Slide Note

Is it important to test a website before launching it?

Slide Note

From the above instances, it is clear that websites are also soft products that need to undergo thorough testing and even a small breach can lead to huge business fiasco.

What Can Happen If a Website Malfunctions?

Some of Amazon's third-party retailers saw their product prices reduced to 1 penny due to a software glitch. They were left with heavy losses.

Starbucks was forced to close about 60 percent of its stores in the U.S. and Canada due to a software failure in its POS system. At one point, the store served coffee for free as they were unable to process the transaction.

Nissan recalled over 1 million cars from the market due to a software failure in the airbag sensory detector. Two accidents were reported due to this software failure.

In April 2015, the Bloomberg terminal in London crashed due to a software glitch and affected more than 300,000 traders in the financial markets. It forced the government to postpone a 3-billion-pound debt sale.

Even a small breach in website malfunction can lead to a huge business loss.

Slide Note

Think and Tell

- How can we enhance the standard of a software product?
- How can we create an ideal bug-free software product for the end users?
- Is there a way to identify the condition of the product and the work standards for the product developed?
- How can we improve the trust and satisfaction of customers by providing a quality product?

Slide Note

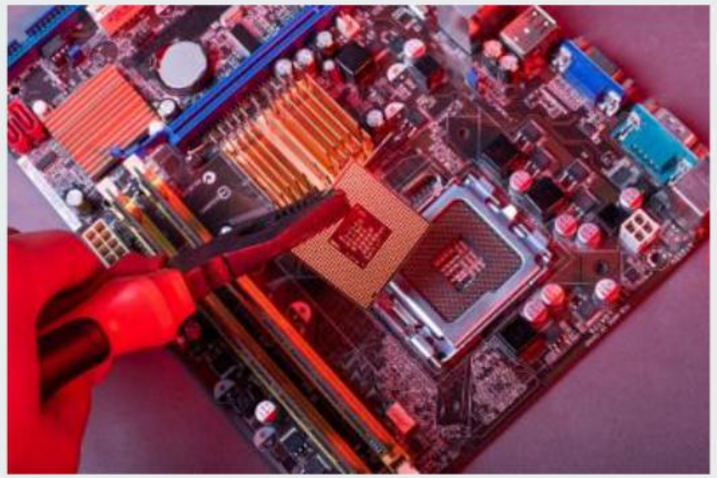
Software Testing and Its Importance

Identifies Defects Early	Improves Product Quality	Increases Customer Trust and Satisfaction	Detects Security Vulnerabilities	Saves Money
Software testing identifies any issues and defects with the written code and the software product can be delivered after fixing it.	Software testing ensure the product meet the criteria and specifications defined by the end users and hence pass the quality assurance.	Testing a product builds customer trust and satisfaction as it provides product's strong and week points. It ensures to deliver a quality product after testing it multiple times.	Software testing prevents a web application to fall victim to a cross-site scripting attack where the attackers try to inject malicious code into the user's web browser.	It is more difficult to trace and resolve issues after the application is launched and it more expensive also. Software testing detects bugs at early stage and hence saves a lot of money to the organization.

