UNIT IV –LIFE CYCLE BASED TESTING

Life Cycle Based Traditional Waterfall Testing, Testing in Iterative Life Cycles, Agile Testing, Agile Model–Driven Development - Model-Based testing: Testing Based on Models - Integration Testing: Decomposition-Based Integration, Call Graph–Based Integration.

Driven development

Driven development refers to software engineering methodologies where a specific artifact, constraint, or perspective dictates the design and writing of production code.

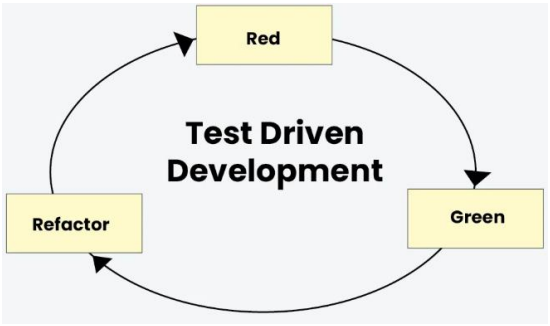
Test-Driven Development (TDD): Developers write automated unit tests before writing the actual production code, following a "Red-Green-Refactor" cycle to ensure code correctness and modular design.

Behavior-Driven Development (BDD): An extension of TDD that involves non-technical stakeholders (product owners, business analysts) to define system behavior in a human-readable, plain-English "Given-When-Then" format.

Feature-Driven Development (FDD): An agile framework that organizes development around client-valued, small features delivered in short, two-week iterations with distinct design and building phases.

Data-Driven Development (DDD): Focuses on using analytics, user metrics, and data collection pipelines to dictate new features, optimizations, and product direction.

Test-Driven Development (TDD)



Test-Driven Development (TDD) is a Software development method in which you write Automation Tests before the actual development process starts, which is coding. Here we are learning TDD in detail with these important points related to the same.

Process of Test Driven Development (TDD)

It is the process in which Test Cases are written before the code that validates those cases. It depends on the repetition of a concise development cycle. Test-driven Development is a technique in which automated Unit tests are used to drive the design and free decoupling of dependencies.

The process of Test-Driven Development (TDD) follows a repetitive cycle called Red-Green-Refactor.

Run all the test cases and make sure that the new test case fails.

- ❖ Red - Create a test case and make it fail, Run the test cases
- ❖ Green - Make the test case pass by any means.
- ❖ Refactor - Change the code to remove duplicate/redundancy and Refactor code - This is done to remove duplication of code.

Once you complete the Red-Green-Refactor cycle, you continue repeating the process for the next piece of functionality or unit of code. Every time you write a new test, your code gets better and more reliable, making the overall software stronger.

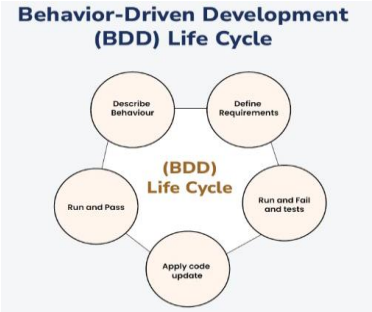
Behavior-Driven Development (BDD)

BDD is an improved approach based on Test-Driven Development (TDD). It focuses on bringing developers, testers, customers, and other stakeholders together to decide how a system should behave before the actual development starts.

In BDD, requirements are written in simple and understandable language, making them easy for both technical and non-technical people to understand.

Importance of BDD

- Improves collaboration: BDD helps team members work together and develop a common understanding of the requirements.
- Faster development: Continuous feedback allows problems to be identified and changes to be made quickly.
- Improves software quality: Well-defined requirements help prevent errors and make the software easier to maintain.
- Reduces risks: Clear communication between stakeholders helps avoid misunderstandings and unnecessary development work.
- Clear expectations: Everyone knows what the system is expected to do before development begins.

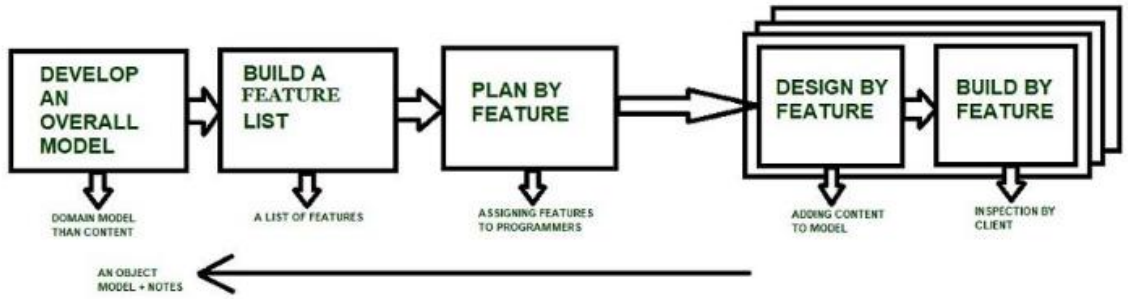


FDD

FDD stands for Feature-Driven Development. It is an agile software development approach that follows an iterative and incremental process. It focuses on developing the software step by step by completing individual features.

The main goal of FDD is to deliver useful and working software to the client on time. The project is divided into smaller features, which are developed and delivered gradually.

In FDD, regular reporting and monitoring of progress are important at every stage of development. This helps the team track the work, identify problems, and ensure that the project is moving according to plan.



Domain-Driven Design (DDD)

Domain-Driven Design (DDD) is a software development approach that focuses on understanding the main business area or problem for which the software is being created. It involves working closely with business experts so that the software model correctly represents real-world business activities and requirements.

Domain: The particular business or problem area that the software is designed to handle, such as banking, online shopping, or healthcare.

Driven: The development and design of the software are mainly guided by business requirements and knowledge of the domain, not just by technical factors.

Design: Refers to planning the system's structure, components, and how different parts interact to satisfy both business and technical requirements.

Model-Based Testing (MBT)

Model-Based Testing (MBT) is a software testing technique in which test cases are created using models that describe how a system should behave. It helps testers achieve better test coverage, maintain consistency, and reduce the amount of manual testing work.

Key Points

- Uses different models: Models such as Finite State Machines (FSM), UML diagrams, and Statecharts can be used to represent system behavior.
- Generates test cases: Test cases can be created automatically or with some manual assistance from the models.
- Better test coverage: It helps test more possible system behaviors and scenarios.
- Less manual effort: Automated test generation reduces the time and effort required to create test cases manually.

