Automation Testing

TDD Basics

#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
cout << "Hello, world!" << endl;
return 0;
}

A Day in the Life of an Automation Test Engineer

Anna now understands what automation testing entails. So she's chosen TDD as the method for her project.

She needs to understand the foundations of TDD, how it works, and how to apply TDD to agile to complete her project.

To achieve the above, she will learn a few concepts in this lesson that can help her find a solution for the scenario.

Learning Objectives

By the end of this lesson, you will be able to:

- Illustrate TDD
- Understand the basics of TDD
- Configure the execution flow of TDD
- Understand TDD in agile

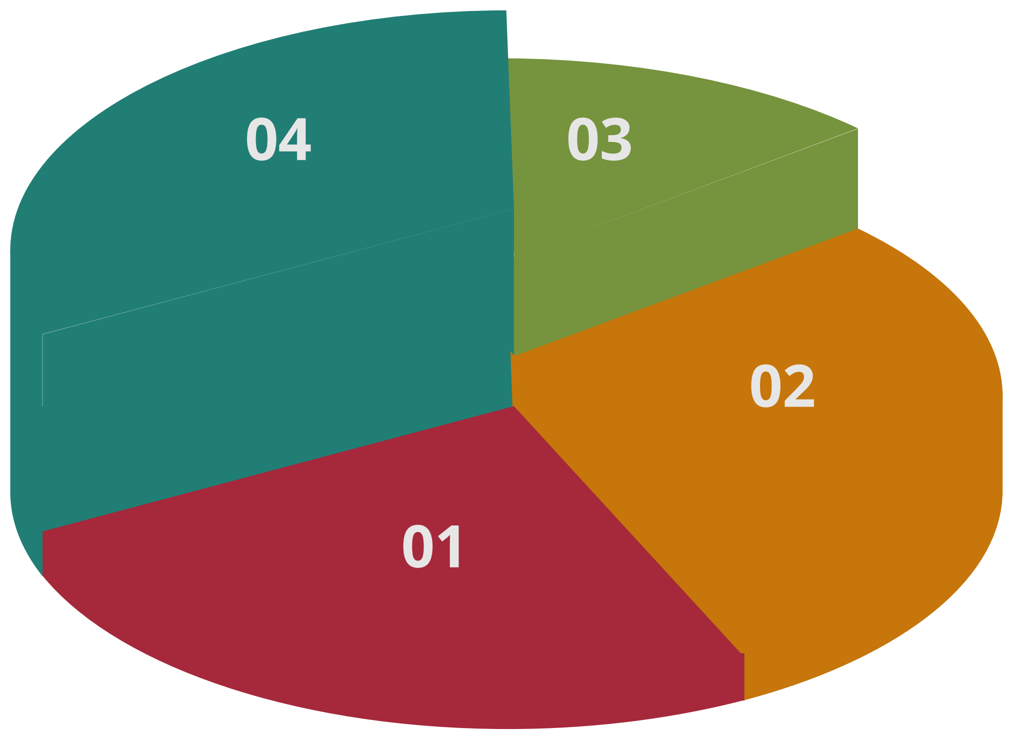
TDD: Overview

Introduction to TDD

Test Driven Development (TDD) is a software development methodology in which test cases are created to specify and validate what the code will accomplish.

Designing and writing tests for each small functionality of an application is the first step in TDD.

Why Is TDD Used?

