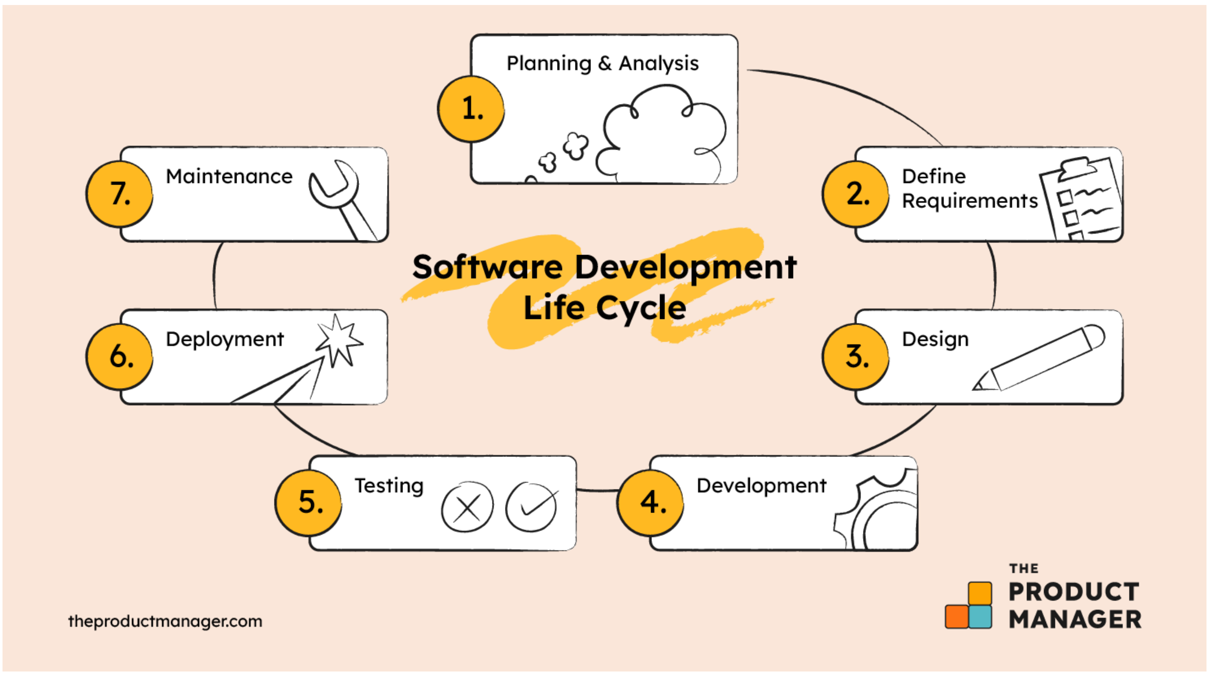
DAY 2 - ASSIGNMENT

Prajwalhonashetti143@gmail.com

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 & Analysis: The first phase of the SDLC is the project planning stage where you are gathering business requirements from your client or stakeholders. This phase is when you evaluate the feasibility of creating the product revenue potential, the cost of production, the needs of the end-users, etc.

To properly decide what to make, what not to make, and what to make first, you can use a feature prioritization framework that takes into account the value of the software/update, the cost, the time it takes to build, and other factors.

Define Requirements: This phase is critical for converting the information gathered during the planning and analysis phase into clear requirements for the development team. This process guides the development of several important documents: a software requirement specification a Use Case document, and a Requirement Traceability Matrix document.

Design: The design phase is where you put pen to paper—so to speak. The original plan and vision are elaborated into a software design document that includes the system design, programming language, templates, platform to use, and application security measures. This is also where you can flowchart how the software responds to user actions. Creating a pre-production version of the product can give the team the opportunity to visualize what the product will look like and make changes.

Development: The actual development phase is where the development team members divide the project into software modules and turn the software requirement into code that makes the product. This SDLC phase can take quite a lot of time and specialized development tools. It's important to have a set timeline and milestones so the software developers understand the expectations and you can keep track of the progress in this stage.

Testing: Before getting the software product out the door to the production environment, it's important to have your quality assurance team perform validation testing to make sure it is functioning properly and does what it's meant to do. The testing process can also help hash out any major user experience issues and security issues. In some cases, software testing can be done in a simulated environment. Other simpler tests can also be automated.

Deployment: During the deployment phase, your final product is delivered to your intended user. You can automate this process and schedule your deployment depending on the type. For example, if you are only deploying a feature update, you can do so with a small number of users If you are creating brand-new software, you can learn more about the different stages of the software release life cycle.

7. Maintenance: The maintenance phase is the final stage of the SDLC if you're following the waterfall structure of the software development process. However, the industry is moving towards a more agile software development approach where maintenance is only a stage for further improvement. In the maintenance stage, users may find bugs and errors that were missed in the earlier testing phase. These bugs need to be fixed for better user experience and retention. In some cases, these can lead to going back to the first step of the software development life cycle.

