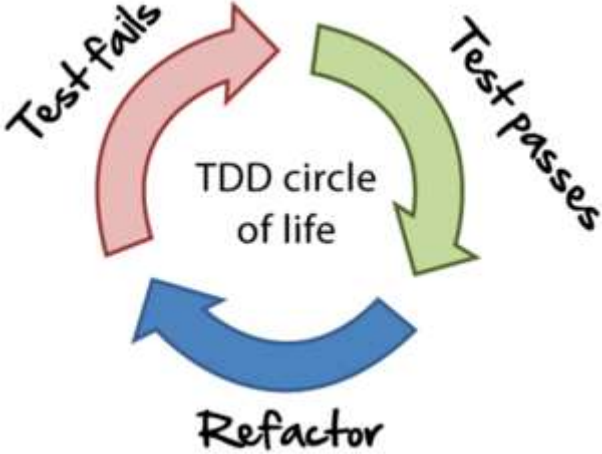


4. Create an infographic illustrating the Test-Driven Development (TDD) process. Highlight steps like writing tests before code, benefits such as bug reduction, and how it fosters software reliability.

<div>Test-Driven Development (TDD) Process</div> <div>1. Write Test:</div> <div><div>- Objective: Define a test for the desired functionality.</div><div>- Action: Write a test that fails because the functionality isn't implemented yet.</div><div>- Example: Write a test to check if a function returns the correct sum of two numbers.</div></div> <div>2. Run Test:</div> <div><div>- Objective: Verify that the test fails as expected.</div><div>- Action: Run the test to ensure it fails due to the missing functionality.</div><div>- Example: Run the test for the sum function, expecting a failure.</div></div> <div>3. Write Code:</div> <div><div>- Objective: Implement the functionality to make the test pass.</div><div>- Action: Write the simplest code that makes the test pass.</div><div>- Example: Write code for the sum function to return the correct sum of two numbers.</div></div>	<div>4. Run Test Again</div> <div><div>- Objective: Verify that the implemented code passes the test.</div><div>- Action: Run the test again to ensure it passes with the implemented code.</div><div>- Example: Run the test for the sum function again, expecting it to pass.</div></div> <div>5. Refactor Code:</div> <div><div>- Objective: Improve the code without changing its functionality.</div><div>- Action: Refactor the code to improve readability, performance, or maintainability.</div><div>- Example: Simplify or optimize the code for the sum function.</div></div> <div>6. Repeat:</div> <div><div>- Objective: Repeat the process for each new functionality.</div><div>- Action: Write a new failing test, implement code, run the test, and refactor if needed.</div><div>- Example: Write tests and code for subtract, multiply, and divide functions.</div></div> <div></div>
---	--

5. 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.

- TDD: Best suited for ensuring code correctness and reliability.
- BDD: Ideal for projects with complex requirements and stakeholder involvement.
- FDD: Suitable for projects with well-defined features and milestone planning.

Aspect	Test-Driven Development (TDD)	Behavior-Driven Development (BDD)	Feature-Driven Development (FDD)
Approach	Tests written before code	Behavior described before code	Development based on features
Focus	Testing and code correctness	Behavior and system requirements	Feature design and delivery
Language	Tests written in code	Natural language specifications	Features defined in technical terms
Process	Test-first approach	Behavior specification first	Feature breakdown and delivery
Collaboration	Less emphasis on collaboration	Collaboration between teams	Collaboration between developers
Documentation	Tests serve as documentation	Behavior scenarios serve as documentation	Feature lists and design documents
Benefits	Early bug detection, improved code quality	Clear communication, business alignment	Structured development, early delivery
Applicability	Best for code correctness and reliability	Ideal for complex requirements and stakeholder involvement	Suitable for projects with well-defined features and milestone planning

Test-Driven Development (TDD)	Behavior-Driven Development (BDD)	Feature-Driven Development (FDD)
<ul style="list-style-type: none">● Approach: Write tests first.● Benefits: Early bug detection, and improved code quality.● Suitability: Best for code correctness.	<ul style="list-style-type: none">● Approach: Describe behavior before coding.● Benefits: Clear communication, and business alignment.	<ul style="list-style-type: none">● Approach: Develop based on features.● Benefits: Structured development, early delivery.● Suitability: Suitable for well-defined features.