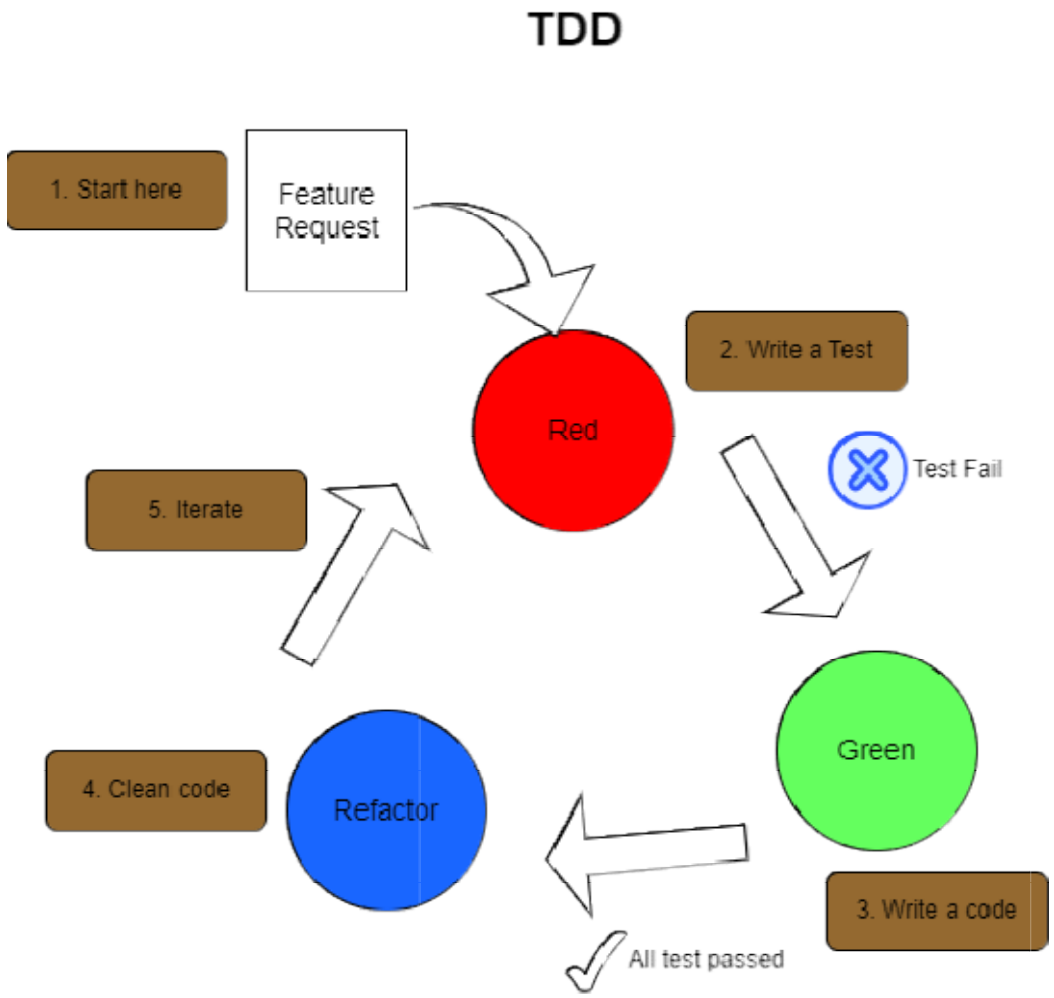
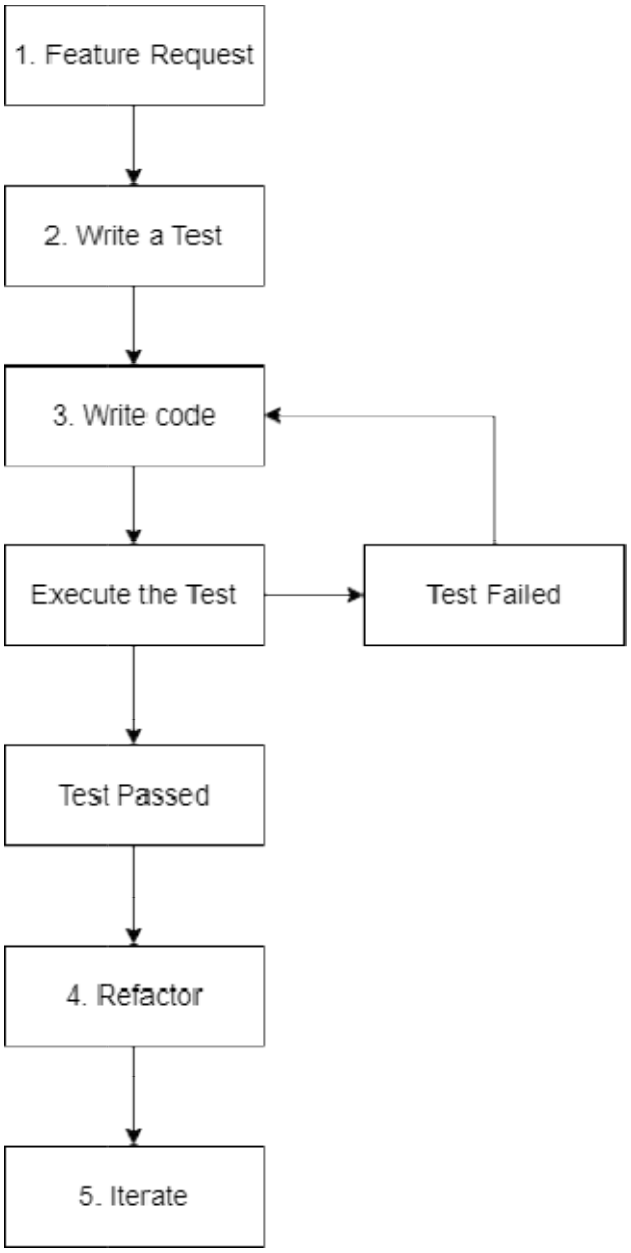
Assignment 2: Produce a comparative info graphic of TDD, BDD, and FDD methodologies. Illustrate their unique approaches, benefits, and suitability for different software development contexts. Use visuals to enhance understanding.