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: Implementation of SDLC Phases in the Development of a Healthcare Management System

Project Overview:

A healthcare organization embarked on developing a comprehensive Healthcare Management System to streamline patient care, enhance operational efficiency, and ensure regulatory compliance. The project aimed to integrate various modules such as patient registration, electronic health records, appointment scheduling, billing, and reporting functionalities.

1. Requirement Gathering:

The project team engaged with healthcare professionals, administrators, and regulatory experts to gather comprehensive requirements. Key features identified included patient demographics capture, medical history recording, insurance verification, appointment reminders, and secure data exchange capabilities. The requirements gathering phase ensured alignment with regulatory standards such as HIPAA and addressed the specific needs of healthcare providers and patients.

2. Design:
The design phase focused on creating a user-friendly interface and scalable architecture. UX designers developed mockups and prototypes, emphasizing intuitive navigation and accessibility compliance. Software architects designed a modular system architecture, leveraging industry-standard frameworks and ensuring interoperability with existing healthcare systems. Collaboration with stakeholders ensured that the design addressed clinical workflows and compliance requirements.

3. Implementation:
Development teams commenced the implementation phase, following agile methodologies to facilitate iterative development and feedback. Software engineers adhered to coding standards and security best practices, incorporating robust authentication and authorization mechanisms to protect patient data. Integration with third-party systems such as laboratory information systems and picture archiving and communication systems was seamlessly implemented to support comprehensive patient care.

4. Testing:
Testing activities ran parallel to development, encompassing functional, performance, and security testing. QA engineers executed test cases to validate system functionality, ensuring accurate data capture and processing. Load testing was conducted to assess system performance under peak usage scenarios, while penetration testing identified and remediated security vulnerabilities. Comprehensive test coverage and automation facilitated early defect detection and resolution.

5. Deployment:
Upon successful testing, the HMS underwent deployment to production environments. DevOps practices facilitated automated deployment pipelines, ensuring rapid and reliable releases. Continuous monitoring tools were implemented to track system performance, availability, and security posture post-deployment. Deployment strategies included phased rollouts and fallback mechanisms to minimize disruption to healthcare operations.

6. Maintenance:
Following deployment, the project transitioned into the maintenance phase, wherein ongoing support and enhancements were provided. Regular updates addressed bug fixes, performance optimization, and regulatory changes. User feedback and usage analytics informed iterative improvements, ensuring that the HMS remained aligned with evolving healthcare needs and industry standards.

DAY 2 - ASSIGNMENT

Project Outcomes:
The structured implementation of SDLC phases resulted in several positive outcomes:

Enhanced Patient Care: The Healthcare Management System streamlined clinical workflows, enabling healthcare providers to deliver timely and personalized patient care.
Operational Efficiency: Automation of administrative tasks and integration with external systems reduced manual errors and improved operational efficiency.
Regulatory Compliance: Adherence to regulatory standards such as HIPAA ensured patient data privacy and compliance with legal requirements.
Scalability and Reliability: The modular architecture and robust infrastructure supported scalability and reliability, accommodating future growth and expanding healthcare needs.
User Satisfaction: Intuitive design and seamless functionality garnered positive feedback from healthcare professionals, administrators, and patients, enhancing user satisfaction and adoption rates.

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.

Waterfall Model: The Waterfall model is a sequential approach where the development process flows steadily through phases such as requirements, design, implementation, testing, deployment, and maintenance. Each phase relies on the completion of the previous one and is characterized by its emphasis on thorough documentation and upfront planning.

Advantages:

- Simple to use and understand
- Management simplicity thanks to its rigidity: every phase has a defined result and process review
- Development stages go one by one
- Perfect for the small or mid-sized projects where requirements are clear and not equivocal
- Easy to determine the key points in the development cycle

Disadvantages:

- The software is ready only after the last stage is over
- High risks and uncertainty
- Not the best choice for complex and object-oriented projects
- Inappropriate for the long-term projects
- The progress of the stage is hard to measure while it is still in the development

Applicability: Best suited for projects with stable requirements, well-defined scope, and a low risk of change. Commonly used in engineering contexts such as manufacturing or infrastructure projects where predictability and documentation are critical.

Agile Model: Agile is an iterative and flexible approach that prioritizes adaptive responses to changing requirements and customer feedback. It emphasizes collaboration, self-organization, and incremental development, with shorter development cycles known as sprints. Agile promotes delivering working software in short iterations and welcomes changing requirements even late in the development process.

