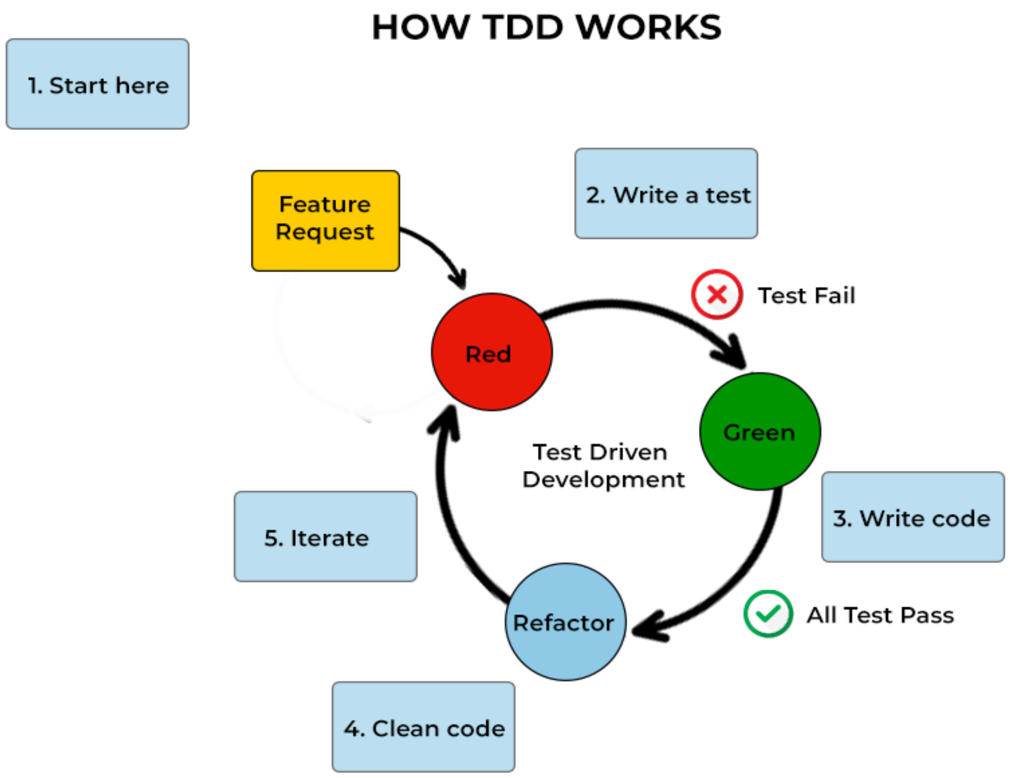
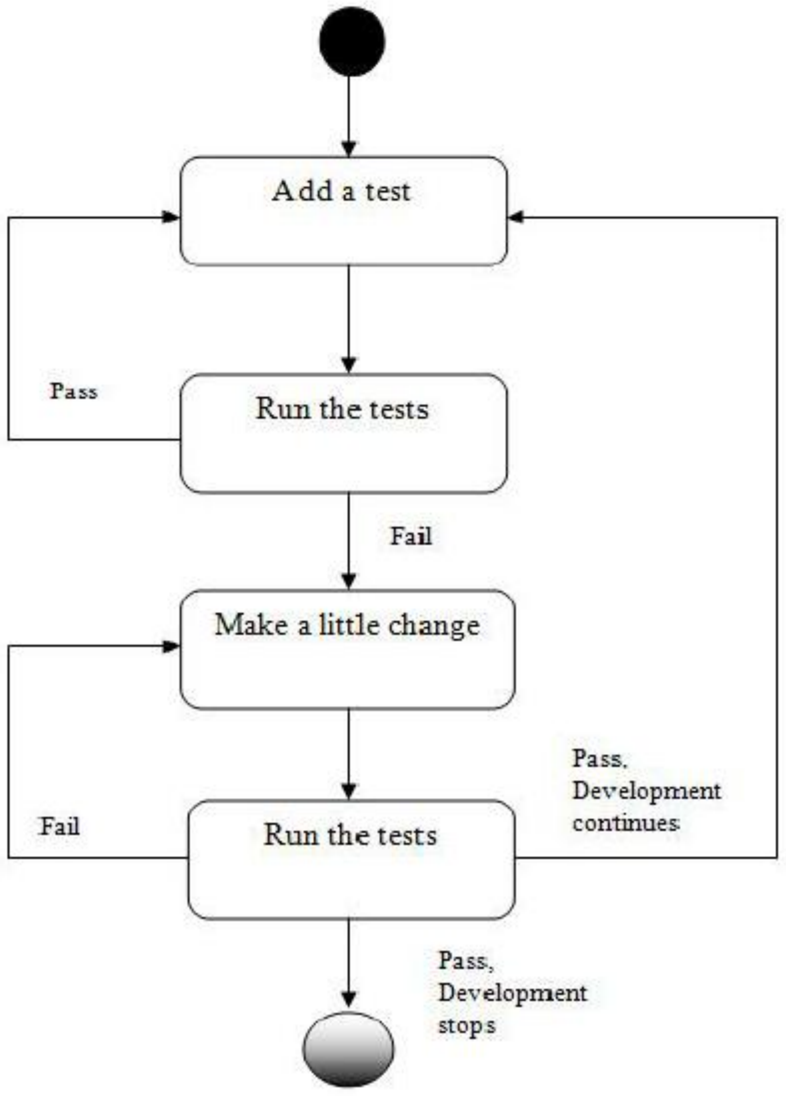
NAME: ANJALI KUMARI

