Unit Test

SD550 – Web Application Development 2

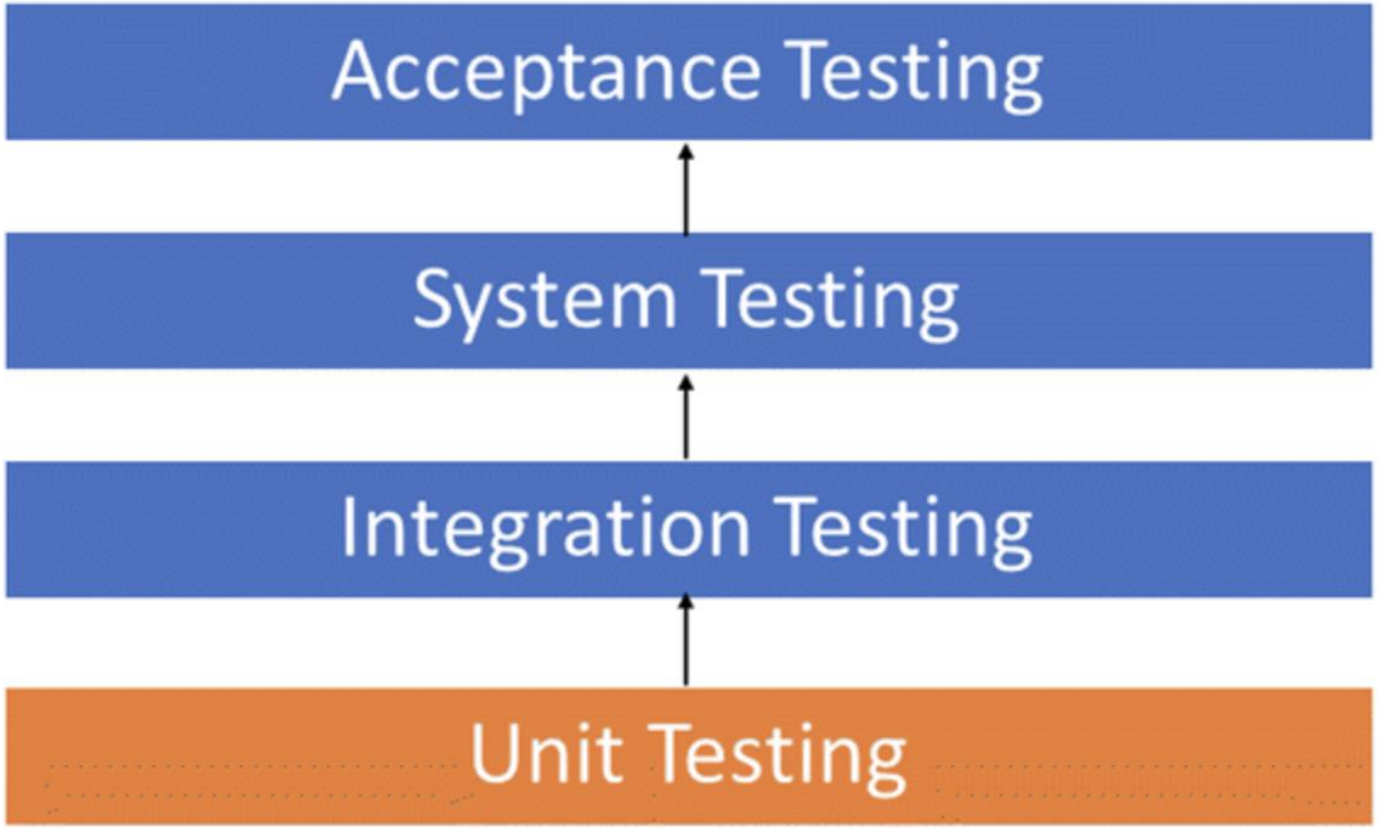
Maharishi International University

Department of Computer Science

M.S. Thao Huy Vu

Maharishi International University - Fairfield, Iowa

Definition


Acceptance Testing
System Testing
Integration Testing
Unit Testing

Definition

- **Unit testing** is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation. This testing methodology is done during the development process by the software developers and sometimes QA staff.
- **Integration testing** -- also known as integration and testing (I&T) -- is a type of software testing in which the different units, modules or components of a software application are tested as a combined entity. However, these modules may be coded by different programmers.
- **System testing**, also referred to as system-level tests or system-integration testing, is the process in which a quality assurance (QA) team evaluates how the various components of an application interact together in the full, integrated system or application.
- **Acceptance Testing** is a method of software testing where a system is tested for acceptability. The major aim of this test is to evaluate the compliance of the system with the business requirements and assess whether it is acceptable for delivery or not. This phase can involve different departments (QA, Business, Customer Service...)

