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.

   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 (SRS) or product specification, a Use Case document, and a Requirement Traceability Matrix document.

   Design:

   The design phase will include the development of a prototype model. Creating a pre-production version of the product can give the team the opportunity to visualise what the product will look like and make changes without having to go through the hassle of rewriting code.

   Implementation:

   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.

   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.

   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.

   Each phase ensures the software meets user needs and functions correctly, from start to finish.

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.

   Project Overview: A leading bank aimed to enhance its customer experience by developing a mobile banking application. The app would allow customers to perform various banking transactions, including account inquiries, fund transfers, bill payments, and mobile check deposits, conveniently from their smartphones.

1. Requirement Gathering Phase: Bank representatives collaborated with product managers, UX designers, and developers to gather requirements for the mobile banking app. Meetings with stakeholders, surveys, and user interviews helped identify essential features, security requirements, compliance regulations, and user experience preferences.

Outcome: Comprehensive requirements document outlining features, security measures, and user interface specifications.

2. Design Phase: The design team created wireframes and prototypes based on the gathered requirements. They focused on designing an intuitive and user-friendly interface, ensuring seamless navigation and accessibility for users of all demographics. Simultaneously, system architects designed the backend infrastructure, including servers, databases, and APIs, to support the app's functionality and scalability.

Outcome: User-centric design mockups and backend architecture plans that aligned with user needs and technical requirements.

3. Implementation Phase: Development teams commenced coding the mobile banking application, following the design specifications and architectural guidelines. Agile methodologies facilitated iterative development, allowing for frequent releases and feedback loops. Security measures, such as encryption protocols and authentication mechanisms, were implemented to safeguard customer data and financial transactions.

Outcome: Functional mobile banking app developed with robust security features and seamless user experience.

4. Testing Phase: Quality assurance engineers conducted comprehensive testing to ensure the reliability, security, and performance of the mobile banking app. Testing scenarios covered functional testing, security testing, usability testing, and compatibility testing across various devices and operating systems. Automated testing tools accelerated the testing process and enabled regression testing with each code change.

Outcome: Thoroughly tested mobile banking app with minimal defects and optimal performance across different platforms.

5. Deployment Phase: Upon successful testing, the mobile banking app was deployed to app stores for public access. Deployment strategies ensured a smooth rollout, with phased releases to manage server loads and user adoption. User training materials and support channels were provided to assist customers in using the app effectively.

Outcome: Successful deployment of the mobile banking app, enabling customers to access banking services conveniently from their smartphones.

6. Maintenance Phase: The mobile banking app entered the maintenance phase, where ongoing support, updates, and enhancements were provided. Feedback from users and monitoring of app performance informed continuous improvements, such as bug fixes, feature enhancements, and security updates, to ensure the app remained competitive and compliant with industry standards.

Outcome: Sustained operation and improvement of the mobile banking app, meeting evolving customer needs and technological advancements.

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:

Advantages:

- **Simple and easy to understand:** Sequential phases make it easy to plan and manage.
- **Well-suited for stable requirements:** Ideal for projects with clearly defined and stable requirements upfront.
- **Structured approach:** Each phase has specific deliverables and milestones, promoting clarity and accountability.

Disadvantages:

- **Limited flexibility:** Changes are difficult to accommodate once a phase is completed.
- **Long development cycles:** Sequential nature may lead to lengthy development times.
- **Risk of late defect detection:** Testing occurs towards the end, increasing the risk of discovering defects late in the project lifecycle.

Applicability: Waterfall is suitable for projects with well-defined requirements and low uncertainty, such as building physical products, infrastructure projects, or projects where regulatory compliance is critical.

2. Agile Model:

Advantages:

- **Flexibility and adaptability:** Embraces change and allows for iterative development based on user feedback.
- **Early and continuous delivery:** Incremental releases enable stakeholders to see progress and provide feedback early.
- **Enhanced collaboration:** Cross-functional teams collaborate closely, fostering communication and synergy.
- **Focus on customer satisfaction:** Prioritizes delivering value to customers through working software.

Disadvantages:

- **Requires experienced team:** Success relies on the expertise and collaboration of team members.

- **Lack of upfront planning:** Some stakeholders may prefer more detailed upfront planning and documentation.
- **Managing scope changes:** Frequent changes can lead to scope creep if not managed effectively.

Applicability: Agile is suitable for projects with evolving requirements, high uncertainty, or where rapid adaptation to changing market conditions is essential, such as software development, digital product development, or research projects.

3. Spiral Model:

Advantages:

- **Risk management:** Iterative approach allows for early identification and mitigation of risks.
- **Flexibility:** Phases can be tailored to suit the specific needs and risks of the project.
- **Early prototypes:** Each iteration produces a prototype, providing stakeholders with tangible deliverables early in the process.
- **Incorporates feedback:** Stakeholder feedback is incorporated into each iteration, improving the final product.