DAY3-

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.





Title: Test-Driven Development (TDD) Process

1. Introduction

Definition: TDD is a software development process that emphasizes writing tests before writing code.

2. Steps of TDD:

-Write a Test: Start by writing a test case for the functionality you want to implement.

-Run the Test: Execute the test. It should fail since the functionality isn't implemented yet.

-Write Code: Implement the code to make the test pass.

-Run All Tests: Run all tests to ensure the new code didn't break existing functionality.

-Refactor Code: Improve the code without changing its functionality. Run tests again to ensure everything still works.

3. Benefits of TDD:

-Bug Reduction: By writing tests first, developers catch bugs early in the development process.

-Improved Code Quality: TDD encourages writing modular, testable, and maintainable code.

-Faster Development: Although writing tests may seem time-consuming, it ultimately saves time by reducing debugging and rework.

4. Fostered Software Reliability:

-TDD ensures that each piece of code has corresponding tests, increasing software reliability.

-Since tests are run frequently, developers gain confidence in the correctness of their code.

5. Conclusion:

Test-Driven Development promotes a structured approach to development, resulting in higher quality, more reliable software.

Advantages:

Early Bug Detection: TDD helps catch bugs early in the development process since tests are written before the code. This reduces the cost and effort required to fix bugs later.

Improved Code Quality: TDD encourages writing modular, testable, and maintainable code. Since developers must think about testing from the beginning, it often leads to cleaner and more organized code.

Increased Confidence: Developers gain confidence in their code as they see tests pass with each iteration. This confidence leads to a greater willingness to refactor and improve the codebase.

Faster Development: While it may seem counterintuitive at first, TDD often speeds up development in the long run. Writing tests upfront helps clarify requirements and design, leading to more efficient implementation.

Better Documentation: Tests serve as executable documentation for the codebase. They provide insights into how the code should behave and serve as living examples of its usage.

Disadvantages:

Initial Learning Curve: TDD requires a shift in mindset and may have a steep learning curve for developers who are new to the practice. Learning to write effective tests and understanding when and what to test can take time.

Time-Consuming: Writing tests upfront can initially slow down the development process. Developers need to invest time in writing tests before writing the actual code, which may seem inefficient, especially for tight deadlines.

Maintaining Tests: Test suites can become complex and difficult to maintain, especially in large projects. Refactoring or changing existing code may require corresponding changes to the tests, adding overhead to development efforts.

False Sense of Security: While TDD can catch many bugs, it's not a silver bullet. Developers may fall into the trap of assuming that passing tests equate to bug-free code, leading to potential blind spots in testing.

Over-testing: In some cases, developers may write too many tests, resulting in redundant or unnecessary test cases. This can lead to bloated test suites, increased maintenance overhead, and slower test execution times.

Overall, while Test-Driven Development offers significant benefits in terms of code quality, bug reduction, and developer confidence, it also comes with its own set of challenges and trade-offs that need to be carefully considered before adopting the practice.

Assignment2 : 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):

Definition: Test-Driven Development (TDD) is a software development approach where tests are written before the code. Developers write failing tests to specify the desired behavior of the system, then write code to make the tests pass. TDD follows a cycle of writing a failing test, writing the minimum code to pass the test, and then refactoring the code while ensuring all tests continue to pass.

Key Points:

Tests are written before implementation code.

Emphasizes writing small, focused tests that verify specific behavior.

Helps ensure code correctness, improve design, and enable safe refactoring.