Advantages:

- Corrections of functional requirements are implemented into the development process to provide the competitiveness
- Project is divided by short and transparent iterations
- Risks are minimized thanks to the flexible change process
- Fast release of the first product version

Disadvantages:

- Difficulties with measuring the final cost because of permanent changes
- The team should be highly professional and client-oriented
- New requirements may conflict with the existing architecture
- With all the corrections and changes there is possibility that the project will exceed expected time

Applicability: Well-suited for projects with evolving requirements, high uncertainty, or rapidly changing market conditions. Widely used in software development but can be applied to various engineering contexts such as product development or research projects.

Spiral Model: The Spiral model is a risk-driven approach that combines iterative development with risk management activities. It involves repeated cycles of prototyping, risk analysis, and refinement, allowing for progressive elaboration of requirements and design. Each cycle begins with risk assessment and ends with a prototype or incremental release, enabling early validation and risk mitigation.

Advantages:

- Lifecycle is divided into small parts, and if the risk concentration is higher, the phase can be finished earlier to address the treats
- The development process is precisely documented yet scalable to the changes

- The scalability allows to make changes and add new functionality even at the relatively late stages
- The earlier working prototype is done – sooner users can point out the flaws

Disadvantages:

- Can be quite expensive
- The risk control demands involvement of the highly-skilled professionals
- Can be ineffective for the small projects
- Big number of the intermediate stages requires excessive documentation

Applicability: Suitable for large-scale projects with high complexity and uncertainty, where risk management is critical. Commonly used in engineering contexts such as aerospace, defense, and critical infrastructure projects where iterative refinement and risk mitigation are necessary.

V-Model: The V-Model is a structured approach that emphasizes the correlation between development and testing activities. It follows a phased approach where each development phase is paired with a corresponding testing phase. Requirements and design phases are followed by verification activities, while implementation and testing phases are followed by validation activities. This model ensures comprehensive test coverage and traceability throughout the development lifecycle.

Advantages:

- Every stage of V-shaped model has strict results so it's easy to control
- Testing and verification take place in the early stages
- Good for the small projects, where requirements are static and clear

Disadvantages:

- Lack of the flexibility
- Bad choice for the small projects
- Relatively big risks

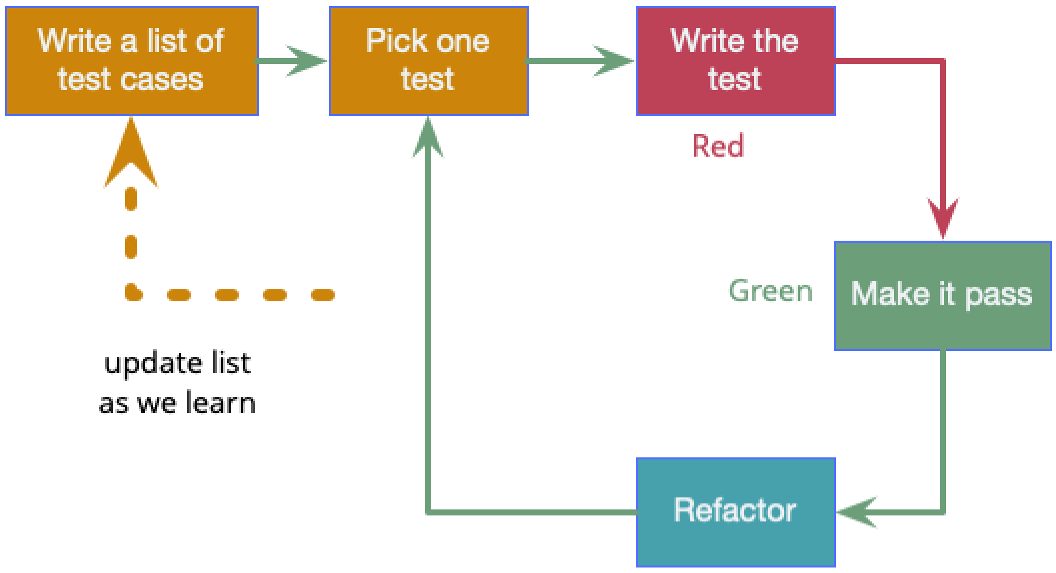
Applicability: Suitable for projects with clear and stable requirements, where comprehensive testing and quality assurance are paramount. Commonly used in engineering contexts such as automotive, medical devices, and regulated industries where compliance and traceability are essential.

DAY 2 - ASSIGNMENT

Assignment 1: Create an infographic illustrating the Test-Driven Development (TDD) process. Highlight steps like writing tests before code, benefits such as bug reduction, and how it fosters software reliability.

TDD is a software development approach where tests are written before the code is implemented. The process involves writing a failing test case, then writing the code to make the test pass, and finally refactoring the code

- Write a test for the next bit of functionality you want to add.
- Write the functional code until the test passes.
- Refactor both new and old code to make it well structured.



Benefits:

• Ensures software is built with testing in mind from the beginning, leading to fewer bugs and higher quality.

