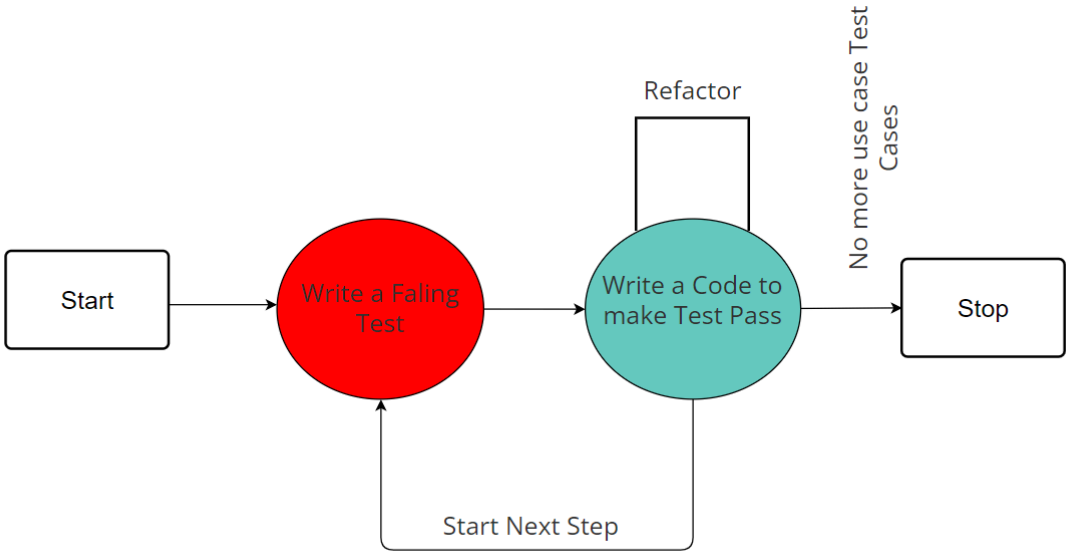
ASSIGNMENT 1

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.

Test-Driven Development (TDD) is a software development methodology that emphasises writing tests before writing the actual code. The main idea behind TDD is to create a feedback loop that guides the development process and helps ensure the reliability and correctness of the software being developed. TDD follows a cyclical pattern, often referred to as the “Red-Green-Refactor” cycle, which consists of the following steps:



1. **Red**: In this phase, you start by writing a test that defines the desired behaviour or functionality of a specific piece of code. Initially, this test will fail because the corresponding code hasn’t been written yet. This failing test is often referred to as a “red” test.
2. **Green**: Once you have a failing test, your next step is to write the minimum amount of code necessary to make the test pass.

This code may not be perfect or efficient; the goal is to satisfy the test's conditions and make it pass. When the test passes, it becomes a "green" test, indicating that the desired functionality has been implemented.

3. **Refactor**: After making the test pass, you can improve the code's design, structure, and efficiency while keeping the test green. Refactoring involves making changes to the code without changing its external behaviour. The tests act as a safety net, helping you catch unintended side effects of your changes.

The TDD cycle is then repeated, starting with the creation of a new test for the next piece of functionality. This process helps ensure that your code is always backed by tests, making it easier to catch bugs and regressions as the codebase evolves.

Benefits:

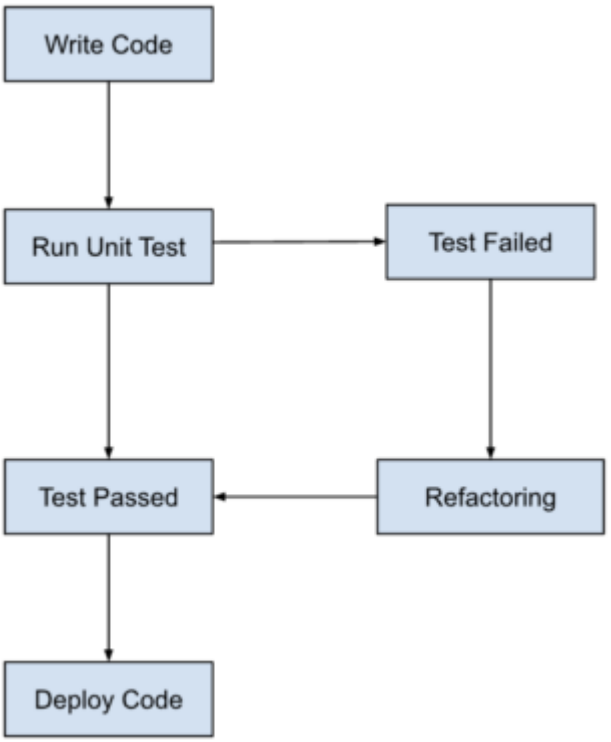
- Bug Reduction: TDD catches bugs early in the development process, reducing the cost of fixing them later.
- Improved Code Quality: By writing tests first, developers are forced to think about the functionality they're implementing, leading to cleaner, more modular code.
- Enhanced Reliability: With a comprehensive suite of tests, developers gain confidence in the reliability of their code, leading to fewer issues in production.

