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.

Solution:

Test-driven development (TDD), also called test-driven design, is a method of implementing software programming that interlaces unit testing, programming and refactoring on source code.

Test-driven development was introduced as part of a larger software design paradigm known as Extreme Programming (XP), which is part of the Agile software development methodology.

Steps of the test-driven development approach

Before any new code is written, the programmer must first create a failing unit test. Then, the programmer -- or pair, or mob -- creates just enough code to satisfy that requirement. Once the test is passing, the programmer may refactor the design, making improvements without changing the behavior.

While TDD focuses on the programmer interactions at the unit level, there are other popular methods, such as acceptance-test-driven development (ATDD) or behavior-driven development (BDD), which focus on tests that can be understood by customers.

Test-driven development flow chart

These methods involve creating concrete examples as tests in collaboration between the technical staff and customer before the code is created, and then running the tests after the code is created to demonstrate the code is

implemented. Having the tests known upfront improves first-time quality. ATDD and BDD require developers, testers and the business side to collaborate to imagine and discuss the software and its implications before the code is created.

Advantages of TDD

Test-driven development can produce applications of high quality in less time than is possible with older methods. Proper implementation of TDD requires the developers and testers to accurately anticipate how the application and its features will be used in the real world.

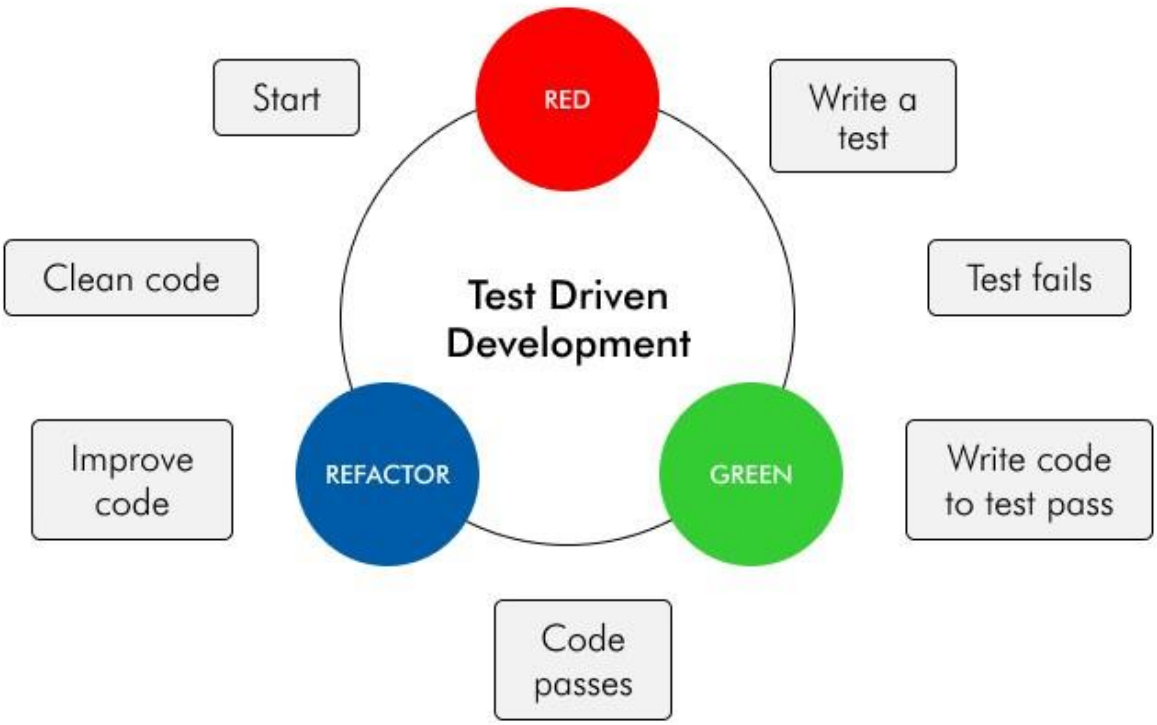
TDD creates a regression-test suite as a side effect that can minimize human manual testing, while finding problems earlier, leading to quicker fixes. The methodical nature of TDD ensures much higher coverage and first-time quality than classic phased code > test > fix > retest cycles. Because tests are conducted from the very beginning of the design cycle, time and money spent in debugging at later stages is minimized.

Disadvantages of TDD

TDD requires considerable skill to be successful, especially at the unit level. Many legacy systems are simply not created with unit testing in mind, making isolation of components in order to test impossible.

Further, many programmers lack the skills to isolate and create clean code. Everyone on the team needs to create and maintain the unit tests, or else they will quickly get out of date. And an organization looking at TDD will need to invest time -- to slow down a bit now in order to go faster later.

Finally, as with any method, the final results of TDD are only as good as the tests that have been used, the thoroughness with which they have been done and the extent to which they mimic conditions encountered by users of the final product.



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.

	TDD	BDD	FDD
1.Description	1.A development process where tests are written before code to ensure functionality. The cycle involves writing a failing test, writing code to pass the test, and then refactoring the code.	1.A methodology focusing on designing and building features. Each feature is a client-valued function that is developed in short iterations.	1. Enhances communication between technical and non-technical team members, aligning development with business needs.
2.Unique Approaches	Ensures code correctness through continuous testing and improvement.	Enhances communication between technical and non-technical team members, aligning development with business needs.	Focuses on delivering client-valued features in a systematic, scheduled manner.
3.Benefits	Early bug detection, robust codebase, improved design.	Clear requirements, shared understanding, better user satisfaction.	Predictable schedules, incremental progress, focused on client needs.
4.Suitability for Different Contexts	Best for projects where code quality and reliability are critical.	Ideal for projects requiring strong collaboration between developers and non-technical stakeholders.	Suited for large, complex projects that can be broken down into well-defined features.