• Encourages simple designs and modular code as developers focus on writing code to satisfy specific test cases.

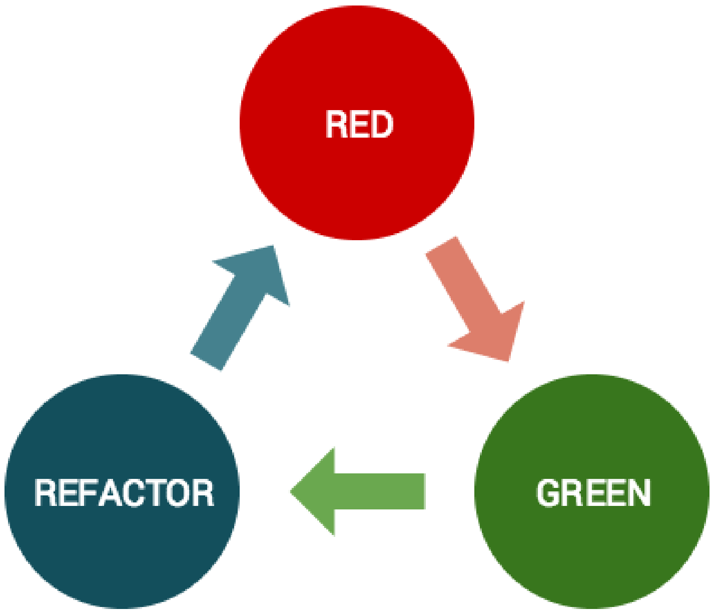
• Provides a safety net for refactoring, allowing developers to make changes confidently knowing that existing functionality is not broken

Red: Write an automated test for a new feature that initially fails because the feature has not yet been implemented. This step ensures that the test is valid, i.e. it fails because the desired behaviour is not implemented, and needs the new functionality to pass.

DAY 2 - ASSIGNMENT

Green: Write the minimum code necessary for the test to pass. This step focuses on making the test pass as quickly as possible, without worrying about code perfection.

Refactor: Once the test is passed, the code is adjusted to improve its structure and quality, while the tests continue to work. This may include removing redundancies, improving readability, and applying software design principles.



Assignment 2: Produce a comparative infographic of TDD, BDD, and FDD methodologies. Illustrate their unique approaches, benefits, and suitability for different software development contexts. Use visuals to enhance understanding.

TDD is a software development approach where tests are written before the code is implemented. The process involves writing a failing test case, then writing the code to make the test pass, and finally refactoring the code.

Benefits:

• Ensures software is built with testing in mind from the beginning, leading to fewer bugs and higher quality.

• Encourages simple designs and modular code as developers focus on writing code to satisfy specific test cases.

• Provides a safety net for refactoring, allowing developers to make changes confidently knowing that existing functionality is not broken.

Process of TDD:

• **Write a failing test:** Developers write a test case that defines the desired behavior or functionality of the code.

• **Write the minimum code to pass:** Developers write the minimum amount of code necessary to make the test pass.

• **Refactor:** Once the test passes, developers refactor the code to improve its design, readability, or performance without changing its external behavior.

BDD is an agile software development methodology that focuses on defining behavior from the perspective of stakeholders using natural language specifications.

Benefits:

• Facilitates communication and collaboration between developers, QA teams, and non-technical stakeholders by using a common language.

• Helps ensure the development process aligns with business goals and user expectations.

• Encourages a test-first approach similar to TDD but with a focus on behavioral specifications rather than low-level unit tests.

BDD process and tools:

• Process:

• Define user stories or features using natural language specifications

• Write executable specifications based on these specifications using tools like Cucumber, SpecFlow, or Behave.

• Implement code to satisfy the specifications, ensuring that each behavior is correctly implemented.

• Run the BDD tests to validate that the implemented code meets the expected behavior.

FDD is an iterative and incremental software development methodology that focuses on delivering tangible, working software features or functionalities in a timely manner.

Principles:

• Develop an overall model of the system.

• Build a feature list based on the model.

• Plan, design, and build features iteratively.

• Inspect and adapt continuously to ensure progress and quality.

FEATURE-DRIVEN DEVELOPMENT (FDD)

• FDD process and phases:

• **Process:**

DAY 2 - ASSIGNMENT

- **Develop an overall model:** Create an initial domain model of the system to understand its structure and relationships.

- **Build a feature list:**

Identify and prioritize features based on user requirements and business priorities.

- **Plan by feature:** Break down features into smaller tasks, estimate effort, and plan iterations to implement them.

- **Design by feature**: Design and implement each feature incrementally, focusing on delivering working functionality.

- **Build by feature:** Develop and test each feature independently before integrating it into the system.

- **Repeat:** Iterate through the process for each feature until all planned features are implemented