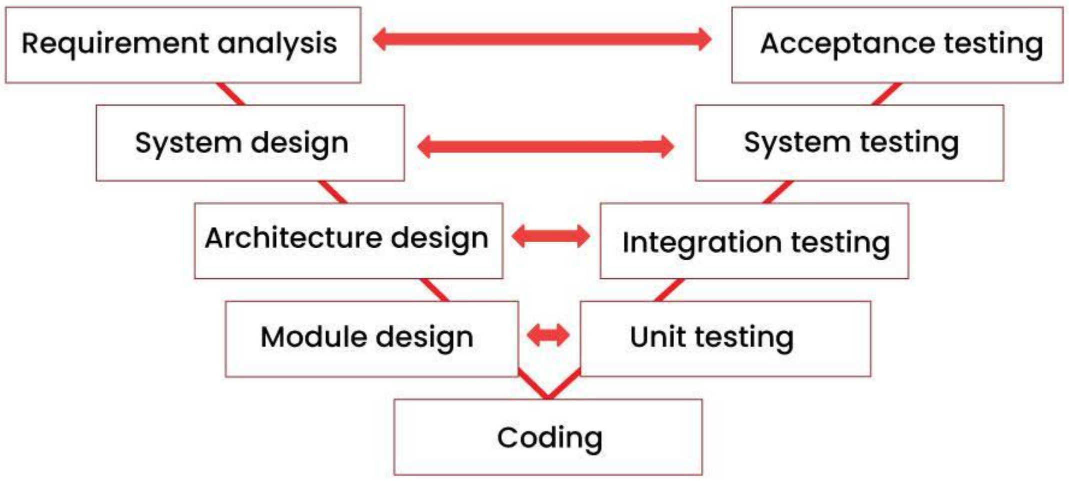
Spiral model

    - **Advantages**:
        - Incorporates risk management throughout the project lifecycle, enabling early identification and mitigation of potential issues.

- Iterative nature allows for progressive refinement of requirements and design based on feedback and experimentation.
- Suitable for large-scale projects with high levels of uncertainty or complexity.

- **Disadvantages**:
    - Requires significant expertise and effort in risk analysis and management, which can increase project overhead.
    - Iterative nature may lead to schedule and cost overruns if not carefully managed.
    - Documentation can be challenging to maintain across multiple iterations.
- **Applicability**: Well-suited for engineering projects with evolving or poorly understood requirements, such as complex software systems, defense projects, or innovative product development.

4. **V-Model**:



V-Model

- **Advantages**:
    - Emphasizes a structured and systematic approach to testing, with testing activities aligned closely with development phases.

    - Provides clarity on testing requirements and expectations, facilitating comprehensive test coverage.
    - Suitable for projects with strict regulatory or quality assurance requirements.
- **Disadvantages**:
    - Sequential nature may lead to delays in testing or feedback loops if issues are identified late in the process.
    - Limited flexibility for accommodating changes once testing activities have begun.
    - May not be suitable for projects with evolving requirements or dynamic environments.
- **Applicability**: Suitable for engineering projects where rigorous testing and verification are critical, such as medical device development, aerospace engineering, or safety-critical systems.

Conclusion: Each SDLC model offers distinct advantages and disadvantages, making them suitable for different engineering contexts based on project requirements, complexity, and uncertainty. Waterfall provides structure and predictability for projects with stable requirements, Agile enables flexibility and responsiveness for evolving projects, Spiral addresses uncertainty and risk through iterative refinement, and V-Model ensures thorough testing and verification for projects with stringent quality requirements. Organizations should carefully evaluate the characteristics of each model and choose the one that best aligns with their project goals and constraints.