Slide Note

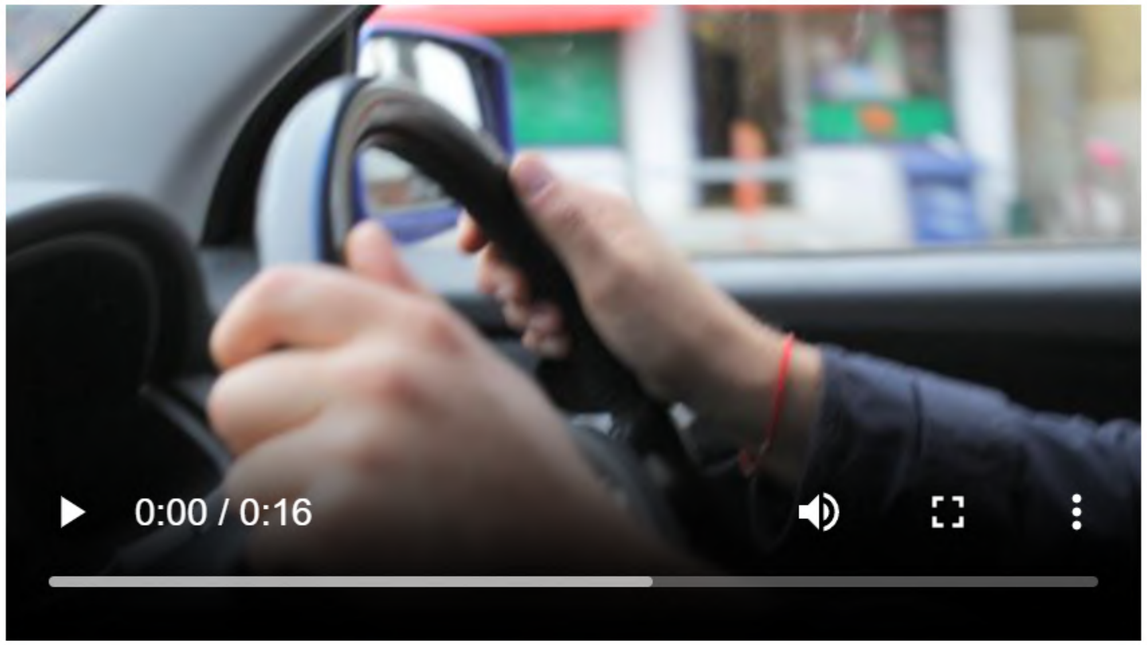
Think and Tell

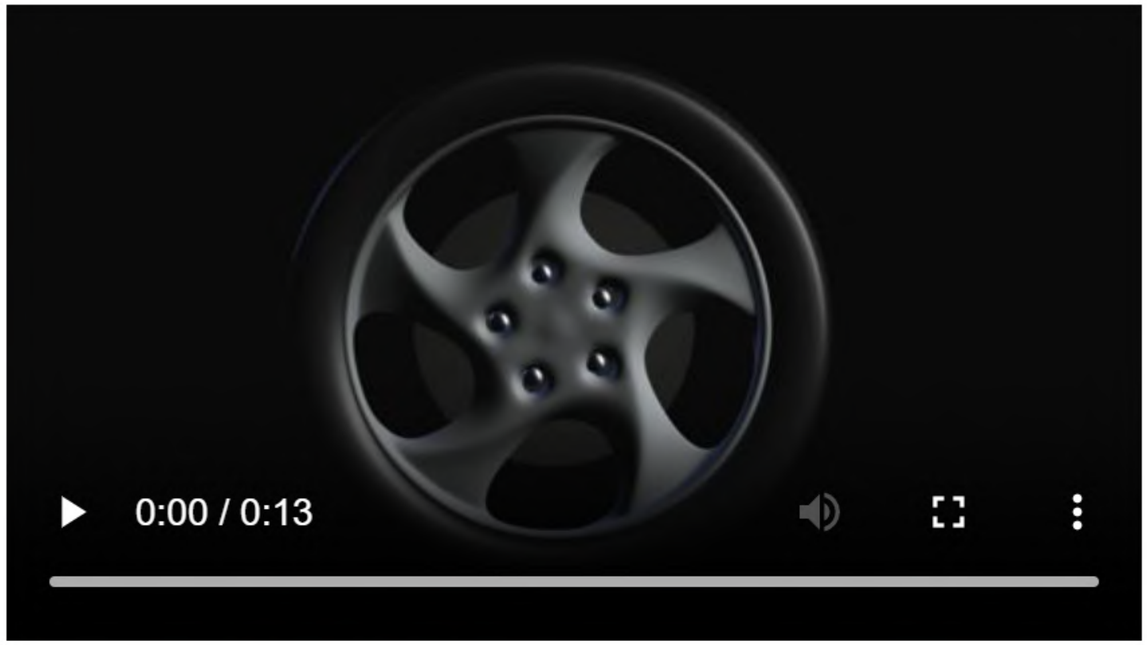
- Consider a scenario where a personal computer (PC) is assembled with its core components.
- Is testing the PC as an entire product sufficient?
- Say a single component is malfunctioning, and because of that, the whole product is considered defective.
- Is it not required to test individual components to demonstrate that they satisfy their requirements so that they ensure a final quality product?
- Don't you think that the core components shipped by the component manufacturer should be tested individually so that they provide a quality product to the assembler?