Disadvantages:

- **Complexity:** More complex than linear models like Waterfall, requiring careful planning and management.
- **Costly:** Iterative nature may lead to increased costs if not managed efficiently.
- **Time-consuming:** Requires thorough risk analysis and multiple iterations, potentially extending the project timeline.

Applicability: Spiral is suitable for projects with high technical complexity, significant risks, or where stakeholder involvement is critical, such as large-scale software development projects, complex engineering projects, or projects with evolving requirements.

4. V-Model:

Advantages:

- **Clarity and structure:** Mirrors the Waterfall model's structured approach, emphasizing verification and validation at each stage.
- **Early focus on testing:** Testing activities are integrated into each phase, promoting early defect detection.
- **Traceability:** Provides clear traceability between requirements and testing activities, ensuring alignment with client needs.
- **Suitable for regulated industries:** Well-suited for projects in industries with stringent quality and compliance requirements.

Disadvantages:

- **Rigid:** Like Waterfall, changes can be challenging to implement once a phase is completed.

- **Limited flexibility:** Less adaptable to changing requirements compared to Agile or Spiral models.
- **Increased documentation:** Requires comprehensive documentation to support traceability, which can be time-consuming.

Applicability: V-Model is suitable for projects with strict regulatory requirements, where thorough documentation, verification, and validation are essential, such as aerospace, automotive, or medical device development projects.

Each SDLC model offers distinct advantages and disadvantages, making them suitable for different engineering contexts based on project requirements, complexity, and stakeholder preferences. Waterfall and V-Model are more traditional and structured, while Agile and Spiral models offer greater flexibility and adaptability to changing requirements and environments. Choosing the most appropriate model depends on factors such as project scope, uncertainty, stakeholder engagement, and regulatory compliance needs.

4. Write an assignment to write a methodology using Test Driven ,Behaviour driven and Feature driven model to create your own chat application ,somewhat similar to whats app.

Project Setup:

- Set up a development environment for the chat application project.
- Choose appropriate programming languages, frameworks, and tools based on your preferences and familiarity.

Test-Driven Development (TDD):

- Implement the chat application using the TDD approach.
- Define unit tests for each component or functionality of the application before writing the corresponding code.
- Write code to fulfill the requirements outlined by the unit tests.
- Run the tests frequently to ensure that new code additions do not break existing functionality.
- Iterate on the development process, writing tests and code incrementally.

Behavior-Driven Development (BDD):

- Define user stories and acceptance criteria for key features of the chat application.
- Implement step definitions to translate feature files into executable tests.
- Write code to satisfy the acceptance criteria defined in the feature files.
- Run feature tests regularly to validate that the application behaves as expected.

Feature-Driven Development (FDD):

- Break down the development process into small, manageable features or functionalities.
- Prioritize features based on user requirements and business value.
- Assign development tasks to team members, focusing on feature ownership and collaboration.
- Implement features iteratively, following the FDD workflow of design, build, and test.

- Conduct regular feature reviews and demonstrations to stakeholders for feedback and validation.

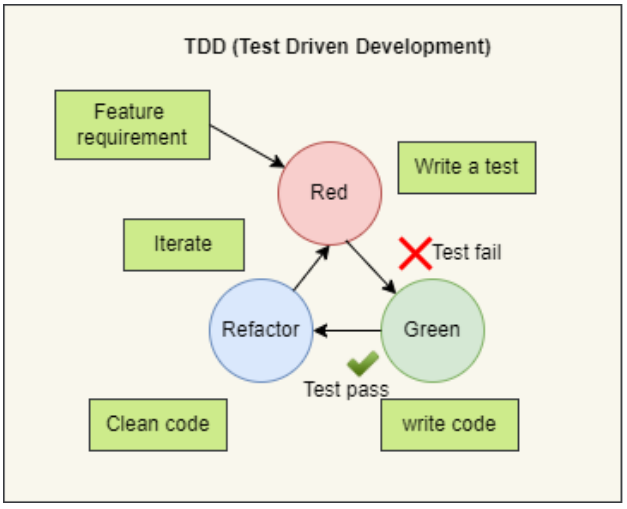
Integration and Deployment:

- Integrate individual components and features into a cohesive chat application.
- Perform integration testing to ensure that all parts of the application work together seamlessly.
- Deploy the chat application to a test environment for further testing and validation.
- Address any issues or bugs identified during testing before final deployment.

Deliverables:

- Documentation outlining the chosen methodologies and their implementation in the chat application project.
- Source code of the chat application, organized according to the chosen development methodologies.
- Test suites demonstrating comprehensive unit tests (TDD), feature tests (BDD), and integration tests (FDD).
- A report summarizing the development process, challenges encountered, and lessons learned from applying TDD, BDD, and FDD in the project.