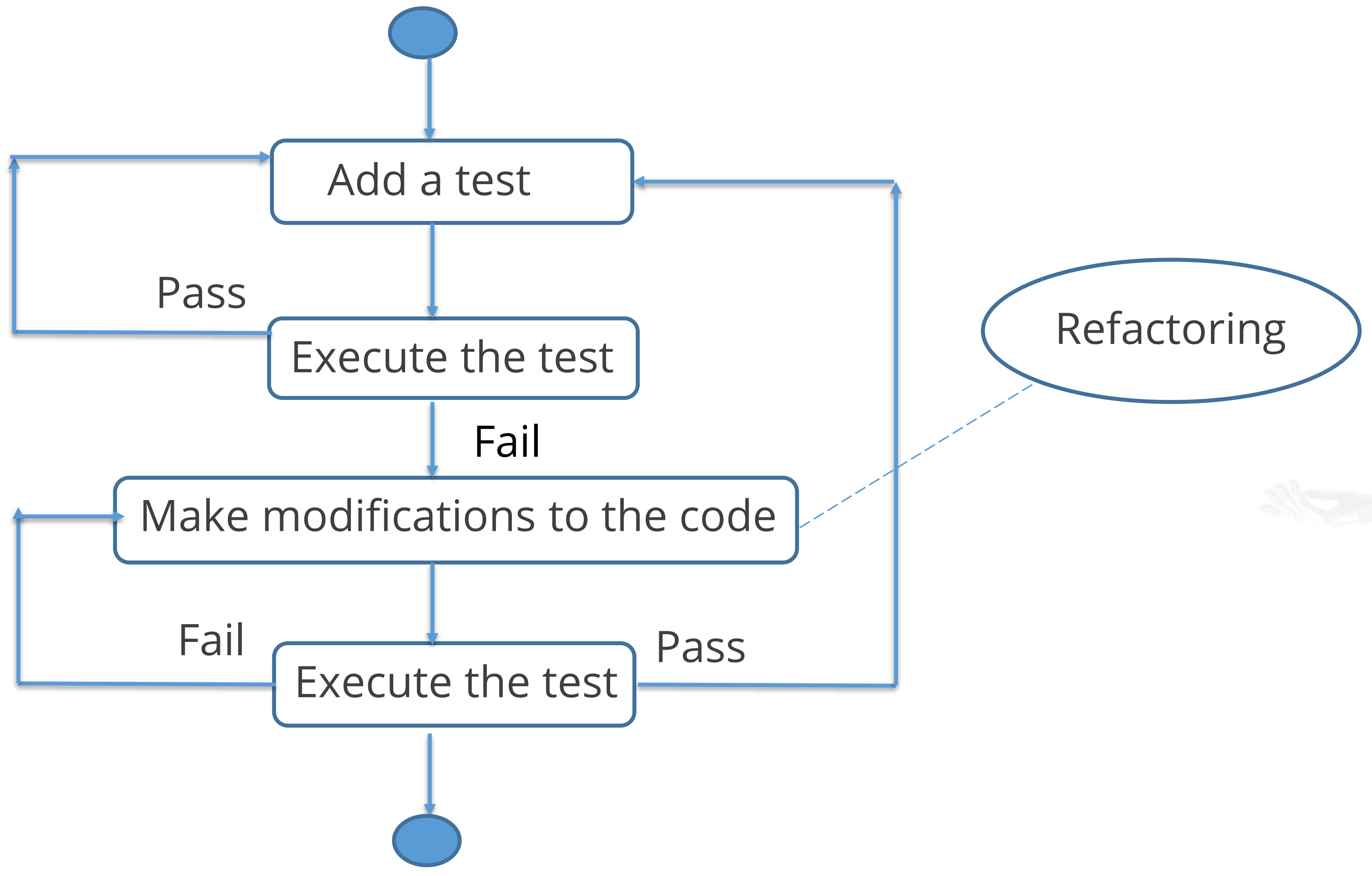


- It allows for faster innovation and continuous delivery because the code is robust.
- It makes user code more adaptable and extensible.
- The code generated is easy to test by design.
- The code can be refactored or transferred without breaking it.

TDD Execution Flow

How to Perform a TDD?

The procedure below outlines how to execute a TDD:



If pass, then development stops

Levels of TDD

TDD is divided into two categories:

The first one is an Acceptance TDD (ATDD).
Users write a single acceptance test with ATDD. This test fulfills the specifications, requirements or confirms the system's behavior.

