Assignment-2

Name: Boyapati Durgamallikarjuna
RPS UserID: 21951

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.

Comparison between TDD, FDD and FDD:

Aspect	TDD (Test-Driven Development)	BDD (Behavior-Driven Development)	FDD (Feature-Driven Development)
Definition	Writing tests before code to guide development and ensure functionality.	Writing behavior specifications in natural language to align with user expectations.	Developing software by breaking it into small, tangible features and delivering them iteratively.
Focus	Code correctness and low-level unit testing.	User behavior and collaboration between developers, testers, and stakeholders.	Delivering complete features with emphasis on upfront planning and design.
Process Flow	1. Write Test 2. Write Code 3. Refactor 4. Repeat	1. Write Features 2. Define Scenarios 3. Implement Code 4. Execute Tests	1. Develop Overall Model 2. Build Features Iteratively 3. Test and Deliver Features
Primary Artifacts	Unit tests (automated).	Gherkin-style scenarios (Given-When-Then).	Feature list, models, and documentation.
Key Characteristics	- Developer-centric. - Tests drive design. - Promotes refactoring.	- Stakeholder collaboration. - Focus on readable tests. - Behavior-driven focus.	- Feature-centric. - Scalable for large teams. - Emphasis on planning and structured process.

Benefits	- Immediate feedback. - Improves code quality. - Reduces defects early.	- Clear collaboration. - Ensures user requirements are met. - Easy-to-read test scenarios.	- Modular and incremental. - Scalability for large projects. - Well-defined scope with measurable progress.
Suitability	- Projects requiring strong unit test coverage. - Teams with technical expertise in testing frameworks.	- Applications where user behavior and business rules are crucial. - Teams with cross-functional collaboration.	- Large-scale projects. - Teams that prefer planning and structured development. - Projects requiring scalability.
Challenges	- Time-consuming for small tasks. - Steeper learning curve for beginners.	- Can be verbose for small projects. - Requires effort in stakeholder collaboration and understanding.	- Heavily reliant on upfront planning. - May lead to delays if features are not well-defined.
Example Frameworks	JUnit, NUnit, pytest.	Cucumber, SpecFlow, Behave.	Customizable based on project; typically uses general software modeling and tracking tools.