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.



Test-Driven Development (TDD)

Test-Driven Development (TDD) is a method in software development. Instead of writing code first, developers begin by creating tests. The main goal is to confirm that the

code works correctly by testing it against these pre-established criteria. TDD follows a repetitive process called the “Red-Green-Refactor” cycle:

Red: Write a failing test case. This test represents a specific requirement or functionality you want to implement.

Green: Write the minimum amount of code required to make the failing test pass. This code may be incomplete or inefficient.

Refactor: Refine the code, making it more efficient, readable, and maintainable while ensuring that all tests still pass.

This process continues iteratively, helping developers build a robust and well-tested codebase.

Write Tests First:

- Developers write automated tests for desired functionality before writing any code.

Run Tests:

- Execute the tests to ensure they fail initially, indicating that the code is yet to be implemented.

Write Code:

- Implement the minimum code necessary to pass the tests.

Run Tests Again:

- Re-run the tests to verify that the newly implemented code passes the tests.

Refactor Code (If Needed):

- Refactor the code to improve readability, performance, or maintainability while ensuring all tests still pass.

Repeat:

- Repeat the process for each new functionality or code change.

Benefits of TDD:

Bug Reduction:

Catching and fixing bugs early in the development process reduces the likelihood of critical issues later on.

Improved Code Quality:

Writing tests first encourages developers to focus on designing modular, loosely coupled, and testable code.

Faster Feedback Loop:

Immediate feedback from failing tests helps developers identify issues quickly, leading to faster resolution.

Enhanced Software Reliability:

Comprehensive test suites ensure that software behaves as expected, increasing overall reliability and stability.

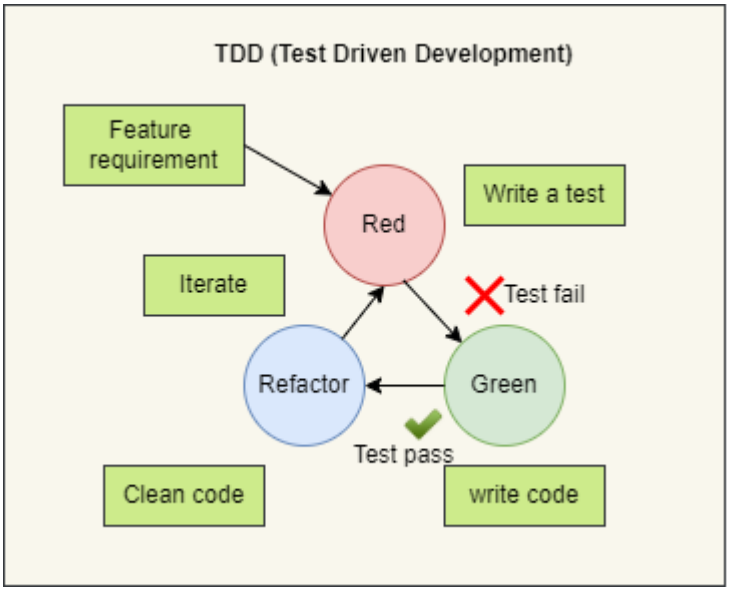
Confidence in Changes:

Developers can make changes confidently, knowing that existing functionality is protected by automated tests.

Test-Driven Development (TDD) promotes a systematic approach to software development, where testing is an integral part of the coding process. By writing tests first, developers can create more reliable, bug-free code, leading to higher-quality software products.

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.

Test-Driven Development (TDD):



Approach:

- Write tests before writing code.
- Develop code to pass the tests.
- Refactor code as needed.

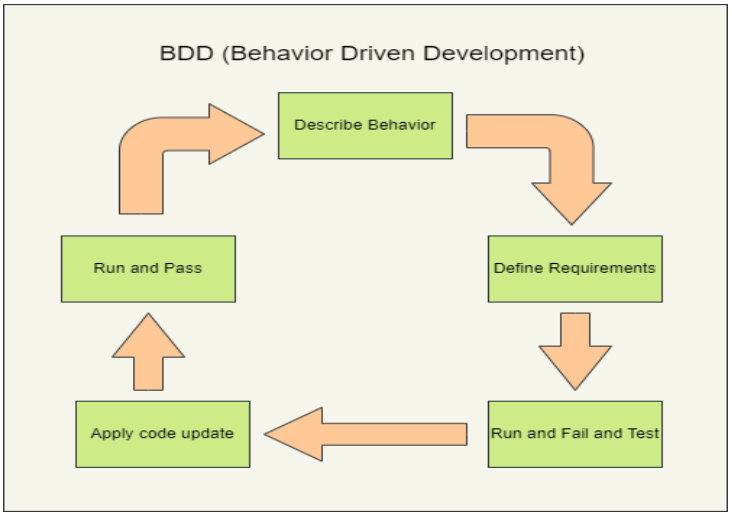
Benefits:

- Early bug detection.
- Improved code quality.
- Increased test coverage.
- Encourages modular and testable code.