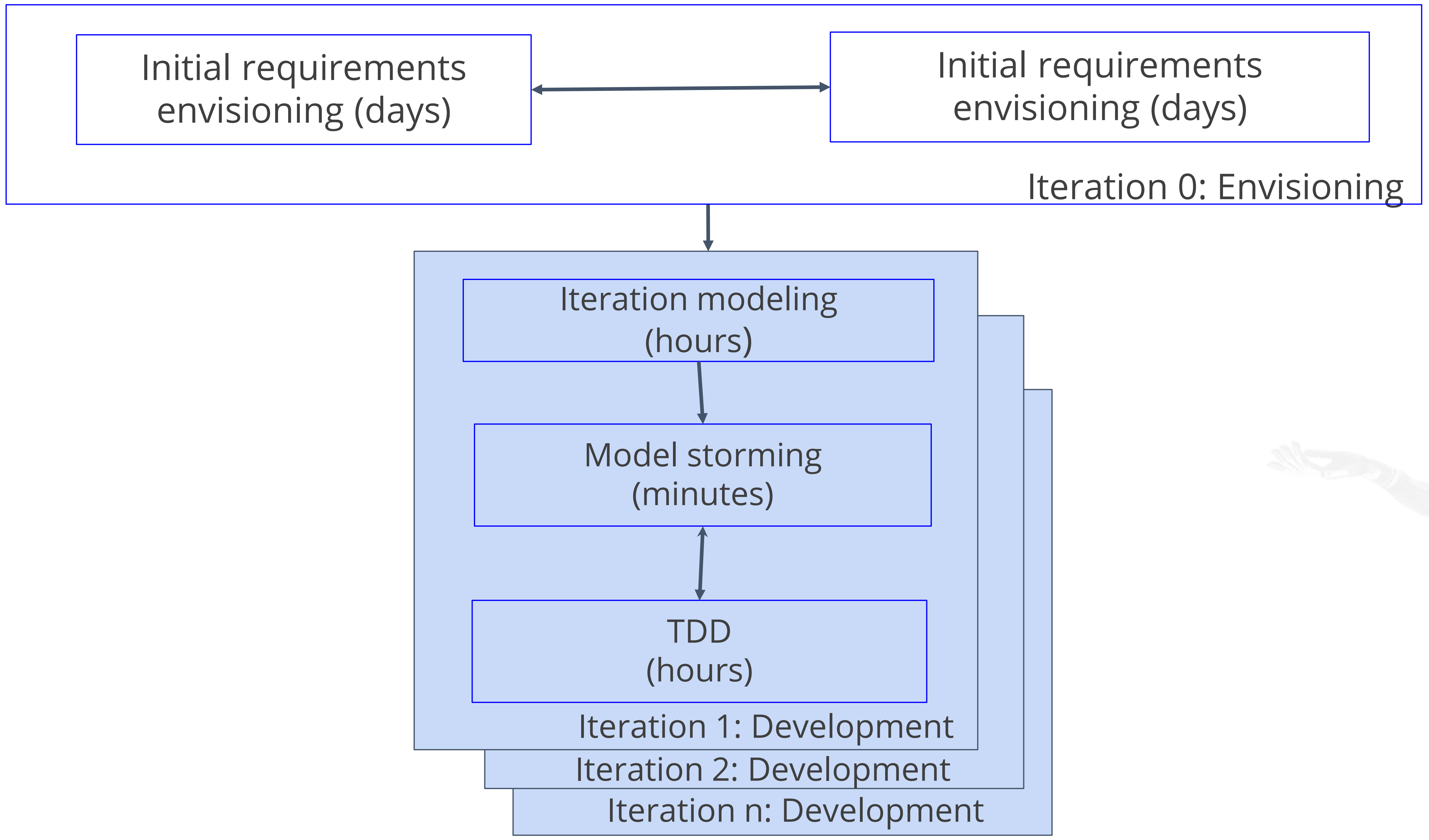
Levels of TDD

The second one is a Developer TDD.
Developer TDD entails writing a single developer, test, such as a unit test, and then writing just enough production code to pass that test.

