Learning Consolidation

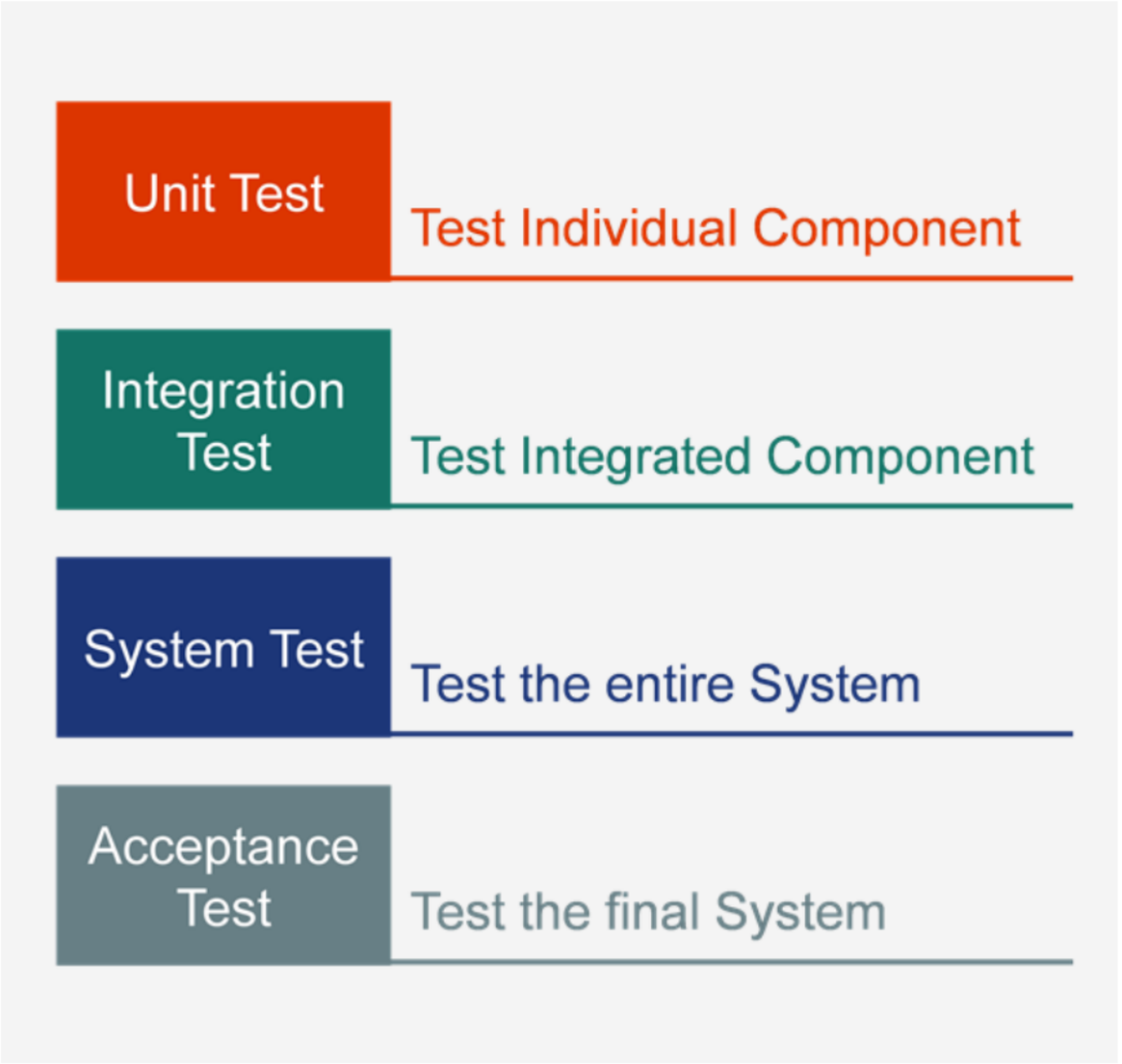
Implement Unit Testing For Angular Components and Services

In this Sprint, you learned to:

- Explain the importance of Unit Testing
- Create an Angular Test app
- Explore the configuration files generated by Angular CLI for Jasmine framework and Karma test runner
- Test Angular Components using Jasmine framework and Karma test runner
- Use Jasmine spies to test dependencies of Angular components