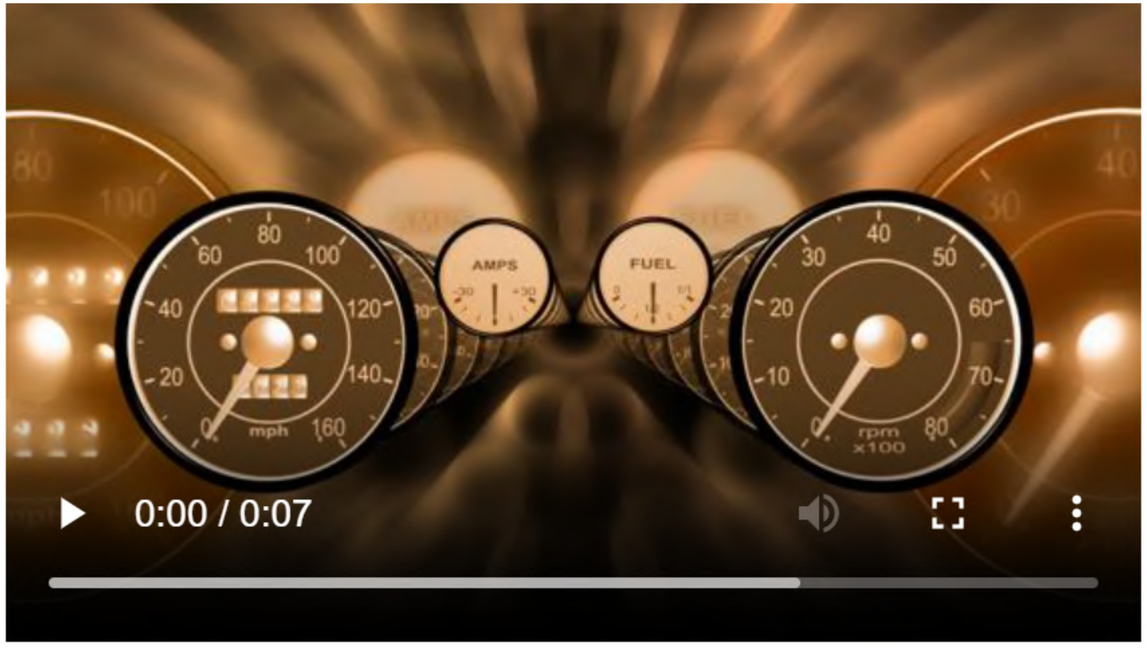


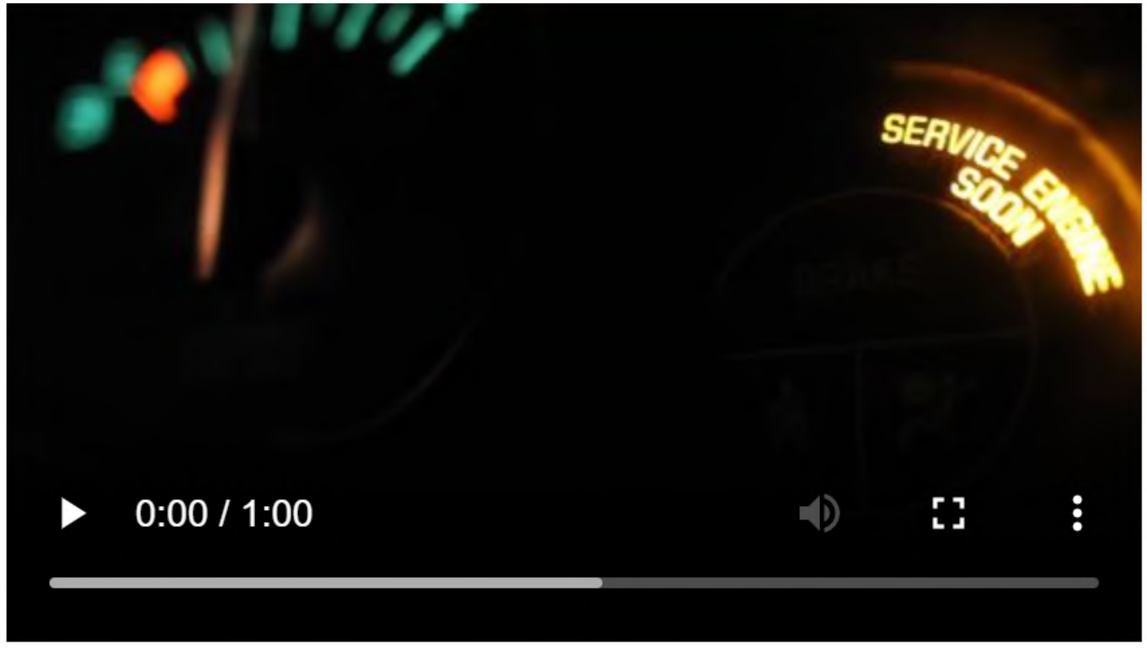
Slide Note

Let's observe another example where each unit of a car is tested individually


0:00 / 0:16


0:00 / 0:13


AMPS
FUEL
mph
rpm
x100
0:00 / 0:07


SERVICE ENGINE SOON
0:00 / 1:00

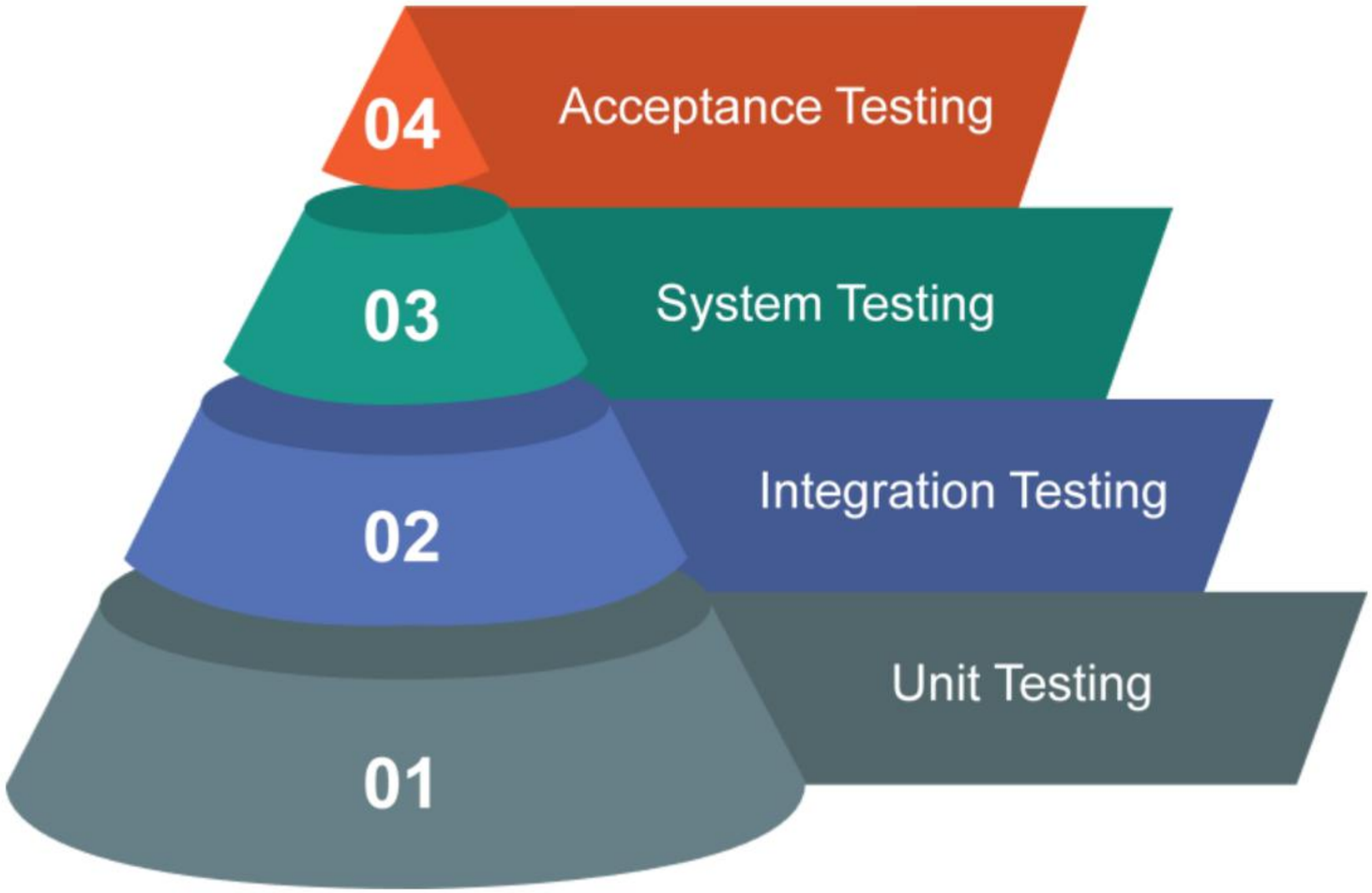
Implement Unit Testing For Angular Components and Services

Learning Objectives

- 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 is a process 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.