Test-Driven Development (TDD):

Info graphic diagram:



Test - Driven Development (TDD) Process Flow:



Approach:

TDD follows a cycle where developers write failing tests before writing any production code. They then implement the minimum code required to pass the tests and subsequently refactor the codebase to improve its design and maintainability. This approach ensures that code is thoroughly tested and validated throughout the development process, resulting in a more robust and reliable software product.

Benefits:

• Early Bug Detection: TDD's preemptive testing methodology detects bugs in the early stages of development, fostering a proactive approach to software quality.

• Improved Design: By encouraging modular and loosely coupled code, TDD facilitates cleaner and more maintainable codebases, promoting long-term project sustainability.

• Regression Testing Confidence: Through a comprehensive suite of tests, TDD instills confidence in developers to refactor code without compromising existing functionality, enhancing code stability and scalability.

Suitability:

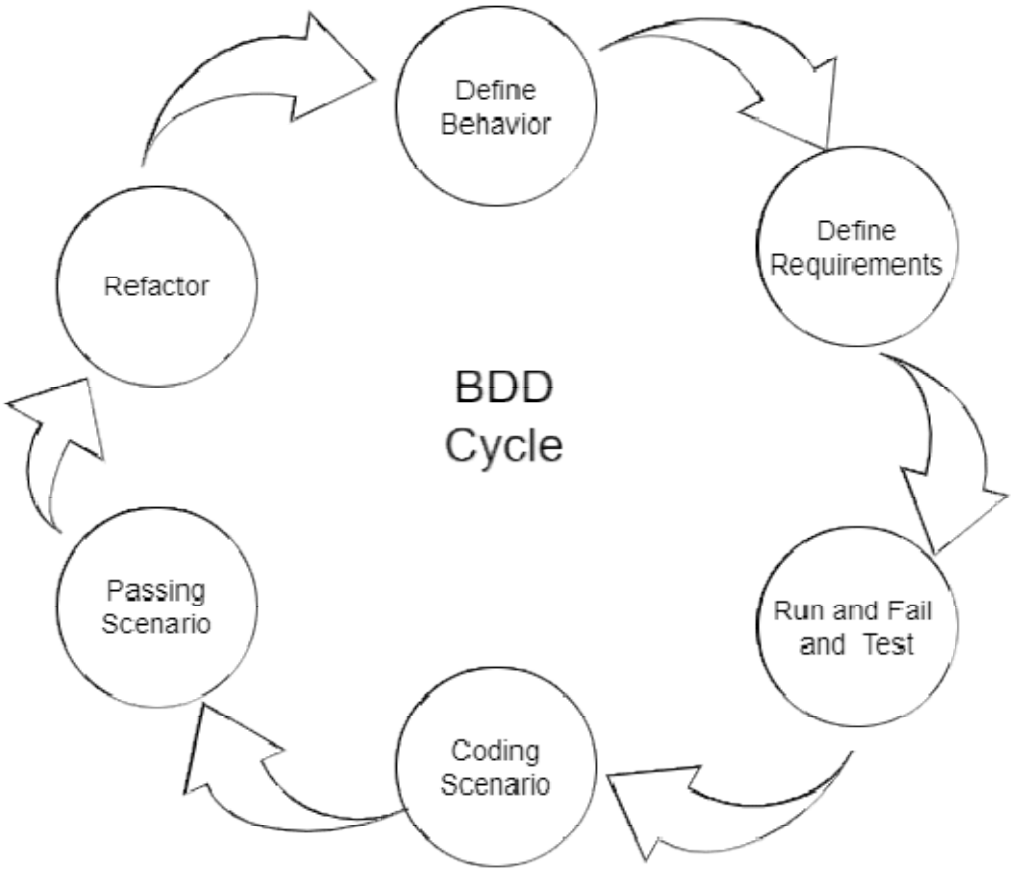
TDD excels in projects with well-defined and stable requirements, particularly for small to medium-sized endeavors with clear and testable specifications.

Real-world Example:

• In the development of a social media platform, TDD ensures that essential features like posting updates, commenting on posts, and sending friend requests function flawlessly. By prioritizing test-driven development, potential issues are identified and rectified early, ensuring a reliable user experience.

Behavior-Driven Development (BDD):

Info graphic diagram:



Approach:

BDD shifts the focus from technical implementation details to the behavior of the system as perceived by its stakeholders. It encourages collaboration between developers, testers, and business stakeholders to define system behavior using a domain-specific language. By articulating requirements in plain language, BDD facilitates better communication and understanding among team members, leading to a more user-centric approach to development.

Benefits:

• Improved Communication: BDD's emphasis on a shared language enhances communication among diverse stakeholders, facilitating a cohesive understanding of system behavior.

• Increased Clarity: By focusing on behavior, BDD ensures that requirements are clearly defined and comprehensible to all team members, minimizing misunderstandings and errors.

• User-Centric Development: BDD prioritizes user stories and acceptance criteria, aligning development efforts with user needs and expectations, thereby enhancing product relevance and satisfaction.

Suitability:

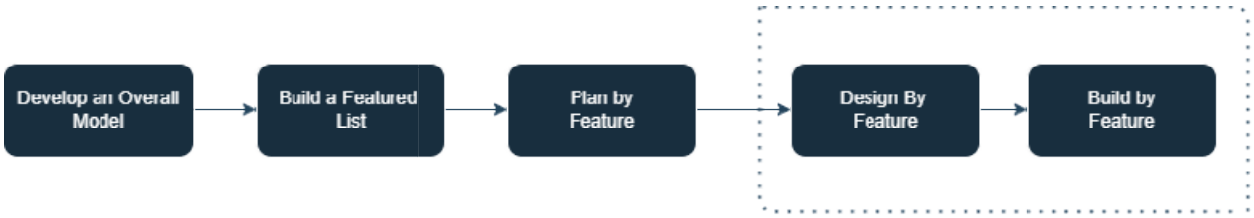
• BDD is ideal for projects with complex business logic and evolving requirements, fostering collaboration and ensuring alignment with user needs.