FULL STACK

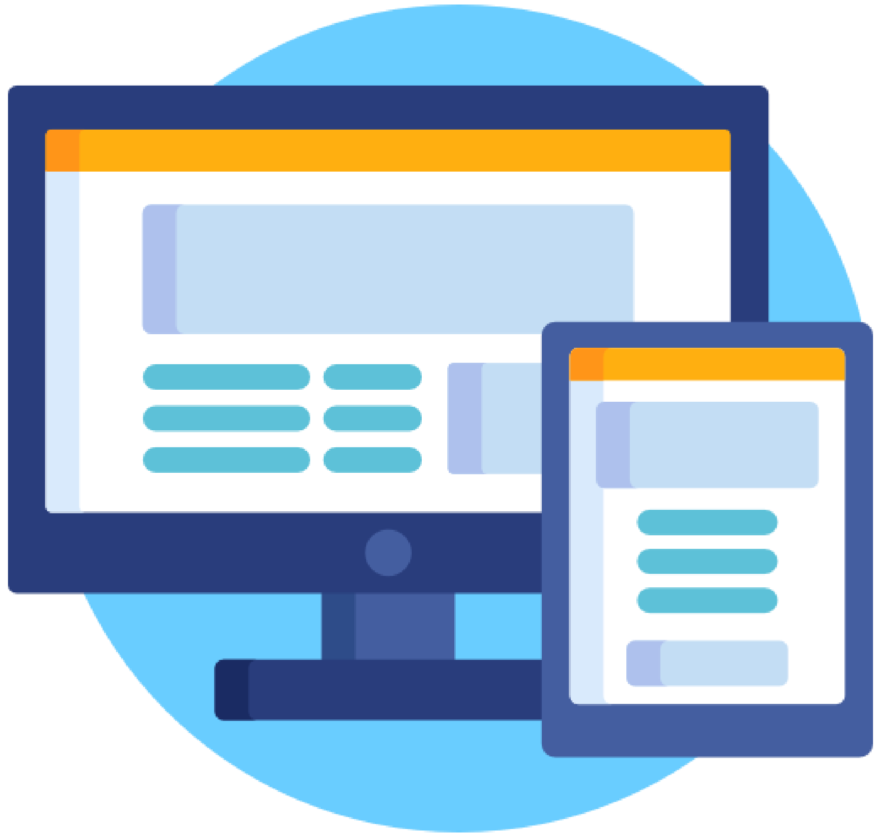
TDD in Agile Development

Lifecycle of AMDD



Iteration 0: Envisioning

There are two main sub-activates.



Initial requirements envisioning

The primary goal is to investigate the usage model, the initial domain model, and the user interface model (UI).

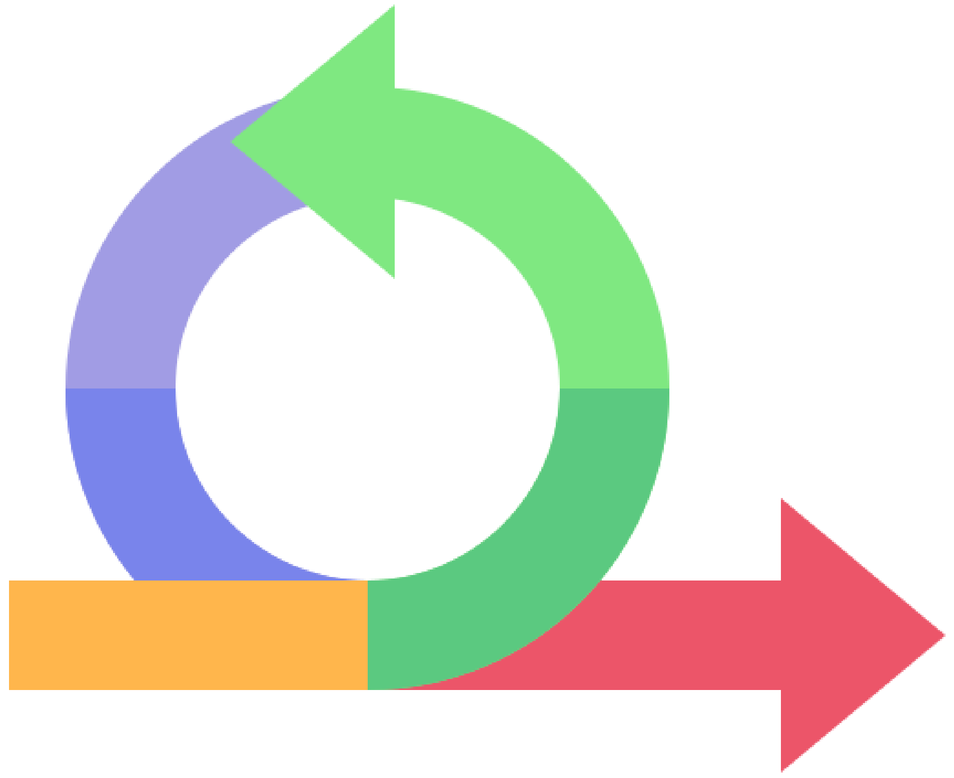
Iteration 0: Envisioning

Initial architectural envisioning

Exploration of technology diagrams, user interface (UI) flow, domain models, and change scenarios is the primary focus.

Iteration Model

The work that will be done for each iteration must be planned here by the team.