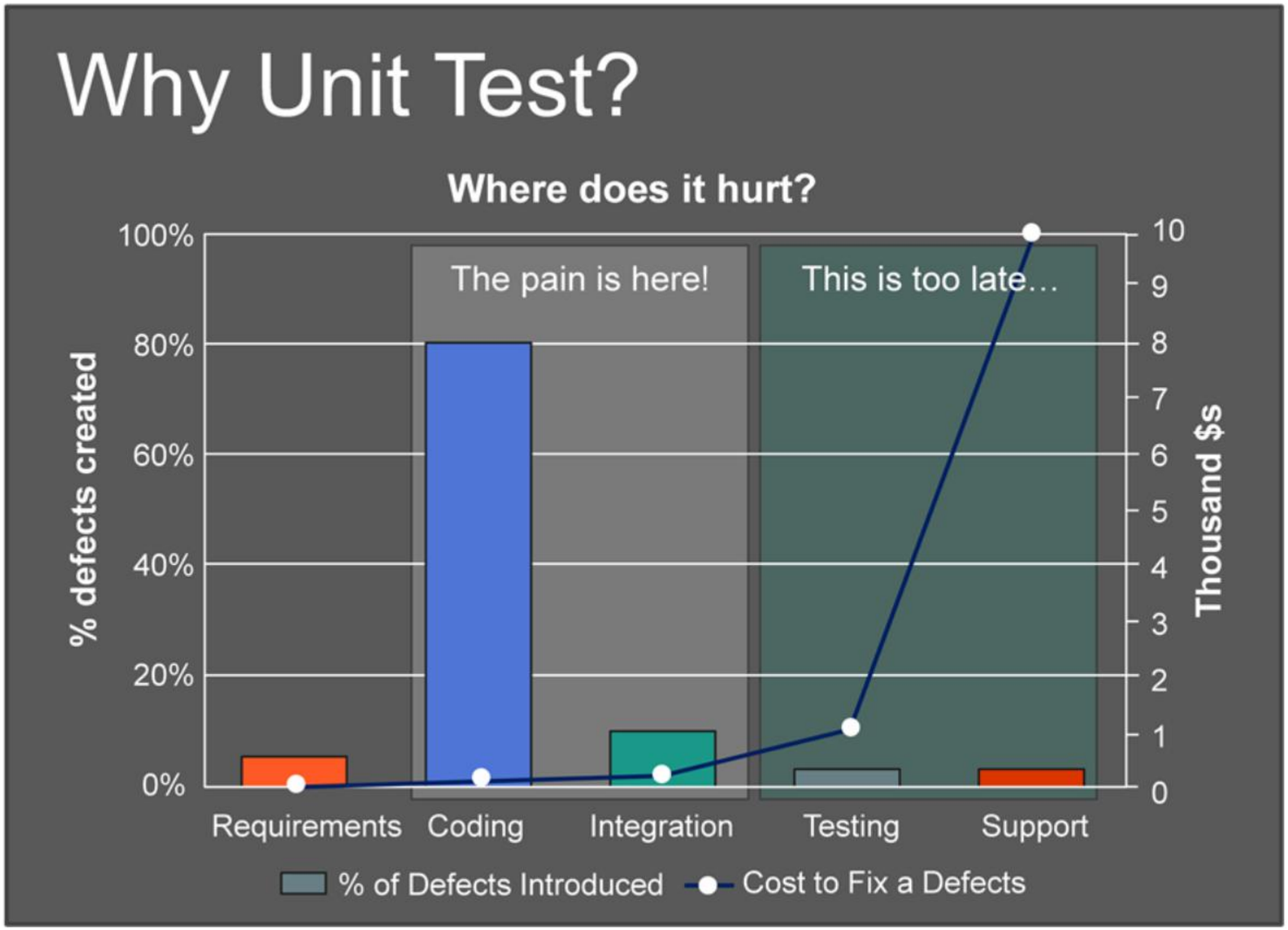


Unit Testing

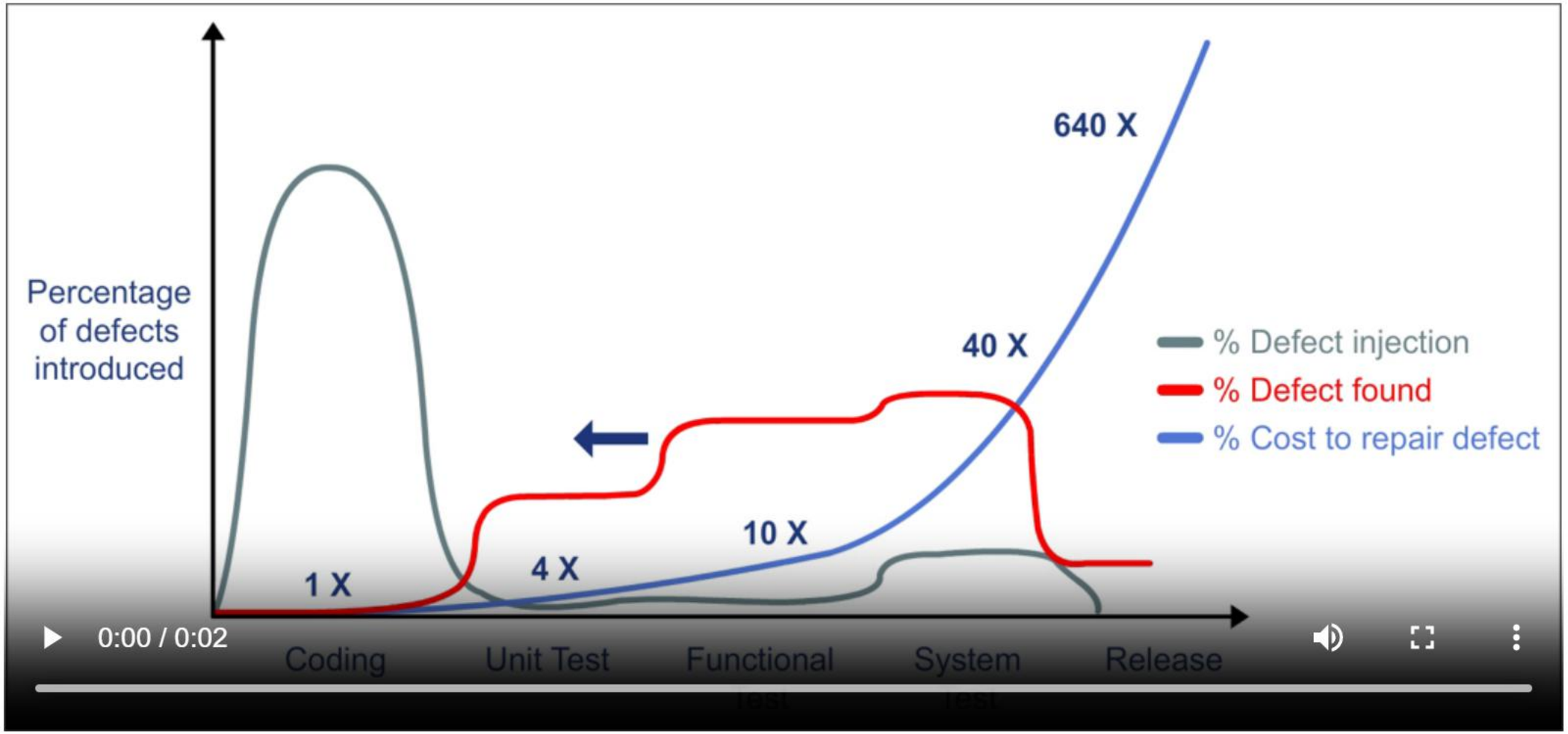
- Unit testing is a method where a single unit, component, or module of software is tested individually.
- A single unit is a block of code that has only one responsibility. For example, a class or a function.
- Unit testing enables you to add new features without breaking other parts of the application.
- It enables the development team to validate the correctness of the developed code, reuse it, and make changes to the code more quickly.

Benefits of Unit Testing

- Isolates a section of code and validates its correctness
- Fixes the bugs at an early stage
- Reduces the cost as bugs are resolved at the earliest
- Improves the design by allowing refactoring of the code
- Assures in achieving a high-quality product
- Simplifies the debugging process