Real-world Example:

• Consider the development of an e-commerce platform, where BDD enables teams to define and test features such as product search and checkout processes. By focusing on user behavior and acceptance criteria, development efforts are aligned with customer expectations, ensuring a seamless shopping experience.

Feature-Driven Development (FDD):

Info graphic diagram:



Approach:

FDD emphasizes iterative and incremental development, with a primary focus on building features in a systematic manner. It starts with developing an overall model of the system architecture, followed by creating a feature list, planning, designing, and implementing features iteratively. FDD promotes clear ownership and accountability by assigning dedicated feature owners, ensuring that each feature is developed efficiently and effectively.

Benefits:

• Iterative Development: FDD's iterative approach allows for incremental feature delivery, facilitating early value realization and adaptability to evolving requirements.

• Clear Ownership: Assigning owners to each feature promotes accountability and ensures efficient development, facilitating effective tracking and management of project progress.

• Emphasis on Design: FDD prioritizes comprehensive modeling and design before implementation, resulting in well-structured and maintainable codebases, thereby enhancing project scalability and sustainability.

Suitability:

• FDD is well-suited for projects with complex feature sets and a need for clear ownership and accountability, promoting systematic and disciplined development.

Real-world Example:

For an enterprise resource planning (ERP) system, FDD streamlines the development of features such as inventory management and customer relationship management. By breaking down development into features and assigning dedicated owners, project accountability is ensured, driving efficient and organized development efforts.

Overall comparison chart of TDD, BDD and FDD,

Feature	TDD	BDD	FDD
Approach	Focuses on testing code functionality through unit tests written before code implementation.	Concentrates on defining the behavior of the system through user stories and scenarios.	Emphasizes the design and development of features based on customer requirements and feedback.
Benefits	Early bug detection, improved code quality, and faster development cycles.	Improved collaboration between developers, testers, and business stakeholders, leading to better alignment of software with user needs.	Streamlined development process, enhanced customer satisfaction, and adaptability to changing requirements.
Suitability	Ideal for projects requiring a high level of code reliability and maintainability, such as software libraries and frameworks.	Well-suited for projects with complex business logic and user interactions, such as web applications and software-as-a-service	Suitable for projects with evolving requirements and a focus on delivering value incrementally, such as agile development

		(SaaS) products.	environments.
Development Focus	Code-centric approach, with a primary focus on writing tests to drive code implementation.	Behavior-centric approach, with an emphasis on defining desired behaviors and outcomes from the user's perspective.	Feature-centric approach, with a focus on delivering specific features based on customer needs and priorities.
Collaboration	Encourages collaboration between developers and testers to ensure code meets specifications and requirements.	Promotes collaboration between developers, testers, product owners, and other stakeholders to ensure alignment with user needs and expectations.	Facilitates collaboration between cross-functional teams to prioritize features and deliver value iteratively.
Testing Scope	Primarily focuses on unit testing individual components and functions of the application.	Extends testing scope to include acceptance testing, ensuring that the application meets user expectations and requirements.	Incorporates testing at both the feature and system level to validate functionality and ensure seamless integration.
Feedback Mechanisms	Provides rapid feedback on code changes, allowing developers to identify and fix issues early in the development process.	Provides feedback on system behavior and user interactions, helping teams validate software functionality and ensure alignment with user needs.	Offers feedback on feature implementation and customer satisfaction, enabling teams to iterate and improve based on user feedback and market trends.
Real World Example	The development team employs TDD to ensure the reliability and maintainability of critical components such as the payment processing module in an e-commerce platform project.	The product team uses BDD to define the behavior of user-facing features such as the checkout process in an e-commerce platform project.	The project follows FDD principles to deliver new features based on customer needs and feedback, such as order tracking and customer reviews in an e-commerce platform project.