Introduction to Automated Testing

- To understand automated testing, let's consider the manual testing process which goes as follows:
  - Write code
  - Compile (Build) to executable form
  - Execute the code manually (in some cases, enter input data in a form etc.) Check the result (such as output on the screen or values of variables or database data or output log files etc.)
  - If it does not work, if the result is incorrect, we repeat the above process
- In Automated Testing, these above steps are perform using code

Automated versus Manual Testing

- Some aspects of application software require manual testing. Such as:
  - Finding strange edge cases
  - Judging about the aesthetics and visual design
  - Judging about the overall user experience
- However, Automated testing offers the following advantages:
  - Tests can be run many times, over and over
  - Tests are quicker to run
  - Tests can be run anytime
  - Excellent for checking mechanical & logical assertions

Automated versus Manual Testing

Manual test	Automated test
Boring	Challenging
Not reusable	Reusable
Complex manual setup and teardown	Automated setup and teardown
High risk to forget something	Zero risk to forget something
No safety net	Safety net
Low training value	Acts as documentation
Slow and time consuming	Fast

Unit Testing with Jest Framework

- A unit test is a test that test one single class.
  - A test case test one single method
  - A test class test one single class
  - A test suite is a collection of test classes
- Unit tests make use of a testing framework

- A unit test
  1. Create an object
  2. Call a method
  3. Check if the result is correct

Qualities of Good Unit Test: F I R S T

- Fast
- Isolated
- Repeatable
- Self-validating
- Timely

First Unit Test

- Create a file for testing a component by the name: <Component-name>.test.tsx
- Import:

```
import { render, screen, fireEvent } from '@testing-library/react';
import '@testing-library/jest-dom';
```

- Write simple test case:

```
test('simple case', () => {
  expect(2+2).toEqual(4);
})
```

Structure of test suite

```
describe('Name of Test Suite', () => {
    //test cases
    //setup & teardown
});
```

Test case

```
test('Name of Test 1', () => {
  // Test implementation for Test 1
});
it('should behave in a certain way', () => {
  // Another test implementation
});
```

Setup

```
// Optional: One-time setup before all tests in this suite run
 beforeAll(() => {
   // Code here runs once before all tests
 });
 // Optional: Setup before each test
 beforeEach(() => {
   // Code here runs before each test
 });
```

Teardown

```
// Optional: Teardown after each test
  afterEach(() => {
    // Code here runs after each test
  });
  // Optional: One-time cleanup after all tests in this suite run
  afterAll(() => {
    // Code here runs once after all tests
  });
```

Test GUI

```
import { render, screen, fireEvent } from '@testing-library/react';
import '@testing-library/jest-dom';
```

Render & Screen

- Render: Tool to create the simulated DOM
- Screen: A utility to access the elements
  - GetByText

  screen.getByText("Save")
  - GetByRole

  screen.getByRole("button", {name: "Submit"})
  - GetByLabelText

  screen.getByLabelText("Email Address")
  - GetByPlaceHolder

  screen.getByPlaceHolderText("Enter name")
  - GetByTestId

  screen.getByTestId("my-button")

Event: fireEvent

- Click

fireEvent.click(screen.getByText('Submit'))

- Change

fireEvent.change(screen.getByPlaceholderText('Enter your name'), { target: { value: 'John Doe' } })

- Focus

fireEvent.focus(screen.getByLabelText('First Name'))

Matchers for HTML elements

- ToBeInTheDocument: Checks if an element is present in the document

expect(screen.getByText('Hello World')).toBeInTheDocument();

- ToBeVisible: Ensures that an element is not only in the document but also visible to the user.

expect(screen.getByText('Visible Content')).toBeVisible();

- ToHaveAttribute: Asserts that an element has a specific attribute with an optional value

expect(screen.getByText('Link')).toHaveAttribute('href', 'https://example.com')

Matchers for HTML elements

- ToHaveValue: Ensures that form elements like **input**, **select**, or **textarea** have the specified value

expect(screen.getByPlaceholderText('Name')).toHaveValue('John Doe');

- ToHaveClass: Checks if an element has certain CSS classes.

expect(screen.getByText('Important')).toHaveClass('highlight');

- ToHaveTextContent: Asserts that an element has specific text content

expect(screen.getByTestId('message')).toHaveTextContent('Hello World');

Matchers for Data

- **ToEqual**: Checks for the equality of complex objects like arrays and objects by checking the equality of all properties and elements.

expect(object).toEqual({ key: 'value' });

- **ToBe**: Checks if a value is exactly the same as the other.

expect(value).toBe(123);

- **ToBeGreaterThan/ToBelessThan:** Asserts that a number is greater than or less than another number.

expect(count).toBeGreaterThan(10);

Matchers for Data

- **ToBeGreaterThanOrEqual/ToBeLessThanOrEqual:** Checks if a number is greater than or equal to, or less than or equal to another number.

expect(temperature).toBeLessThanOrEqual(100);

- **ToThrow:** Checks if a function call throws an exception.

expect(() => myFunction()).toThrow();

- **ToHaveProperty:** Asserts that an object has a specific property, optionally checking that property's value.

