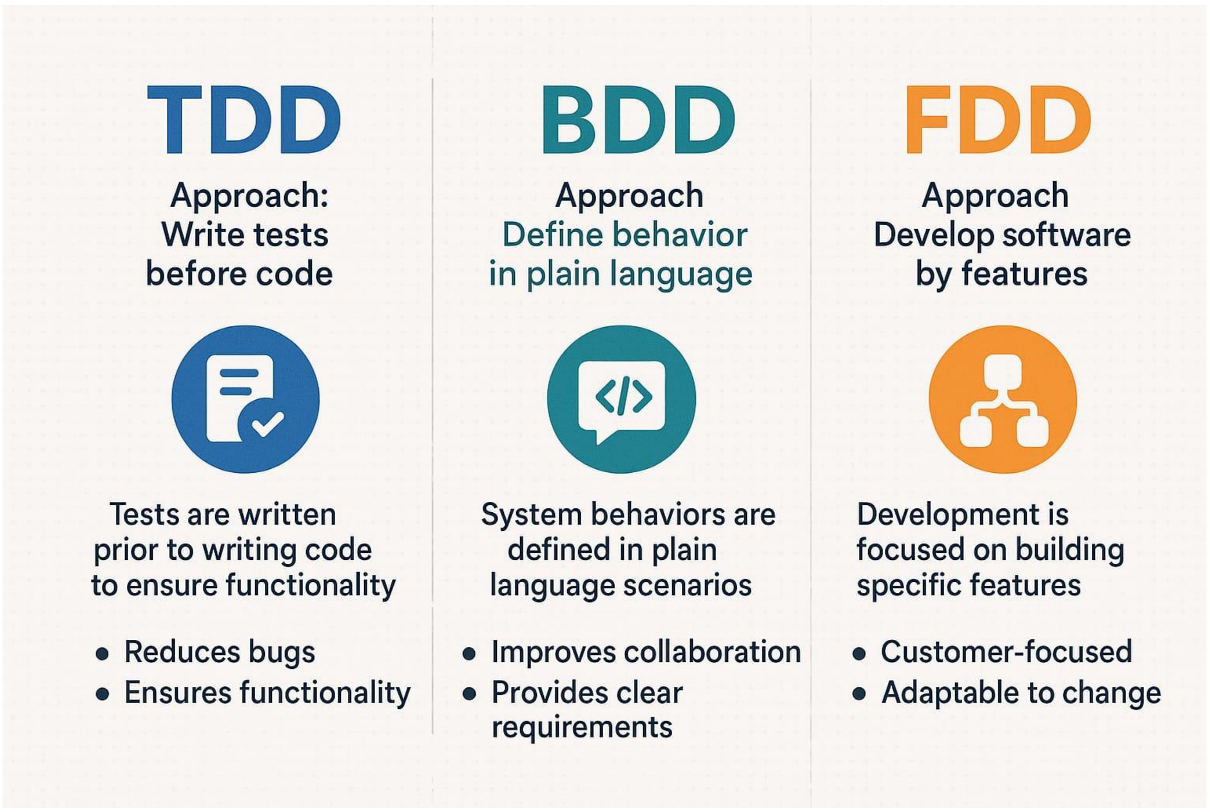
INFROGRAPHIC ILLUSTRATION OF TDD, BDD AND FDD


TDD
Approach:
Write tests before code
Tests are written prior to writing code to ensure functionality
• Reduces bugs
• Ensures functionality
BDD
Approach
Define behavior in plain language
System behaviors are defined in plain language scenarios
• Improves collaboration
• Provides clear requirements
FDD
Approach
Develop software by features
Development is focused on building specific features
• Customer-focused
• Adaptable to change

1.TDD (Test-Driven Development):

Approach:

- Write tests before writing actual code. Code is developed to pass these tests.

Benefits:

- Reduces bugs early.
- Ensures functionality from the start.
- Makes refactoring safer.

Suitability:

- Best for complex algorithms, critical systems, or backend services where correctness is vital.
- Ideal when automated unit testing is important.

2.BDD (Behaviour-Driven Development):

Approach:

- Define system behaviours using plain language scenarios (e.g., "Given-When-Then" format).

Benefits:

- Improves collaboration between developers, testers, and non-technical stakeholders.
- Provides clear, shared understanding of requirements.
- Bridges the gap between business and technical teams.

Suitability:

- Excellent for user-centric applications, web applications, or projects involving multiple stakeholders.
- Useful where user behaviour is a major focus.

3.FDD (Feature-Driven Development):

Approach:

- Develop software by building features — small, client-valued functions.

Benefits:

- Highly customer focused.
- Offers frequent, tangible progress.
- Adaptable to changing requirements.

Suitability:

- Best for large, complex systems where delivering client-visible features quickly is critical.
- Suitable for enterprise applications and projects needing clear milestones