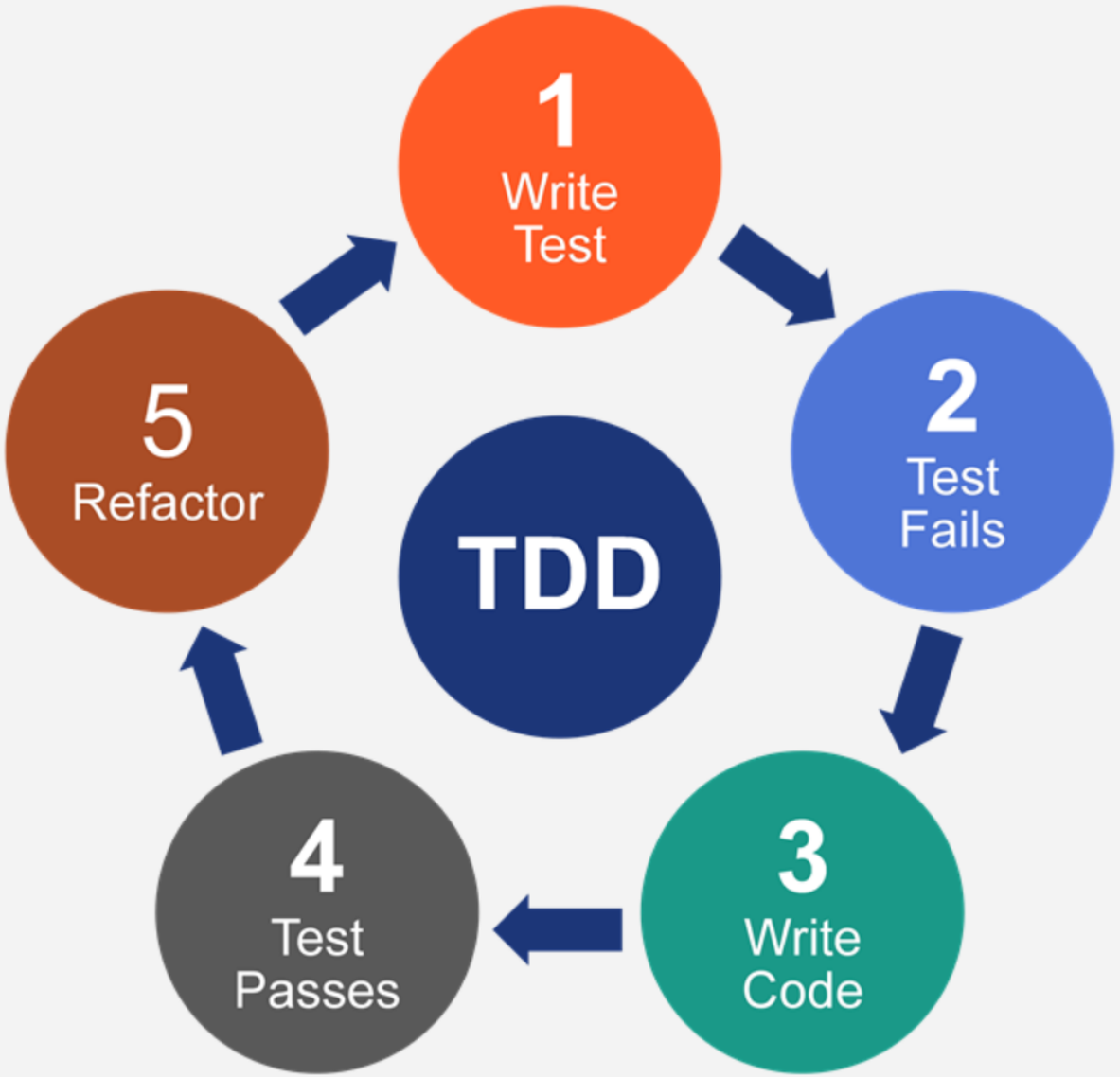
Levels Of Testing

- A level of software testing where every unit or component of a software or system is tested.
- The levels of testing makes testing more efficient and make it easier to find all possible test cases at a given level.



