**Assignment 1:**

**Date: 15-03-2021**

**Angular Unit Testing**

**(Nunit + Jasmine + Karma)**

1. **Write brief description about unit testing and functional testing and its benefit as developer perspective?**

**Unit Testing:**

Unit testing is a popular practice adopted by many organizations as it primarily helps to eliminate the risk of having a bug in codes.

As the name suggests, unit testing is a type of software testing where individual units or components are tested. The purpose is to validate each unit of the software code and check whether they are performing as expected.

The goal of unit testing is to break each part of the source code into the unit and check that each part works properly. It means that if any set of input is not functional, it should return an expected output.

**The Prime Objective of Unit Testing Is:**

* To isolate a section of code.
* To verify the correctness of code.
* To test every function and procedure.
* To find early bugs and fix in the development cycle and to save costs.
* To help the developers to understand the code base and enable them to make changes quickly.
* To help for code reuse.

**Functional Testing:**

Functional testing is a software testing process that validates the software system against functional requirements. The purpose of Functional testing is to test each functionality of the software application, by providing apt input, output verification against the Functional requirements. In other words, functional testing is a type of testing whereby the system is tested against the function requirement/specification.

### The Prime Objective of Functional Testing Is:

* Finding defects that the programmer may have missed while developing software.
* To Gain confidence and provide information about the level of quality.
* To prevent defects.
* To ensure that the result meets the business and the user’s requirements.

## **Difference Between Unit Testing and Functional Testing**

|  |  |  |
| --- | --- | --- |
| **Factors** | **Unit Testing** | **Functional Testing** |
| Definition and Purpose | Testing individual modules | Testing the functionality as per user requirements |
| Written by | Developers | Tester |
| Testing Techniques | White-box testing | Black box testing |
| Errors | Code branches, Edge cases | Software / Application |
| No. Of Test Cases | Higher than another testing | Lower than unit and integration testing |
| Cost and Maintains | Low | High |
| Changes | Changes frequently | Low rates of changes |

## **Benefits of Testing:**

* Makes the Process Agile. One of the main benefits of unit testing is that it makes the coding process more Agile.
* Quality of Code. Unit testing improves the quality of the code.
* Finds Software Bugs Early.
* Facilitates Changes and Simplifies Integration.
* Provides Documentation.
* Debugging Process.
* Design.
* Reduce Costs.
* **Improve the design of implementations.**  
  Start coding a feature without giving it a lot of thought to the design is a very common mistake among developers. Using unit testing is going to enforce you to think and re-think the design, and if you are using TDD the impact is even bigger.
* **Allows refactoring.**  
  Since you already have tests that ensure that everything is working as expected, you can easily add changes to that code with the certainty that you are not adding any bugs.
* **Add new features without breaking anything.**  
  When you are adding a new feature you can run the tests to ensure that you aren’t breaking any other part of the application.
* **Tests are good documentation.**
* **Tests make developers more confident about their work.**

I have Attached one type script file of some test cases for above implementation which is based on your examples given in lecture classroom.



**2. Where and why do you needed unit testing in your project? give me 10 example and code snap?**

Unit testing allows the programmer to refactor code at a later date, and make sure the module still works correctly. The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified and fixed.

Unit Testing is important because software developers sometimes try saving time doing minimal unit testing and this is myth because inappropriate unit testing leads to high cost Defect fixing during System Testing, Integration Testing and even Beta Testing after application is built. If proper unit testing is done in early development, then it saves time and money in the end.

* Unit tests help to fix bugs early in the development cycle and save costs.
* It helps the developers to understand the testing code base and enables them to make changes quickly.
* Good unit tests serve as project documentation.
* Unit tests help with code re-use. Migrate both your code and your tests to your new project. Tweak the code until the tests run again.

Angular unit testing enables you to test your app based on user behavior. While testing each possible behavior would be tedious, inefficient, and ineffective, writing tests for each coupling block in your application can help demonstrate how these blocks behave.

Here some reasons are mentioned below.

## Discipline and Rigor

## Reduce Cyclomatic Complexity

## Your Software Is Used Before Delivery

## Documentation

## Measure the Effort Needed to Modify an Existing Feature

## Code Coverage

## Performance

## Enables Continuous Integration (CI)

**Examples:**

**1. Angular automated testing**

describe('TestSuitName', () => {

// suite of tests here

it('should do some stuff', () => {

// this is the body of the test

});

});

**2. Components**

describe('HeaderComponent', () => {

let component: HeaderComponent;

let element: HTMLElement;

let fixture: ComponentFixture<HeaderComponent>;

// \* We use beforeEach so our tests are run in isolation

beforeEach(() => {

TestBed.configureTestingModule({

// \* here we configure our testing module with all the declarations,

// \* imports, and providers necessary to this component

imports: [CommonModule],

providers: [],

declarations: [HeaderComponent],

}).compileComponents();

fixture = TestBed.createComponent(HeaderComponent);

component = fixture.componentInstance; // The component instantiation

element = fixture.nativeElement; // The HTML reference

});

it('should create', () => {

expect(component).toBeTruthy();

});

it('should create', () => {

// \* arrange

const title = 'Hello Chetan';

const titleElement = element.querySelector('.header-title');

// \* act

component.title = title;

fixture.detectChanges();

// \* assert

expect(titleElement.textContent).toContain(title);

});

});

**3. Pipes**

describe('TroncaturePipe', () => {

it('create an instance', () => {

const pipe = new TroncaturePipe(); // \* pipe instantiation

expect(pipe).toBeTruthy();

});

it('truncate a string if its too long (>20)', () => {

// \* arrange

const pipe = new TroncaturePipe();

// \* act

const ret = pipe.transform('123456789');

// \* assert

expect(ret.length).toBe(20);

});

});

**4. Directives**

// \* Host component:

@Component({

template: `<div [appPadding]="2">Test</div>`,

})

class HostComponent {}

@NgModule({

declarations: [HostComponent, PaddingDirective],

exports: [HostComponent],

})

class HostModule {}

**5. Services**

describe('LocalService', () => {

let service: LocalService;

beforeEach(() => {

TestBed.configureTestingModule({

providers: [LocalService],

});

service = TestBed.get(LocalService); // \* inject service instance

});

it('should be created', () => {

expect(service).toBeTruthy();

});

it('should set the local', () => {

// \* act

service.setLocal('fr');

// \* assert

expect(service.getLocal()).toBe('fr');

});

});

**6. Dealing with HTTP requests**

class ApiService {

// Implement the methods you want to overload here

getData() {

return of({ items: [] }); // \* mocks the return of the real method

}

}

//...

TestBed.configureTestingModule({

imports: [],

declarations: [myComponent],

providers: [

{

provide: RealApiService,

useClass: ApiService,

}, ], });

**7. Dealing with asynchronous code**

it('should do something async', async () => {

// \* arrange

const ob = { id: 1 };

component.selected = ob;

// \* act

const selected = await component.getSelectedAsync(); // get the promise value

// \* assert

expect(selected.id).toBe(ob.id);

});

**8. Dealing with observables**

it('should do something async', (done) => {

// \* arrange

const ob = { id: 1 };

component.selected = ob;

// \* act

const selected$ = component.getSelectedObs(); // get an Observable

// \* assert

selected$.subscribe(selected => {

expect(selected.id).toBe(ob.id);

done(); // let Jasmine know that you are done testing

});

});