**Assignment 1 : 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 theory**

**Test-Driven Development (TDD): Building Reliable Software on Solid Ground**

This infographic dives into Test-Driven Development (TDD), a software development approach that emphasizes writing tests before writing code, leading to more reliable and maintainable software.

**The TDD Cycle: A Continuous Loop for Quality**

1. **Red: Write Failing Tests First (Foundation)** (Icon: Red Stop Sign ())
   * Identify a small, specific feature or functionality.
   * Craft an automated test for that feature that **deliberately fails** (red signifies failing tests). This clarifies requirements and focuses your development efforts.
2. **Green: Make the Tests Pass (Building Blocks)** (Icon: Green Checkmark (✅))
   * Write the **minimal** amount of code necessary to make the failing test pass (green signifies passing tests). This ensures the code full fills its intended purpose.
3. **Refactor: Refine Your Code (Strengthening the Structure)** (Icon: Wrench ())
   * Improve the code's readability, maintainability, and overall design without breaking the functionality (ensure tests stay green). This promotes clean, well-structured code that's easier to understand and modify in the future.

**Benefits of Building with TDD: A Robust Structure**

* **Reduced Bugs:** Early and continuous testing catches errors early on, preventing them from becoming bigger problems later.
* **Improved Design:** Focus on testability leads to cleaner, more modular code with fewer dependencies.
* **Clearer Requirements:** Defining tests upfront clarifies what the code needs to achieve, improving communication between developers and stakeholders.
* **Increased Confidence:** Passing tests provide confidence that the code functions as expected.

**TDD: Fostering Software Reliability Theory**

* **Safety Net of Tests:** A suite of automated tests acts as a safety net, catching regressions when new code is introduced, helping to maintain a stable and reliable codebase.
* **Clear Communication:** Tests serve as clear documentation, specifying the expected behavior of the code and promoting understanding.
* **Focus on Functionality:** TDD keeps developers focused on what the code needs to do, ensuring it delivers the desired functionality.
* **Code :**
* import org.junit.Test  
  import kotlin.test.assertEquals  
    
  class AdderTest {  
   @Test  
   fun `test addition`() {  
   val result = 2.add(3)  
   assertEquals(5, result)  
   }  
  }

**In essence, TDD is a disciplined approach that builds reliable software one well-tested step at a time. By prioritizing tests first, you lay a solid foundation for robust and maintain.**