Behavior-Driven Development (BDD):

Definition: Behavior-Driven Development (BDD) is a software development methodology that extends TDD by focusing on the behavior of the system from the perspective of its stakeholders. BDD uses a specialized language, such as Gherkin, to describe system behavior in terms of scenarios or examples. BDD encourages collaboration between developers, testers, and business stakeholders to define and understand system behavior, leading to clearer requirements and improved communication.

Key Points:

Focuses on defining system behavior using scenarios or examples.

Uses a common language (e.g., Gherkin) to describe behavior in a structured format.

Promotes collaboration and shared understanding among stakeholders.

Feature-Driven Development (FDD):

Definition: Feature-Driven Development (FDD) is an iterative and incremental software development methodology that focuses on delivering features in small, manageable increments. FDD breaks down development into five basic activities: developing an overall model, building feature lists, planning by feature, designing by feature, and building by feature. FDD emphasizes frequent releases, clear progress tracking, and a feature-centric approach to development.

Key Points:

Emphasizes feature delivery and progress tracking.

Breaks development into manageable feature sets.

Defines clear roles and responsibilities within the development team.

Brief overview of TDD, BDD, and FDD methodologies.

1. Test-Driven Development (TDD):

Visual: Icon representing writing tests before code.

Approach: Write tests before implementing functionality.

Benefits:

Early bug detection.

Improved code quality.

Faster development with frequent testing.

Suitable for: Projects where code correctness and maintainability are crucial.

2. Behavior-Driven Development (BDD):

Visual: Icon representing collaboration between stakeholders.

Approach: Define behavior using scenarios or examples.

Benefits:

Collaboration between developers, testers, and business stakeholders.

Improved communication and shared understanding.

Focus on user behavior and acceptance criteria.

Suitable for: Projects with complex business logic or user interactions.

3. Feature-Driven Development (FDD):

Visual: Icon representing feature-based development.

Approach: Break down development into small, manageable features.

Benefits:

Emphasis on feature delivery and progress tracking.

Clear roles and responsibilities within the development team.

Iterative development based on feature prioritization.

Suitable for: Large-scale projects with a focus on feature delivery and team scalability.

4. Comparison:

Visual: Side-by-side comparison chart or table.

Factors to compare:

Approach (e.g., test-first, behavior-driven, feature-based).

Key benefits.

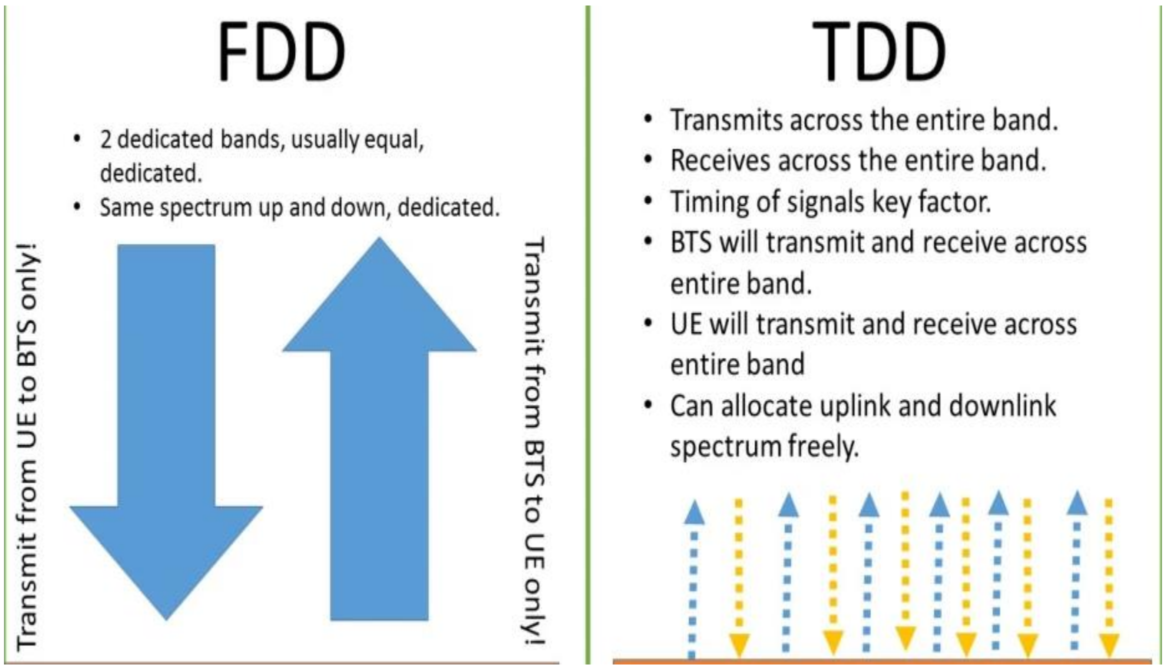
Suitable project contexts.