Think and Tell

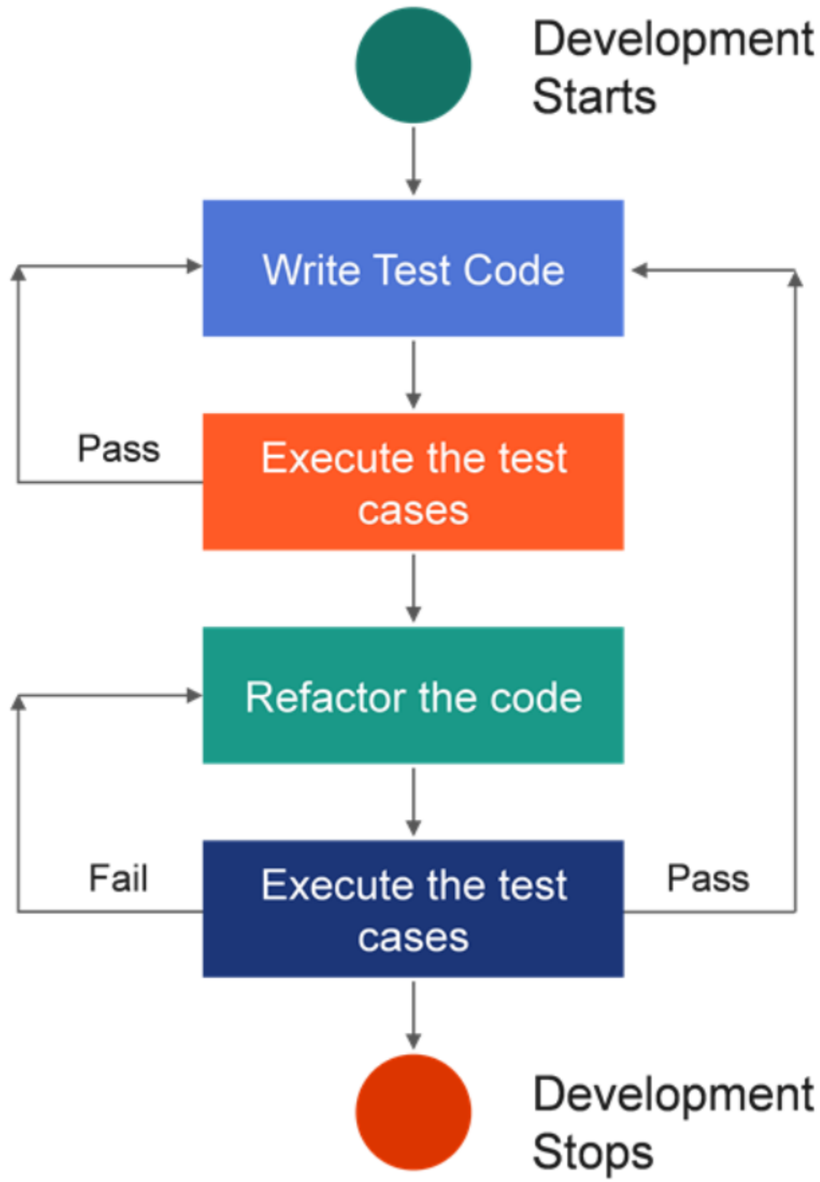
- Software development approaches have slowly moved toward an agile model in the past decades.
- The rapid changes within the software development process lead towards continuous delivery and faster speed to production environment.
- Then, how can a quality product be delivered without skipping the scheduled deadlines?
- Wouldn't it be great if developers could concentrate on a single feature at a time, receive feedback more quickly, and be encouraged to write solid, clean code?