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.

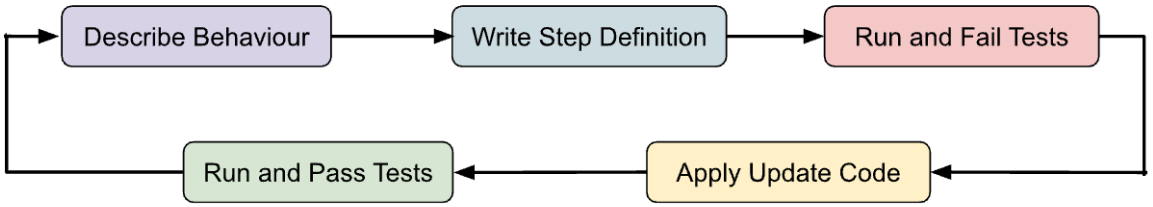
Feature	TDD (Test-Driven Development)	BDD (Behavior-Driven Development)	FDD (Feature-Driven Development)
Focus	Writing tests first	User behaviour and acceptance criteria	Feature decomposition and iterative development
Approach	Test-first, code-second	Collaborative story definition	Top-down planning with iterations
Strengths	Clean, well-tested code, improved design	Clear communication, user focus, early defect detection	Improved project management, reduced risk, stakeholder involvement
Suitability	Unit testing, individual tasks	Agile projects, cross-functional teams	Large, complex projects, requirement-heavy projects
Key Practice	Writing failing automated tests before code	Defining user stories with "Given-When-Then" scenarios	Breaking down features into small, testable tasks
Communication	Primarily between developers	Between developers, testers, and stakeholders	Between all project stakeholders
Project Management	Less emphasis	Less emphasis	Strong focus on planning and iterations

1. Test-Driven Development (TDD):



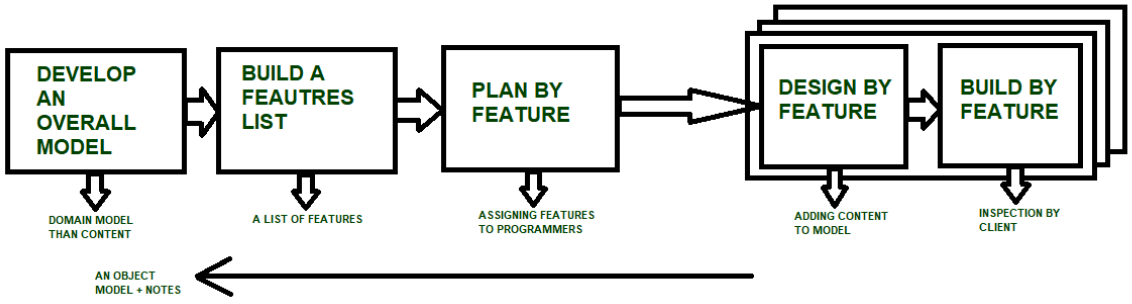
- **Approach: Test First, Code Later!** Imagine a developer writing failing automated tests (the test tube) before writing a single line of code. These failing tests become the blueprint for the code, ensuring it fulfils its purpose. Once the code makes the tests pass, it's refactored for efficiency.
- **Benefits:**
    - **Strong Foundation:** Clean, well-tested code forms the backbone of your application.
    - **Improved Design:** Tests guide development, leading to well-structured and maintainable code.
    - **Reduced Bugs:** Early detection and prevention of bugs saves time and effort in the long run.
- **Suitability:** Perfect for unit testing and individual developer tasks, promoting a test-first mentality.

2. Behaviour-Driven Development (BDD):



- **Approach: Focus on the User!** BDD prioritises user behaviour. Stakeholders (represented by the handshake) come together to define user stories (conversation bubble) with clear acceptance criteria using the "Given-When-Then" format. This ensures everyone is on the same page about what the software should do.
- **Benefits:**
    - **Communication Bridge:** Clear communication between developers, testers, and stakeholders leads to a shared understanding.
    - **User-Centric Design:** Development revolves around user needs, leading to a product that truly meets their expectations.
    - **Early Defect Detection:** Misunderstandings are caught early through collaborative story definition, minimising bugs later.
- **Suitability:** Thrives in Agile projects and cross-functional teams where collaboration is key.

3. Feature-Driven Development (FDD):



- **Approach: Break Down and Conquer!** FDD utilises a top-down planning approach. Features (puzzle piece) are meticulously broken down into smaller, testable tasks (checklist) with rigorous

inspections (magnifying glass) throughout the development cycle. This ensures quality control and reduces risk.

- **Benefits:**
    - **Project Management Powerhouse:** Improved project management with clear goals, milestones, and risk mitigation strategies.
    - **Reduced Risk:** Early identification of potential issues helps prevent costly rework.
    - **Stakeholder Engagement:** Strong emphasis on stakeholder involvement throughout the development process.
- **Suitability:** Ideal for large, complex projects with extensive requirements, where meticulous planning is crucial.