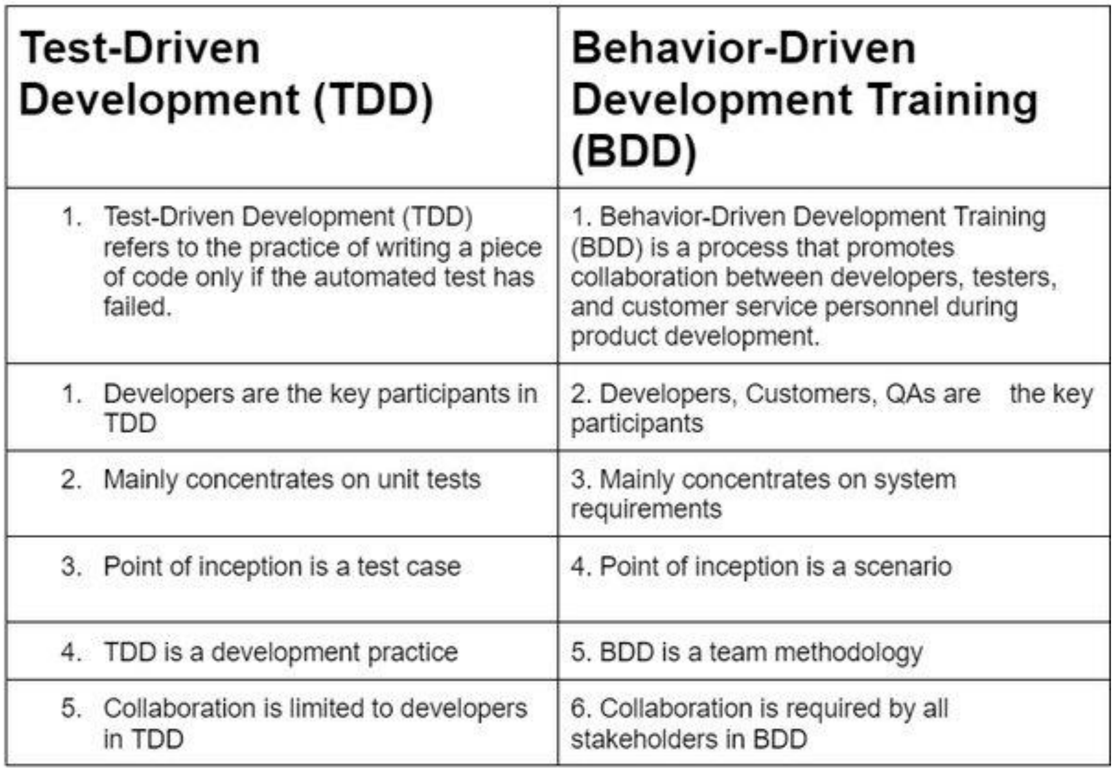
Highlight strengths and limitations of each methodology.

5. Conclusion:

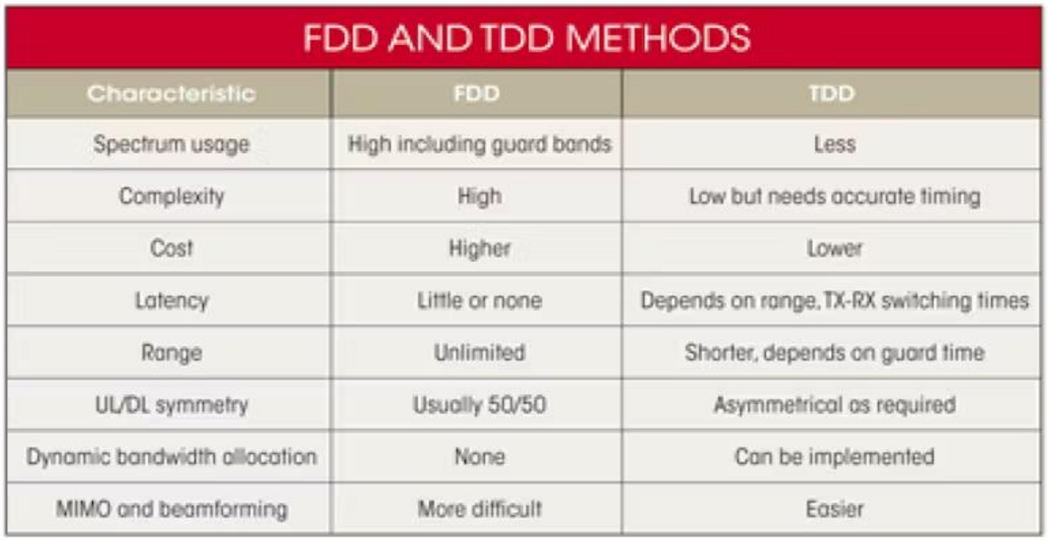
Summary of key takeaways.

