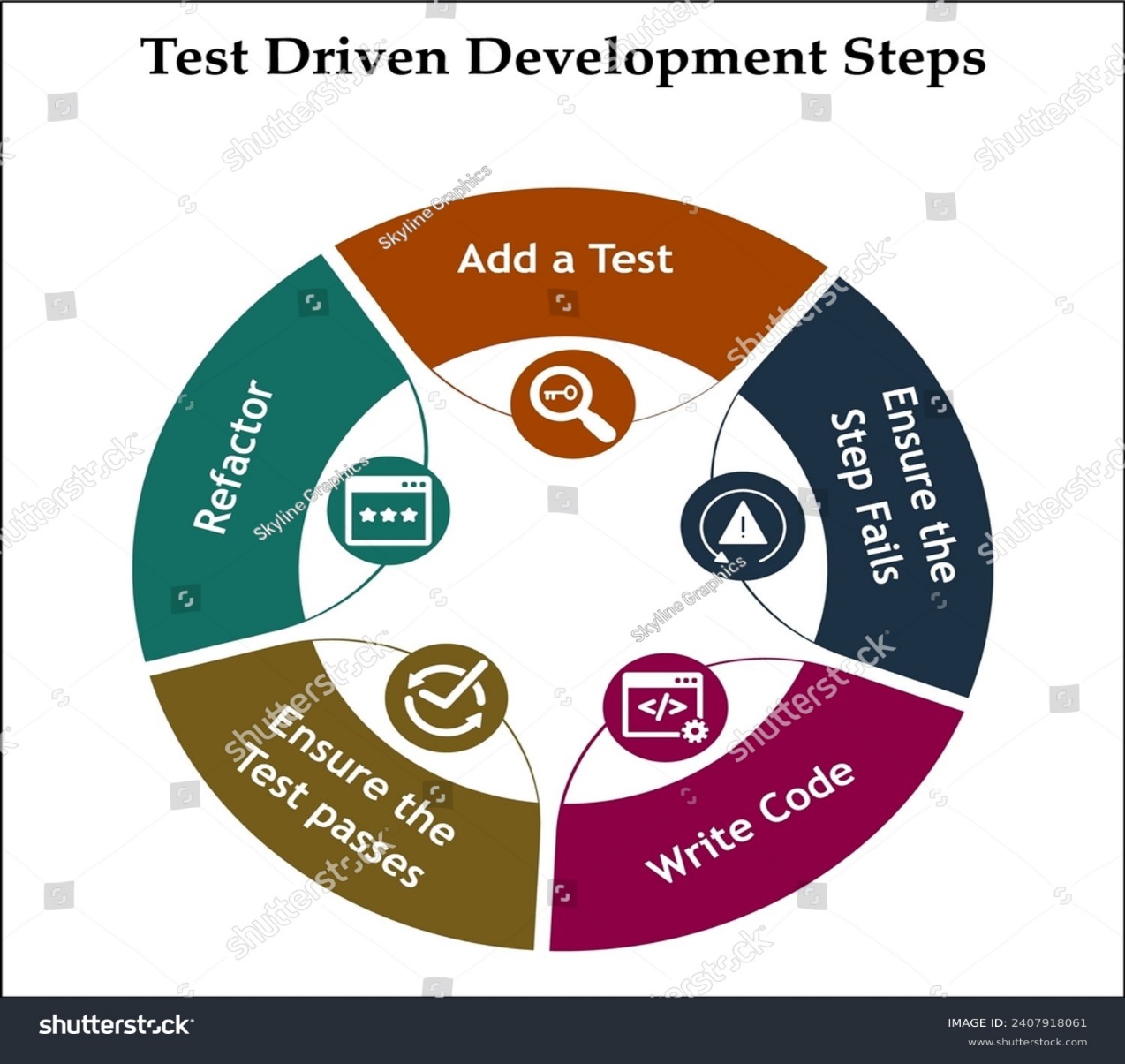
Produce a comparative infographic of TDD, BDD, and FDD methodologies. Illustrate their unique approaches, benefits, and suitability for different software development contexts. Use visuals to enhance understanding.

**Test-Driven Development (TDD)**



**Approach:**

* Write a failing test before writing the functional code.
* Write the minimal code to pass the test.
* Refactor the code for readability and maintainability.
* Repeat the cycle for each new feature.