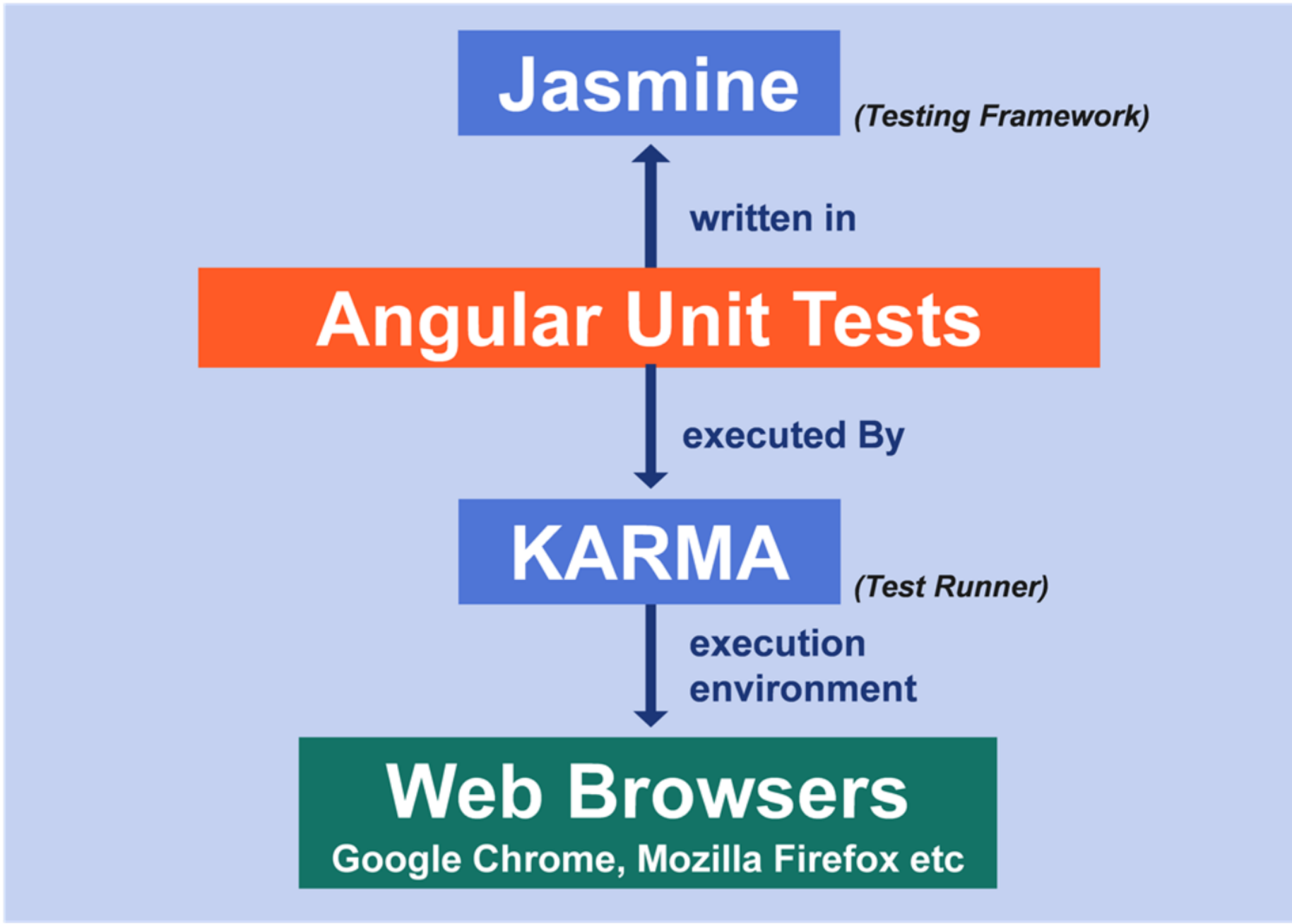


Test-Driven Development (TDD)

- Test-driven development (TDD) is a software development approach that focuses on writing unit test cases before developing the actual code.
- It uses a repeated approach that combines programming, the creation of unit tests, and refactoring.
- This approach is also called the Test First Approach.

TDD Approach




Jasmine
(Testing Framework)
written in
Angular Unit Tests
executed By
KARMA
(Test Runner)
execution
environment
Web Browsers
Google Chrome, Mozilla Firefox etc

Karma Test Runner

- Karma is a node-based test tool that allows you to test your JavaScript codes across multiple real browsers.
- It is a tool developed by Angular team that makes our test-driven development fast, fun and easy.
- It is technically termed as a test runner.
- As a test runner, Karma does three important things:

1. It starts a web server and serves your JavaScript source and test files on that server.
2. Loads all the source and test files in the correct order.
3. Finally, it spins up browsers to run the tests.

Setup Testing in Angular

- When an Angular project is created using the Angular CLI, it downloads and installs everything that is required to test the Angular application using Jasmine and Karma.
- Angular CLI creates a Jasmine spec file along with the main code file whenever you create a file using a CLI command.
- The spec file will have the same name as the main code file but will end with **.spec.ts**

  Example: `ng generate component home` will create

  `home.component.ts` and

  `home.component.spec.ts`

- Giving the **"`ng test`"** CLI command builds the application in watch mode and launches the Karma test runner.
- Test results are displayed in the console, and a Chrome browser opens and displays the test output in the "Karma Jasmine HTML Reporter"

Test Entry File

The Angular-CLI configuration of Karma uses the file “test.ts” as the entry point of the tests for the application.