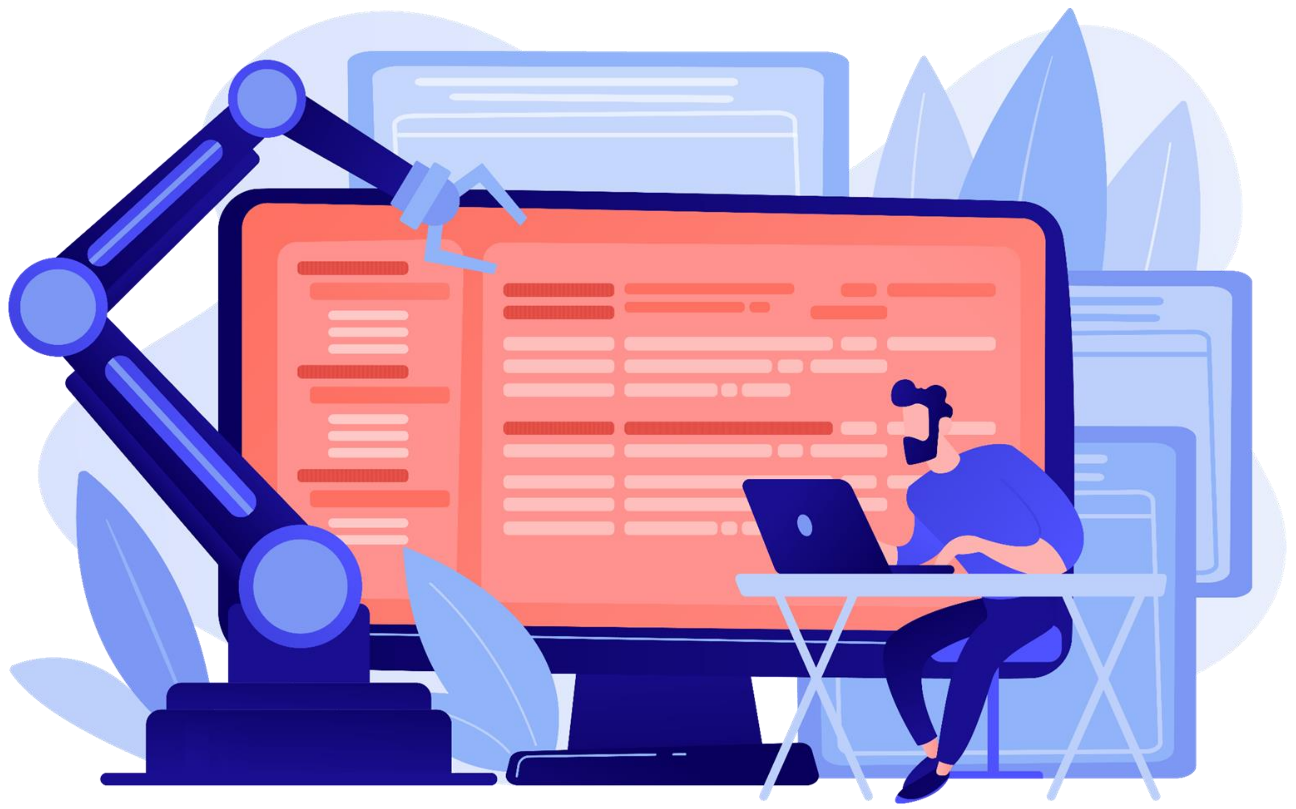


Model Storming

Just-in-time modeling is another name for model storming.
In this modeling session, a group of two to three people analyzes challenges on paper or a whiteboard.

Test Driven Development (TDD)

TDD simplifies and clarifies the code. It enables the developer to keep less documentation.



Review

Code inspections and model evaluations are included in the review.

Difference between TDD and AMDD

TDD	AMDD
TDD reduces the length of the programming feedback loop.	AMDD reduces the length of the modeling feedback loop.
The term TDD refers to a comprehensive specification.	AMDD is effective in dealing with more serious difficulties.
TDD encourages the creation of high-quality code.	AMDD encourages developers and stakeholders to communicate effectively.
TDD is a non-visually oriented.	AMDD is a visually oriented.
The scope of TDD is restricted to software development.	AMDD covers a wide range of issues, including stakeholders. It entails attempting to reach a common understanding.
Programmers are addressed by TDD.	AMDD consults with business analysts, stakeholders, and data experts.

Example of TDD

Users will define a class password in this test driven development example. They can attempt to meet the following criteria.

Password condition:
The password should be five to ten characters long.

Example of TDD

```
Package Temp;

import org.testng.Assert;
import org.testng.annotations.Test;

public class TestPW {
@Test
      public void TestPWLen() {
      PasswordValid pw=new PasswordValid
();
      Assert.AssertEquals(true,
pw.isValid("P1Q2R3"));
      }
}
```

Benefits of TDD

Advantages of TDD

01 Early bug detection

02 Code that is better designed, clearer, and more adaptable.

03 Refactor

04 Beneficial to developers

Key Takeaways

- Each functionality test case are built and tested first, and if the test fails, a new code is produced to pass the test, keeping the code simple and bug-free.
- When the code is regularly improved, TDD produces the best outcomes.
- TDD encourages the creation of high-quality code.
- TDD shortens the programming feedback loop.