Early Detection of Errors

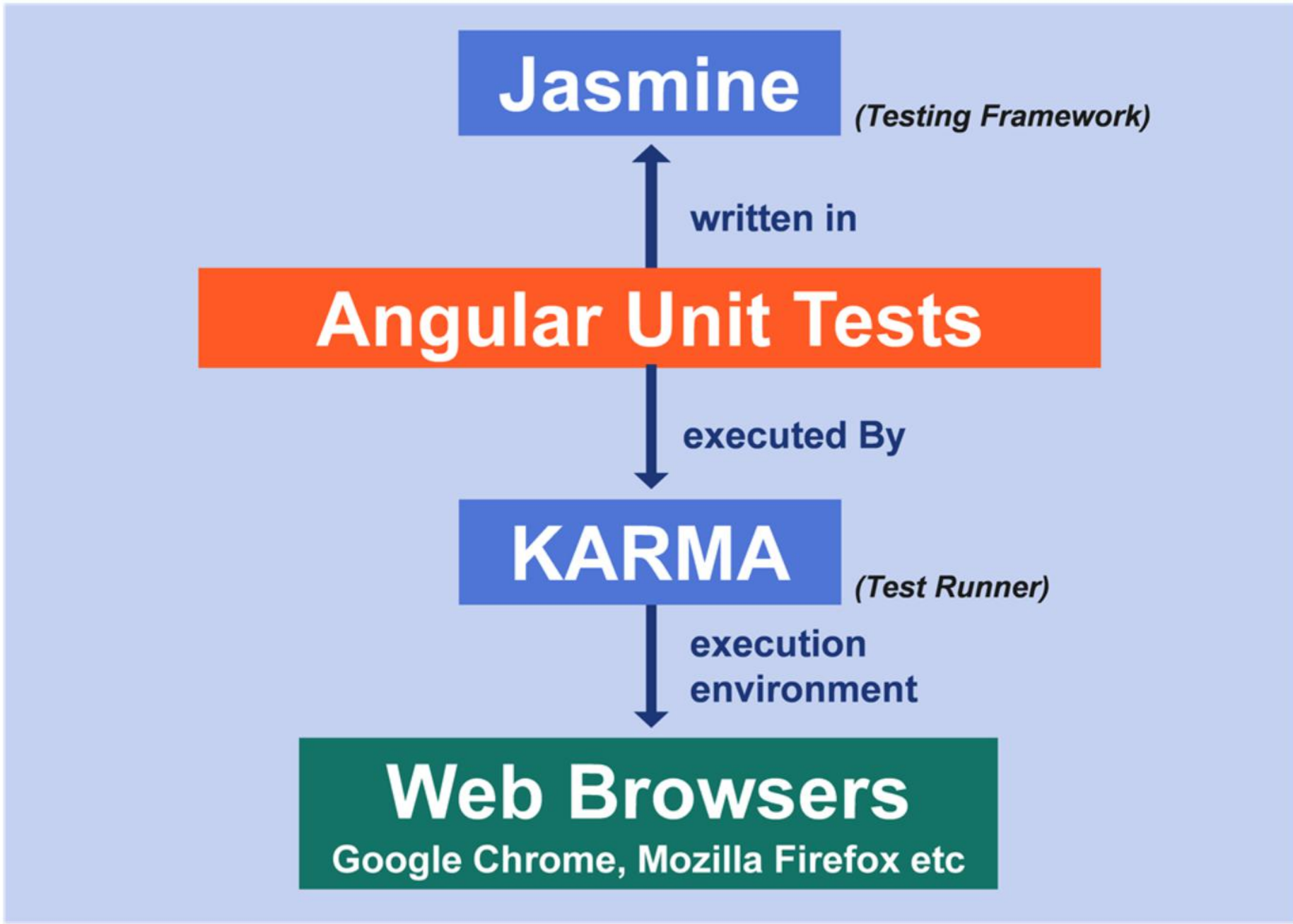


Jones, Capers, Applied Software Measurement: Global Analysis of Productivity and Quality.

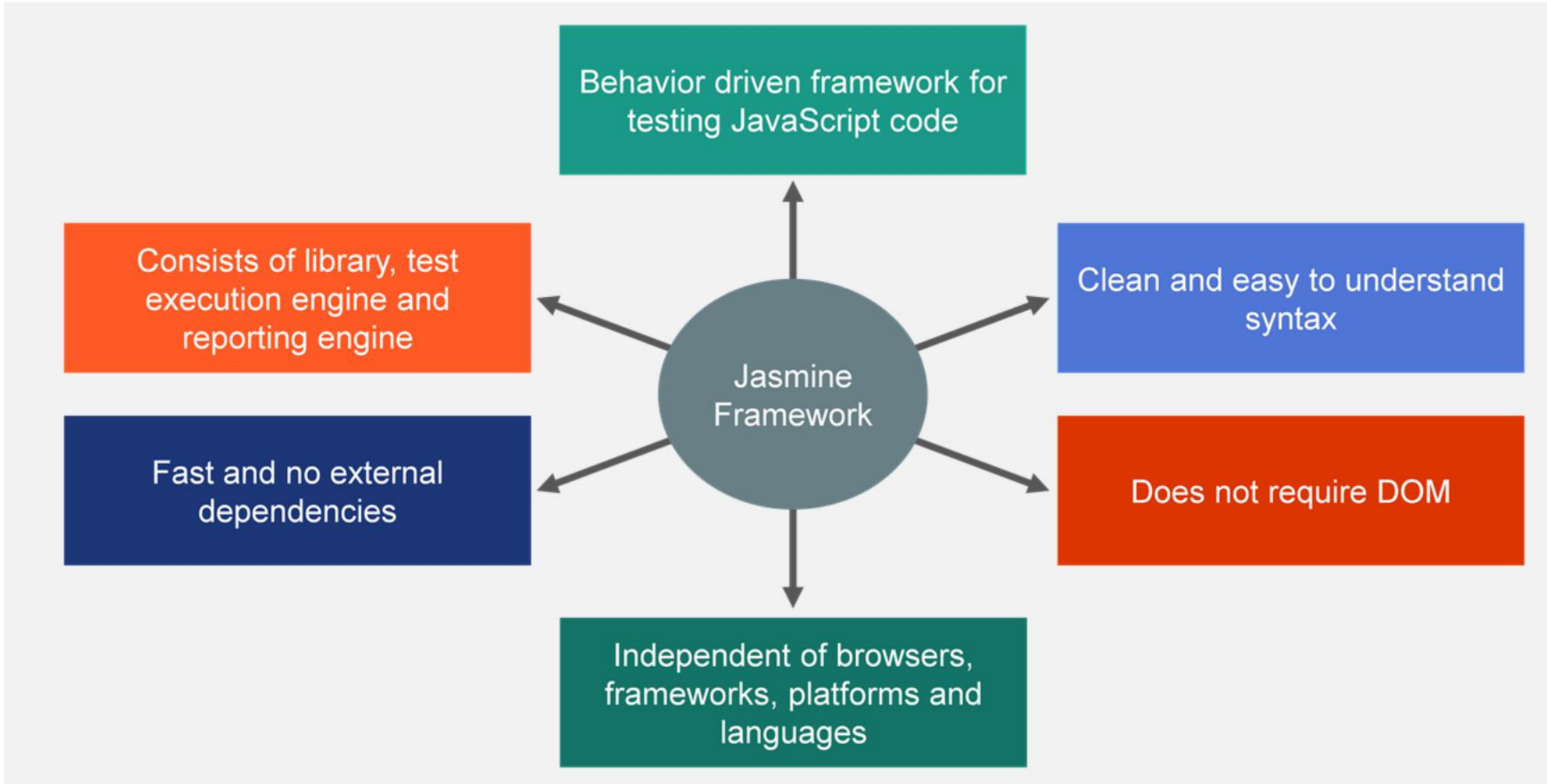
Did You Know?

- To recall, JavaScript code has been tested using Mocha framework and Chai library.
- Which testing tools and framework does Angular provide for testing Angular code?

