Learning Consolidation

Implement Unit Testing Using Mocha and Chai

In this Sprint, you learned to:

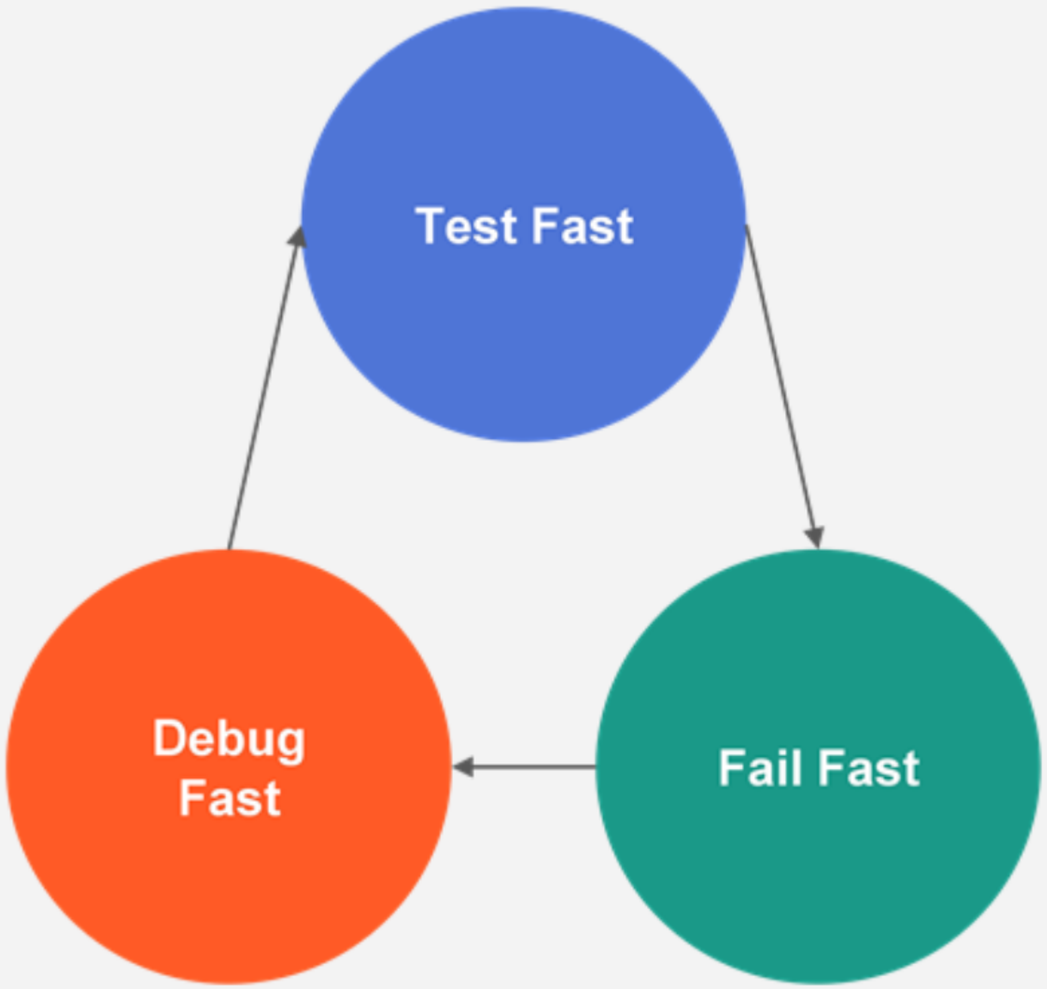
- Explain importance of software testing
- List levels of testing
- Explain significance of early testing
- Install `Mocha` and `Chai` for unit testing JavaScript code
- Implement testing on JavaScript code

What is Software Testing?