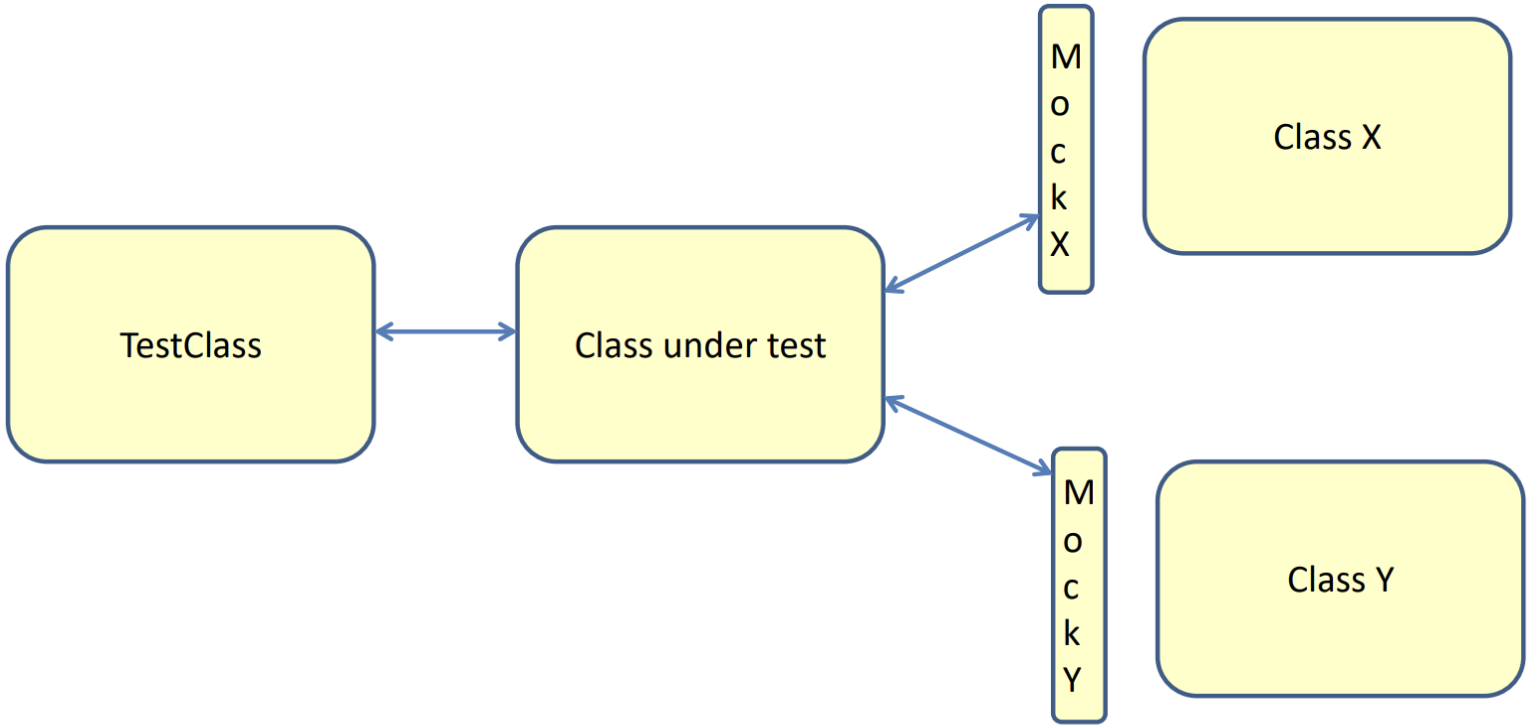
expect(obj).toHaveProperty('key', 'value');

- **ToBeTruthy/ToBeFalsy:** Asserts that a value is truthy or falsy. This is useful for validating conditions.

expect(isError).toBeFalsy();

Mock

- Help to isolate a module by leveraging fake modules



Mock - Module

- Declare the mocked module at top

jest.mock('some-module');

- Return value

someModule.someFunction.mockReturnValue('returned value');

- Return Promise.Resolve

someModule.fetchData.mockResolvedValue({ data: 'some data' });

- Return Promise.Reject

someModule.fetchData.mockRejectedValue(new Error('Error'));

Mock - Function

- Declare the mocked module at top

someFunction = jest.fn();//synchronous function

FetchData = jest.fn();//async function

- Return value

someFunction.mockReturnValue('returned value');

- Return Promise.Resolve

fetchData.mockResolvedValue({ data: 'some data' });

- Return Promise.Reject

fetchData.mockRejectedValue(new Error('Error'));

Matchers for functions

- **toBeCalled() / toHaveBeenCalled()**

expect(myMockFunction).toHaveBeenCalled();

- **toBeCalledWith() / toHaveBeenCalledWith()**

expect(myMockFunction).toHaveBeenCalledWith(arg1, arg2);

- **toBeCalledTimes() / toHaveBeenCalledTimes()**

expect(myMockFunction).toHaveBeenCalledTimes(3);

Reference

https://jestjs.io/