Jasmine Framework and Karma Test Runner


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

Jasmine Framework



Karma Test Runner

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

- As a test runner, Karma does three important things:
  - It starts a web server and serves your JavaScript source and test files on that server.
  - It loads all the source and test files in the correct order.
  - 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"

Create and Run Tests Using Angular CLI

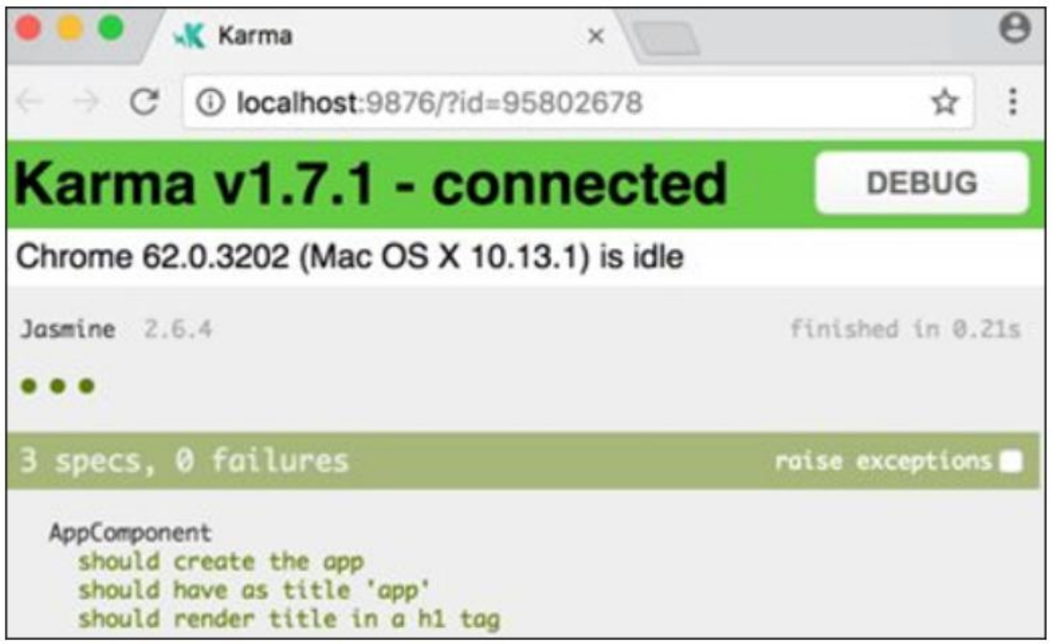
Create an Angular application using the Angular CLI command. Observe the configuration files like angular.json, karma.conf.js, and test.ts, which specify the configuration settings required for testing Angular code.

Run the test code by giving "ng test" CLI command in the terminal and view the test results in console and in the browser window.

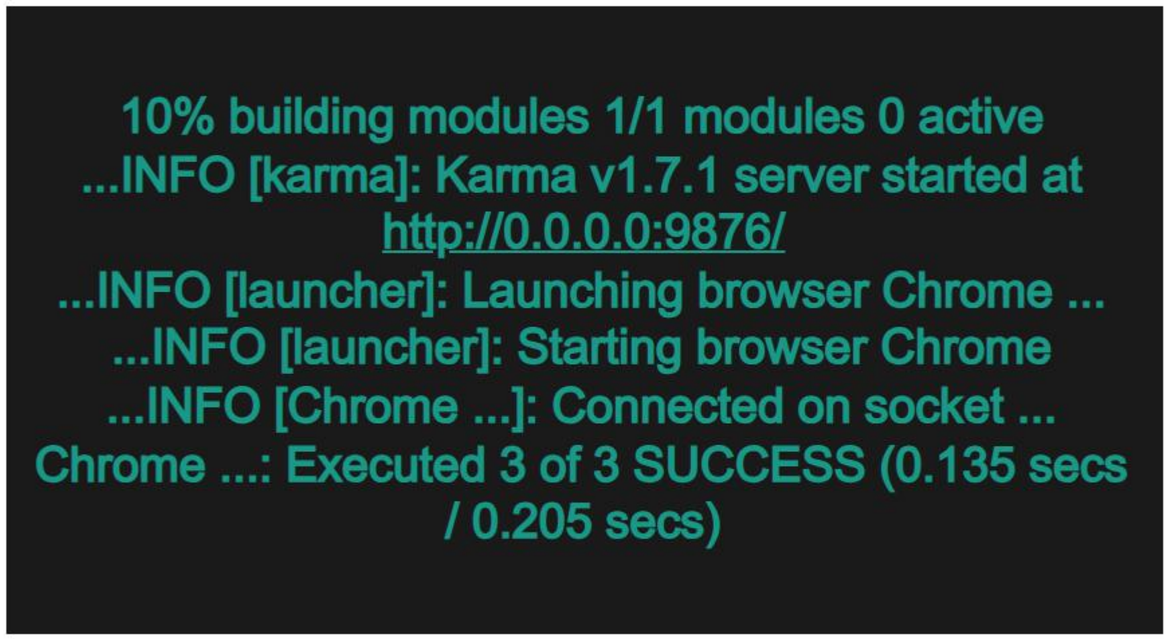
Click here for the demo code

DEMO

Test Results in Browser and Console



Test Results in Browser



Test Results in Console

Test Entry File

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

This file contains the following:

- An environment to run aAgular 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 have "spec.ts" in their name are considered to be tests.

Jasmine Spec File

- The `describe(string, function)` function defines a Test Suite composed of individual test specifications.
- The `it(string, function)` function defines an individual Test Spec which contains one or more Test Expectations.
- The `expect(actual)` expression in conjunction with the *`matcher`* expression describes the expected piece of behavior in the application.
- The `matcher(expected)` expression compares the *`expected`* value passed in and the *actual* value passed to the `expect` function

```
function welcome(){
return 'Welcome to Angular testing!'
}
```

```
describe('Welcome Testing', () => {
  it('says welcome', () => {
    expect(welcome())
        .toEqual('Welcome to Angular
testing!');
  });
});
```

Quick Check

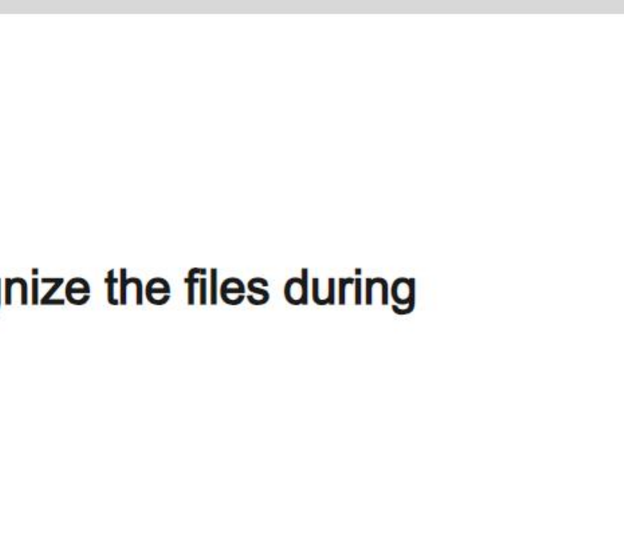
Which of the following file extensions must be used for the Karma test runner to recognize the files during testing?

a) .test.ts

b) .spec.ts

c) .spec.json

d) .conf.js



Quick Check: Solution

Which of the following file extensions must be used for the Karma test runner to recognize the files during testing?