This file contains the following:

- An environment to run angular tests is being created using all the imports at the beginning of the file.
- TestBed which is a powerful unit testing tool provided by angular is initialized in this file.
- Finally, Karma loads all the test files of the application matching their names against a regular expression. All files inside our app folder that has “spec.ts” on its name are considered as a test.

```
import 'zone.js/testing';
import { getTestBed }from 'angular/core/testing';
import {
  BrowserDynamicTestingModule,
  platformBrowserDynamicTesting
} from '@angular/platform-browser-
dynamic/testing';
declare const require: {
  context(path: string, deep?: boolean, filter?:
RegExp): {
    keys(): string[];
    <T>(id: string): T;
  };
};
// First, initialize the Angular testing
environment.
getTestBed().initTestEnvironment(
  BrowserDynamicTestingModule,
  platformBrowserDynamicTesting()
);
// Then we find all the tests.
const context = require.context('./',true,
/\.spec\.ts$/);
// And load the modules.
context.keys().forEach(context);
```

app.component.spec.ts

```
import { TestBed } from '@angular/core/testing';
import { AppComponent } from './app.component';
describe('AppComponent', () => {
  beforeEach(async () => {
    await TestBed.configureTestingModule({
    declarations: [
    AppComponent
    ],
    }).compileComponents();
  });
  it('should create the app', () => {
    const fixture
= TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    expect(app).toBeTruthy();
  });
  it(`should have as title 'unit-testing'`, () => {
    const fixture =
TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    expect(app.title).toEqual('unit-testing');
  });
});
```