Suitability:

- Small to medium-sized projects.
- Projects with clear specifications.
- Projects where requirements are likely to change.

Behavior-Driven Development (BDD):



Approach:

- Define behavior using scenarios written in natural language (Given, When, Then).
- Implement code to fulfill behavior.
- Validate behavior through automated tests.

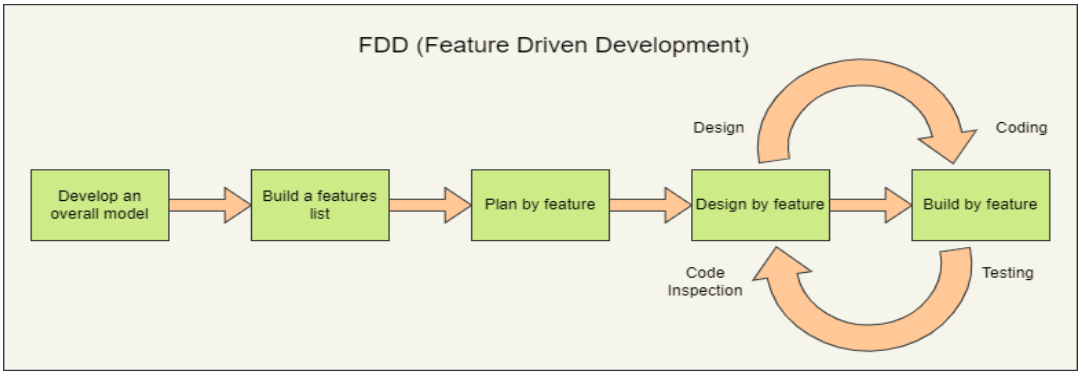
Benefits:

- Enhanced collaboration between stakeholders.
- Improved understanding of requirements.
- Focus on user behavior and outcomes.
- Supports Agile development practices.

Suitability:

- Projects with complex business logic.
- Cross-functional teams.
- Projects with evolving requirements.

Feature-Driven Development (FDD):



Approach:

- Break down development into feature sets.
- Plan, design, and implement each feature set.
- Build, merge, and test features iteratively.

Benefits:

- Emphasis on tangible deliverables.
- Efficient use of resources.
- Clear progress tracking.
- Encourages collaboration and communication.

Suitability:

- Large-scale projects.
- Projects with well-defined requirements.
- Projects requiring fast-paced development.

Each methodology offers unique approaches to software development, catering to different project needs and contexts. TDD emphasizes code quality and test coverage, BDD focuses on user behavior and collaboration, while FDD prioritizes feature delivery and iterative development. Choosing the right methodology depends on project size, complexity, and stakeholder preferences.

1. Agile Project Planning - Create a one-page project plan for a new software feature using Agile planning techniques. Include backlog items with estimated story points and a prioritized list of user stories.

   Project Name: Agile Checkout Process Optimization

   Project Objective: To optimize the checkout process on the e-commerce platform to improve user experience and increase conversion rates.

   Project Team:

   - Product Owner: [Name]
   - Scrum Master: [Name]
   - Development Team: [Team Members]

   Sprint Duration: 3 weeks

   Backlog Items:

   User Story 1: Guest Checkout

   - As a user, I want to be able to checkout as a guest without creating an account.

- **Story Points:** 5

User Story 2: Simplified Checkout Steps

- As a user, I want the checkout process to be simplified with fewer steps.
- **Story Points:** 8

User Story 3: Address Autofill

- As a user, I want my address to be autofilled based on postal code or geolocation.
- **Story Points:** 3

User Story 4: Inline Validation

- As a user, I want inline validation to be implemented to help me correct errors in real-time.
- **Story Points:** 3

User Story 5: Guest Order Tracking

- As a user, I want to track my guest order using a tracking number provided after checkout.
- **Story Points:** 5

Prioritised User Stories:

User Story 2: Simplified Checkout Steps

User Story 1: Guest Checkout

User Story 3: Address Autofill

User Story 4: Inline Validation

User Story 5: Guest Order Tracking

Project Timeline:

Sprint 1 (3 weeks):

Week 1:

User Story 2: Simplified Checkout Steps

User Story 1: Guest Checkout

Week 2:

User Story 3: Address Autofill

User Story 4: Inline Validation

Week 3:

User Story 5: Guest Order Tracking

Testing and Bug Fixes

2. Daily Standup Simulation - Write a script for a Daily Standup meeting for a development team working on the software feature from Assignment 1. Address a common challenge and incorporate a solution into the communication flow.