a) .test.ts

b) .spec.ts

c) .spec.json

d) .conf.js

Test Structure

The actual testing code lies inside the it() block. The testing code typically consists of three phases:

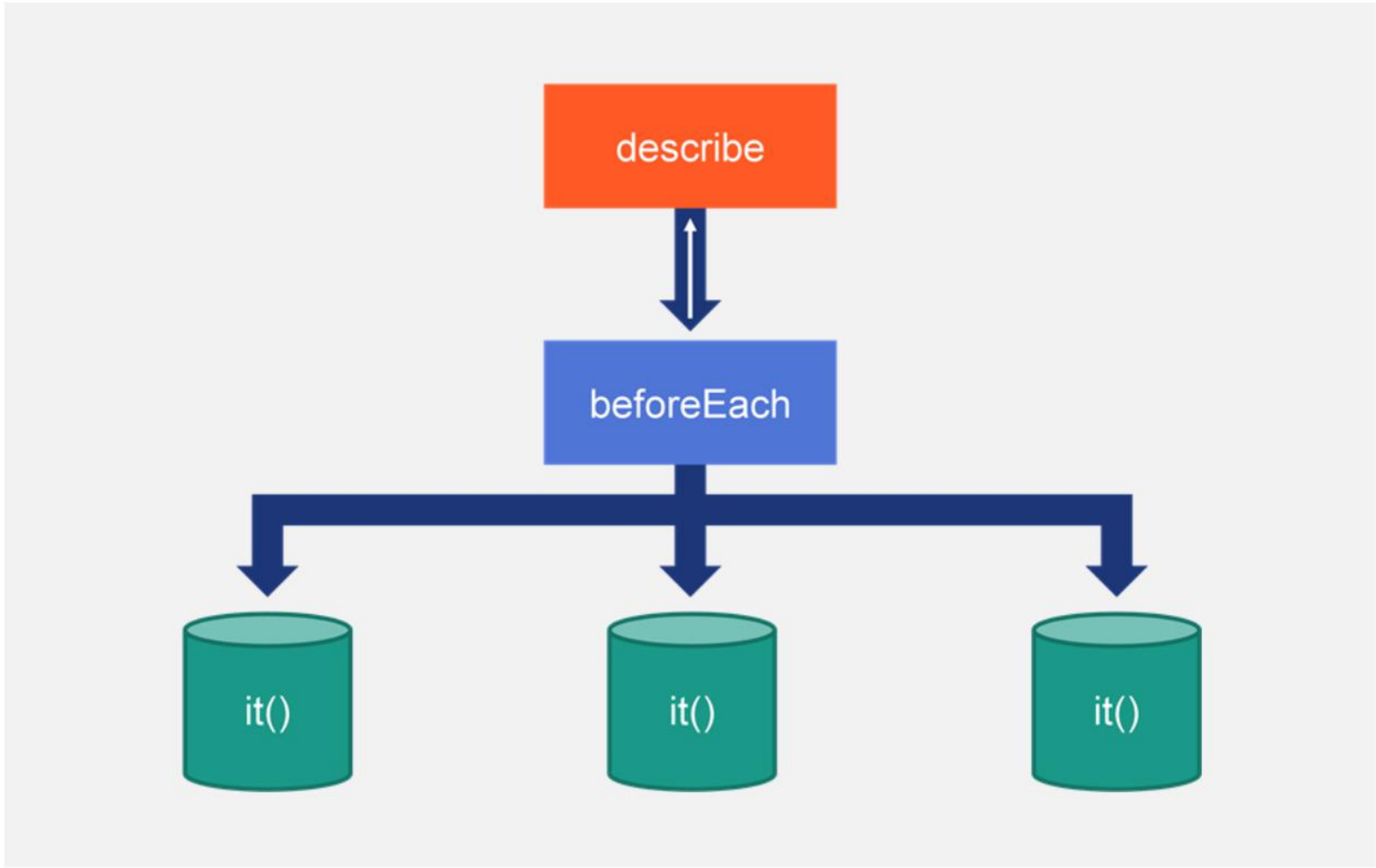
1. Arrange: Preparation and setup phase. For example, the class under test is instantiated.
2. Act: Interaction with the source code under test happens. For example, a method is called or button element in the DOM is clicked.
3. Assert: Code behavior checked and verified. For example, the actual output is compared to the expected output.

```
describe('Suite description', () => {
  beforeAll(() => {
console.log('Called before all specs are
run');
  });
  afterAll(() => {
console.log('Called after all specs are
run');
  });
  beforeEach(() => {
console.log('Called before each spec is
run');
  });
  afterEach(() => {
console.log('Called after each spec is
run');
  });
  it('Spec 1', () => {
  console.log('Spec 1');
  });
  it('Spec 2', () => {
  console.log('Spec 2');
  });
});
```

Efficient Test Suites: Setup and Tear Down

- As you can see, the Arrange phase is similar across multiple specs in a suite.
- When testing a component, creating the component instance and rendering it to the document are repeated over and over.
- This setup can be written inside a function and called at the beginning of each spec.
- Jasmine provides four functions for this, which are called inside `describe` block just like `it()`.
  - `beforeEach`
  - `beforeAll`
  - `afterEach`
  - `afterAll`

Flow of Running Test



Testing Angular Components

Let's Think

- Multiple chores are necessary to render a component in Angular.
- For any component defined, you can find that a "platform" is created, a module is declared, and this module is bootstrapped in `main.ts` file.
- The Angular framework does a lot of work in the background to create a component instance and render it to the DOM finally.
- To do all these jobs manually while testing Angular components is a difficult task.
- Does the Angular framework provide any environment to ease unit testing in Angular?

TestBed

- The TestBed creates and configures an Angular environment to test Angular components easily.
- It configures a testing Module to declare Components, Directives, provide Services as well as to import other modules using the static method `configureTestingModule()`.
- It creates an Angular Module for testing which can be used to instantiate Components.
- In unit test, only those parts to the modules that is strictly necessary are added.
- The Angular compiler compiles all the declared components using `TestBed.compileComponents()` method.

```
beforeEach(async () => {
  await TestBed.configureTestingModule({
  imports: [ /*… */ ],
  declarations: [ /*… */ ],
  providers: [ /*… */ ],
});
});
```

Test Bed Generic Configuration

```
beforeEach(async () => {
  await TestBed.configureTestingModule({
  declarations: [
  AppComponent
  ],
  }).compileComponents();
});
```

**Test Bed Specific Configuration:
Only Declare What is Necessary**

Write Test Cases for Fruit Fantasy App

Edit the app.component.spec.ts file to test the source code present in the Fruit Fantasy app.

Add test cases for Header and Fruit-Manager component for the Fruit Fantasy app to check and validate the features of these components.

Click here for the demo code.