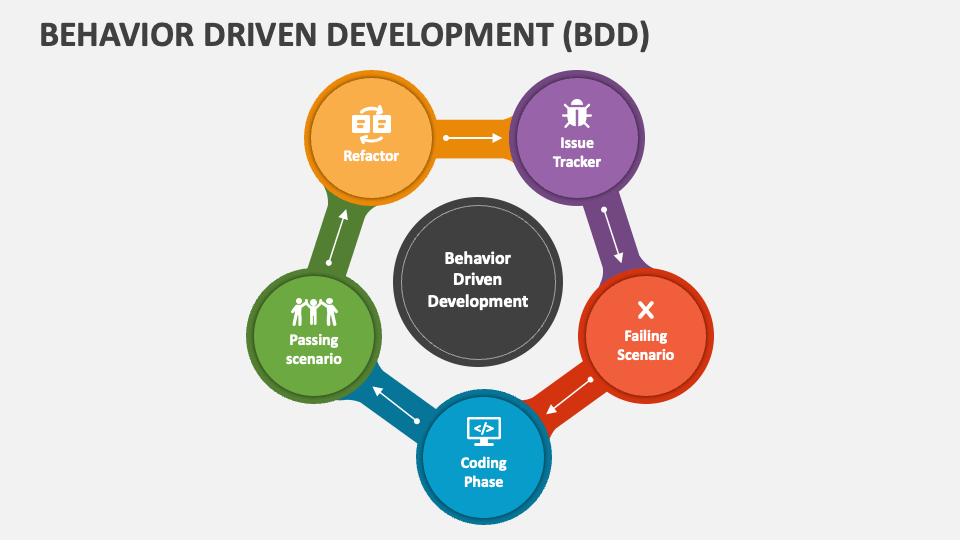
**Benefits:**

* Early bug detection.
* Ensures every feature has corresponding tests.
* Facilitates continuous refactoring.
* Results in cleaner, more maintainable code.

**Suitability:**

* Ideal for complex applications requiring robust and reliable code.
* Suited for projects where code quality and maintainability are a priority.

**Behaviour-Driven Development (BDD)**



**Approach:**

* Define user stories to describe the desired behaviour from the user's perspective.
* Translate user stories into scenarios using a language like Gherkin.
* Write behaviour tests based on these scenarios.
* Implement the functionality to full fill the behaviour described in the tests.
* Maintain continuous collaboration among all team members.

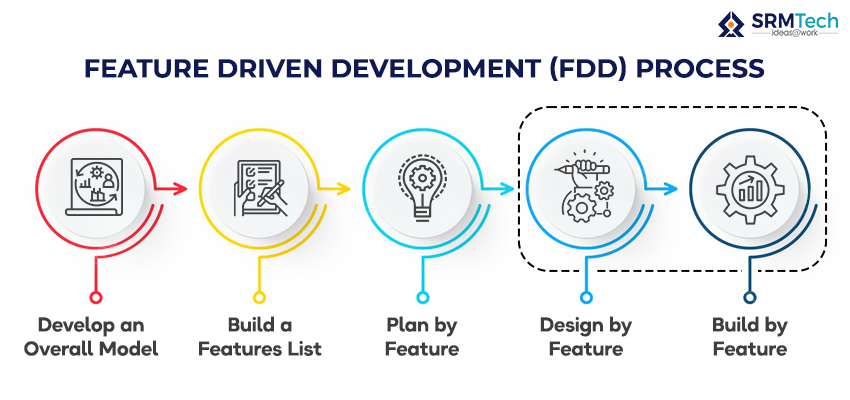
**Benefits:**

* Enhances communication between technical and non-technical stakeholders.
* Provides clear, behaviour-focused scenarios, reducing misunderstandings.
* Ensures development aligns with user expectations.
* Creates living documentation that reflects user-centric outcomes.

**Suitability:**

* Well-suited for projects where user experience and behaviour are critical.
* Ideal for web and mobile applications.

**Feature-Driven Development (FDD)**



**Approach:**

* Start by creating a model of the desired features.
* Build a feature list and plan by feature.
* Design by feature and then by class.
* Insist on frequent builds and integration.
* Component and feature teams are used to implement the features.

**Benefits:**

* Provides a clear direction with an initial feature set and model.
* Facilitates regular builds, ensuring early integration issues are addressed.
* Encourages client involvement and feedback through short iteration cycles.
* Results in a feature-rich, incremental delivery of software.

**Suitability:**

* Suited for large teams working on complex, enterprise-scale systems.
* Ideal for projects requiring a high degree of client involvement and iterative feedback.

**Visual Comparison**

| **Aspect** | **TDD** | **BDD** | **FDD** |
| --- | --- | --- | --- |
| **Focus** | Code functionality and reliability | End-user experience and application behaviour | Feature set and client-valued functionality |
| **Collaboration** | Developer-centric | Cross-functional (developers, QA, business stakeholders) | Client and feature team collaboration |
| **Testing Level** | Unit tests for small pieces of functionality | Behaviour tests for user interactions and scenarios | Feature completion and integration tests |
| **Documentation** | Technical documentation for developers | Living documentation understandable by all stakeholders | Feature list and model for client and team reference |
| **Tool Examples** | JUnit, NUnit, TestNG | Cucumber, SpecFlow, Behat | None specific; depends on the development environment |
| **Ideal use cases** | Complex applications with a focus on code quality (e.g., financial software) | User-centric applications like web and mobile apps | Large-scale systems with iterative client feedback (e.g., enterprise solutions) |

**Conclusion:**

Each development methodology has its unique approach and benefits, making them suitable for different software development contexts. TDD is ideal for ensuring code reliability, BDD excels in aligning development with user expectations, and FDD is designed for large teams working on complex systems with iterative feedback loops. The choice of methodology should be based on the project's requirements, team composition, and client involvement.