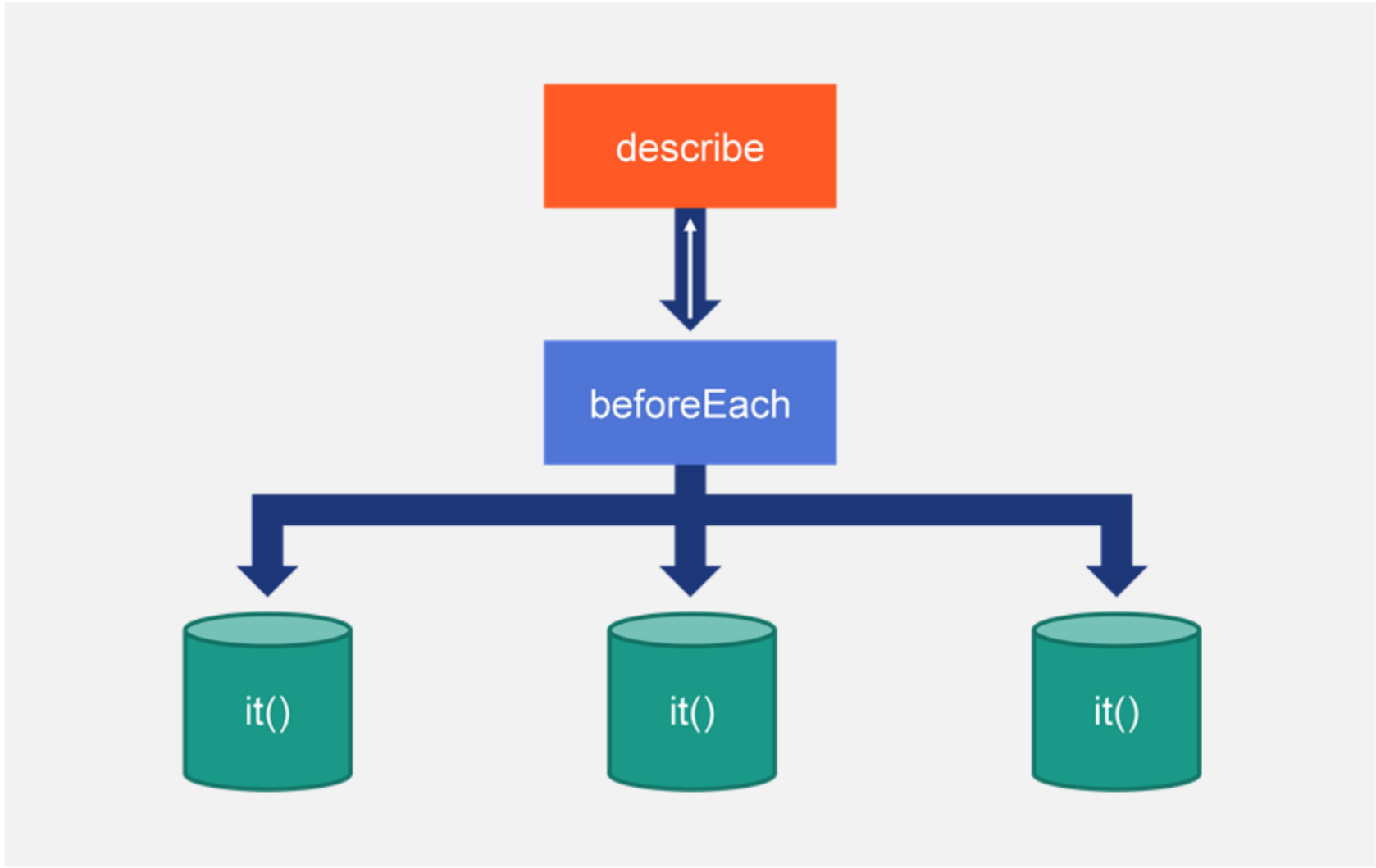
CLI Generated Test File

Configuring the TestBed by declaring the component to test

Creates an instance of the AppComponent, adds a corresponding element to the test-runner DOM, and returns a ComponentFixture

Asserts whether the title of the app is 'unit-testing'

Flow of Running Test



Built-in Matchers

- expect().toBe(expected)
- expect().toBeDefined()
- expect().toBeFalse()
- expect().toBeFalsy()
- expect().toBeGreaterThan(expected)
- expect().toBeGreaterThanOrEqual(expected)
- expect().toBeInstanceOf(expected)
- expect().toBeLessThan(expected)
- expect().toBeLessThanOrEqual(expected)
- expect().toBeNaN()
- expect().toBeNegativeInfinity()
- expect().toBeNull()
- expect().toBePositiveInfinity()
- expect().toBeTrue()
- expect().toBeTruthy()
- expect().toBeUndefined()
- expect().nothing()
- expect().toContain(expected)
- expect().toEqual(expected)
- expect().toHaveBeenCalled()
- expect().toHaveBeenCalledBefore(expected)
- expect().toHaveBeenCalledOnceWith()
- expect().toHaveBeenCalledTimes(expected)
- expect().toHaveBeenCalledWith()
- expect().toHaveClass(expected)
- expect().toHaveSize(expected)
- expect().toMatch(expected)
- expect().toThrow(expectedopt)
- expect().toThrowError(expectedopt, messageopt)
- expect().toThrowMatching(predicate)
- expect().withContext(message)

Faking Functions Using Jasmine Spies

- Jasmine provides patterns called **"spies"** to create fake implementations for replacing a function dependency.
- A spy is a function that records its calls.
- It records the function arguments, which can be later used to assert that the spy has been called with input values.
- A spy can have a return value that will be the same regardless of the input parameters.

```
let fixture =
TestBed.createComponent(AppComponent);
let app = fixture.componentInstance;
let calc = app.calc;
let addSpy = spyOn(calc,
'add').and.returnValue(40);
expect(app.add(1, 2)).toEqual(40);
expect(addSpy).toHaveBeenCalled();
```

Self-Check

Which of the following TestBed methods is used to create an Angular component under test?

a) configureComponent
b) createTestComponent
c) configureTestingComponent
d) CreateComponent

Explanation:
Option 1, 2 and 3 are wrong because there is no such Test method

Self-Check: Solution

Which of the following TestBed methods is used to create an Angular component under test?

a) configureComponent
b) createTestComponent
c) configureTestingComponent
d) **CreateComponent**

Explanation:
Option 1, 2 and 3 are wrong because there is no such Test method

Self-Check

Which file provides the options and plugins to configure Karma and Jasmine?

a) angular.json
b) karma.conf.js
c) test.js
d) package.json

Self-Check: Solution

Which file provides the options and plugins to configure Karma and Jasmine?

a) angular.json
b) **karma.conf.js**
c) test.js
d) package.json