DEMO

Rendering the Component Using Fixtures

- Components can be rendered using the `createComponent()` method.
- `createComponent` returns a `ComponentFixture` which is a wrapper around the Component with useful testing tools.
- `createComponent` renders the Component into a `div` container element in the HTML DOM.
- The template bindings, like `{{title}}` are not evaluated while rendering the component.
- You need to trigger the change detection manually since there is no automatic change detection in our testing environment.
- The fixture references the Component instance via the `componentInstance` property.

```
describe('HeaderComponent', () => {
let fixture:
ComponentFixture<HeaderComponent>;
let component: HeaderComponent;
beforeEach(async () => {
  await TestBed.configureTestingModule({
  declarations: [HeaderComponent],
  }).compileComponents();
  fixture =
TestBed.createComponent(HeaderComponent);
    fixture.detectChanges();
  component = fixture.componentInstance;
   });
});
```

```
const { debugElement } = fixture;
const { nativeElement } = debugElement;
console.log(nativeElement.tagName);
console.log(nativeElement.textContent);
console.log(nativeElement.innerHTML);
```

DebugElement and NativeElement

- The fixtures's `debugElement` property returns the component's host element. For the HeaderComponent, this is the app-header element.
- The DebugElement wraps the native DOM element.
- The DebugElement offers properties like `attributes`, `properties`, `classes`, and `styles` to examine the DOM element.
- Properties like `parent`, `children`, and `childNodes` are used to navigate in the DOM tree.
- Every DebugElement has a `nativeElement` property to access the native element inside.
- nativeElement is typed as `any` because Angular does not know the exact type of the wrapped DOM element.

Querying the DOM

- Every `DebugElement` offers the methods `query` and `queryAll` for finding descendant elements.
- `query` returns the first descendant element that is matching.
- `queryAll` returns an array of all matching elements.
- Both methods expect a predicate, that is a function which checks every element and returning true or false.
- Angular provides predefined predicate function `By.css` to query the DOM using CSS selectors.
- `query` method returns a `DebugElement`, while `queryAll` returns an array of `DebugElements` (`DebugElement[]`)

```
const { debugElement } = fixture;
// Find the first p element
const p= debugElement.query(By.css('p'));
// Find all elements with the class .card
const cardElements =
debugElement.queryAll(By.css('.card'));
// Find all elements of button type
const buttonElements = debugElement.query
All(By.css('button[type=button]'));
```

```
let button =
fixture.debugElement.query(By.css(
'button'));
button.triggerEventHandler('click'
, null);
fixture.detectChanges();
```

Triggering Event Handlers

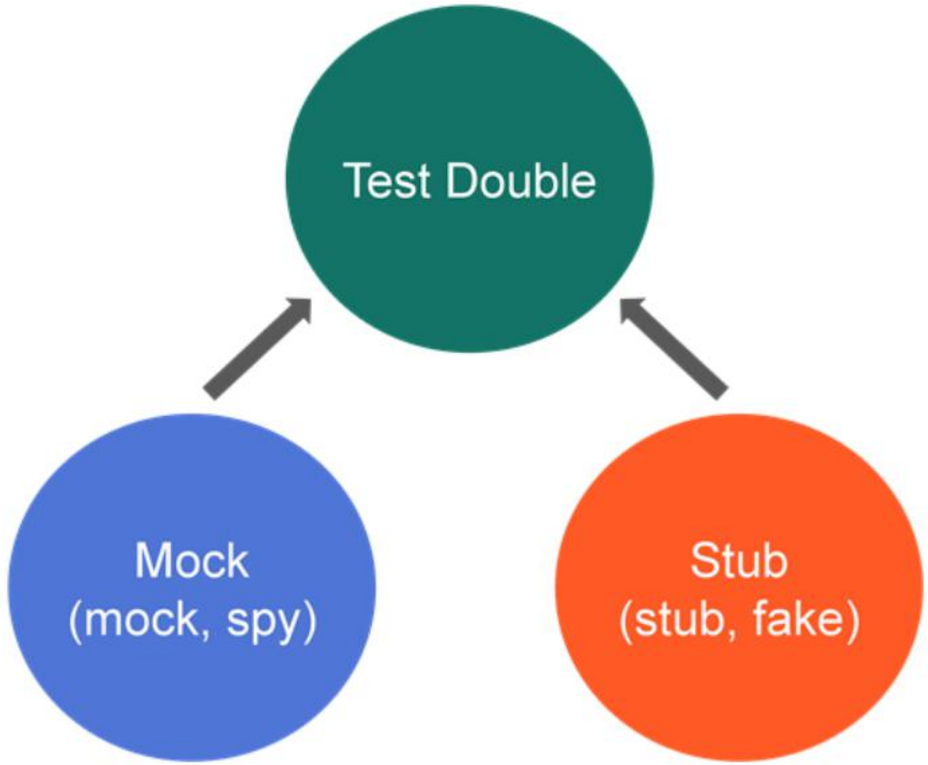
- It is a general practice in tests to simulate user inputs like typing, clicking, moving pointers and pressing keys.
- User events cause DOM events. These events are handled by the registered event handler methods.
- `DebugElement` **provides** `triggerEventHandler` **method for firing events during testing.**
- This method calls all event handlers for a given event type, like *click*.
- This method takes a fake event object as a second argument that is passed to event handler methods.
- If the event handler methods do not access the event object, then `null` can be passed as a second argument.

Code Dependencies and Unit Testing

- In a real-time application, Angular components depend on other codes; that is, they have dependencies. May be, it gets some inputs from some data sources, or it may send its outputs to some destinations.
- There might be services that get some information from an external API.
- Unit tests are not integration tests, and they must test single unit of code at any time.
- For example, it is not the right choice to write test for code that interacts with databases by booting up a "test database" where the test can trigger a write and validate it by querying the DB.
- Test Doubles help us in these situations where our code is interacting with external APIs.
- Whether it may be a connection to a database, or an API, or even a file, Test Doubles provide the code that uses the services with a simpler version.
- This new version (called stub) returns a known and controlled value.

Testing a Code Unit In Isolation

- While doing unit testing, the purpose of the specification is to test the component, not its dependents, and real dependents can be trouble.
- It is usually better if they are test doubles (stubs, fakes, spies, or mocks).
- It is far easier and safer to create and register a test double in place of the dependents.



Note: A test double is a generic term to refer to any object in production during testing.

Faking Functions Using Jasmine Spies

Create a utility class which has a function to perform a simple addition operation. Make the AppComponent to be dependent on this utility class function.

Then test the AppComponent in isolation with the dependent function using Jasmine spies.

Click here for the demo solution.

DEMO

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();
```

Quick Check

Which of the following TestBed methods is used to create an Angular testing module to create module environments for testing the class?

a) createTestModule

b) createTestingModule

c) configureTestModule

d) configureTestingModule

Quick Check: Solution

Which of the following TestBed methods is used to create an Angular testing module to create module environments for testing the class?

a) createTestModule

b) createTestingModule

c) configureTestModule

d) configureTestingModule