Consideration of factors to choose the most suitable methodology for a specific project.


FDD
• 2 dedicated bands, usually equal, dedicated.
• Same spectrum up and down, dedicated.
Transmit from UE to BTS only!
Transmit from BTS to UE only!
TDD
• Transmits across the entire band.
• Receives across the entire band.
• Timing of signals key factor.
• BTS will transmit and receive across entire band.
• UE will transmit and receive across entire band
• Can allocate uplink and downlink spectrum freely.

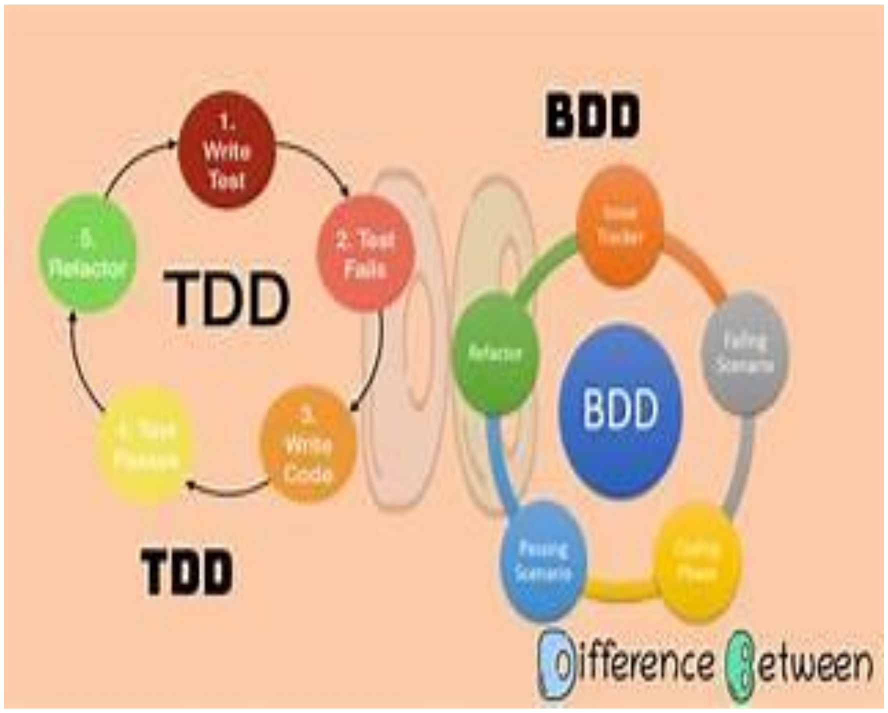


Test-Driven Development (TDD)	Behavior-Driven Development Training (BDD)
1. Test-Driven Development (TDD) refers to the practice of writing a piece of code only if the automated test has failed.	1. Behavior-Driven Development Training (BDD) is a process that promotes collaboration between developers, testers, and customer service personnel during product development.
1. Developers are the key participants in TDD	2. Developers, Customers, QAs are the key participants
2. Mainly concentrates on unit tests	3. Mainly concentrates on system requirements
3. Point of inception is a test case	4. Point of inception is a scenario
4. TDD is a development practice	5. BDD is a team methodology
5. Collaboration is limited to developers in TDD	6. Collaboration is required by all stakeholders in BDD



FDD AND TDD METHODS

Characteristic	FDD	TDD
Spectrum usage	High including guard bands	Less
Complexity	High	Low but needs accurate timing
Cost	Higher	Lower
Latency	Little or none	Depends on range, TX-RX switching times
Range	Unlimited	Shorter, depends on guard time
UL/DL symmetry	Usually 50/50	Asymmetrical as required
Dynamic bandwidth allocation	None	Can be implemented
MIMO and beamforming	More difficult	Easier


TDD
1. Write Test
2. Test Fails
3. Write Code
4. Test Passes
5. Refactor
TDD
BDD
BDD
Issue Tracker
Failing Scenario
Coding Phase
Passing Scenario
Refactor
Difference Between