- Software testing is a process that verifies and validates code to ensure it is:
  - Defect free
  - Meets the users' requirements
  - Meets the technical requirements listed down in design document
- Testing can be done at different levels to ensure each unit of code works as expected, the complete solution code works as expected, and the solution code meets all the users' expectations.
- There are 4 levels of testing:
  - Unit testing
  - Integrated testing
  - System testing
  - Acceptance testing

Importance of Testing

Customer Satisfaction	Helps ensure code meets users expectations and hence is reliable
Cost Effectiveness	Cost on testing is much less than the cost incurred to fix a defect
Security	Assures security of users data and hence builds trust
Quality Product	Helps test for correctness, completeness and performance of code
Development Efficiency	Guarantees code developed meets the stated user as well as technical requirements



When to Perform Unit Testing?

- Testing is an iterative process
- Testing is performed until the actual results match expected results
- For better cost efficiency, first write test case and then the solution code
- It helps detect and correct defects at an early stage of development
- This approach is known as **Test First** approach

Importance of Early Testing

- Testing is an integral part of software development.
- Testing builds user's trust in the software.
- Testing helps discover defects in system.
- Early testing helps discover defects early in the development process.
- Hence, for better cost efficiency, first write test case and then the solution code.

		Time Detected				
		Requirements	Architecture	Construction	Software test	Post-release
Time introduced	Requirements	1×	3×	5–10×	10×	10–100×
	Architecture	-	1×	10×	15×	25–100×
	Construction	-	-	1×	10×	10–25×

Cost to fix defect increases 10 to 100 times if not detected in early stages

Early Testing Approaches

TDD

- TDD – Test Driven Development
- The process starts by writing a test case
- Test cases are written in a programming language.
- Collaboration is required only between the developers
- Tests in TDD can only be understood by people with programming knowledge.

BDD

- BDD – Behavior Driven Development
- The process starts by writing a scenario as per the expected behavior
- Scenarios are more readable when compared to TDD as they are written in simple English format
- Collaboration is required between all the stakeholders
- Tests in BDD can be understood by any person including the ones without any programming knowledge

Types of Testing

- Testing can be done manually
- Testing process can also be automated

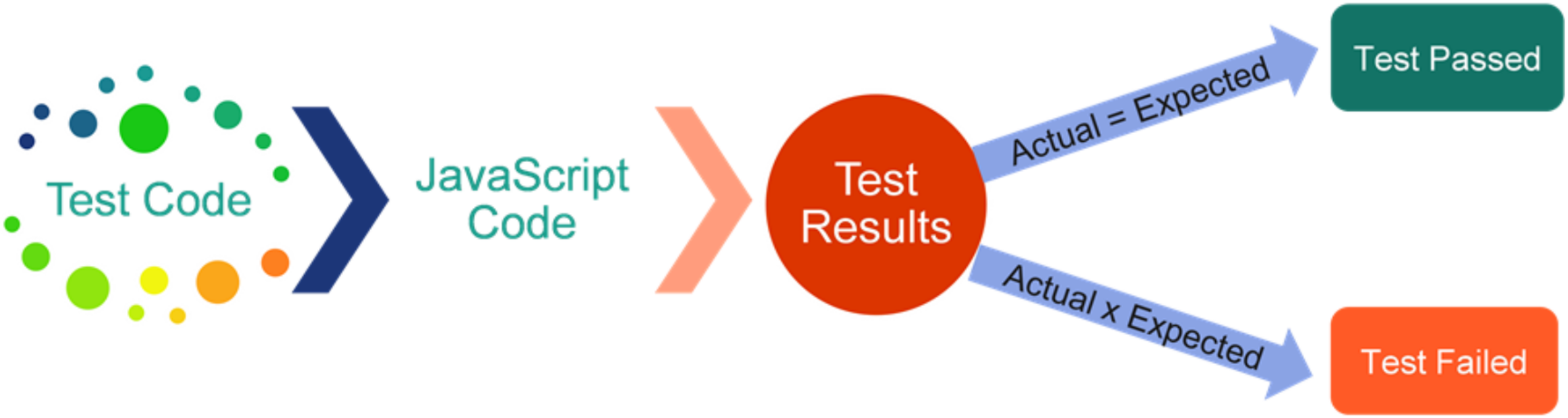
Manual Testing

- Performed by humans
- Time consuming
- Lacks accuracy
- **Less Reliable**

Automated Testing

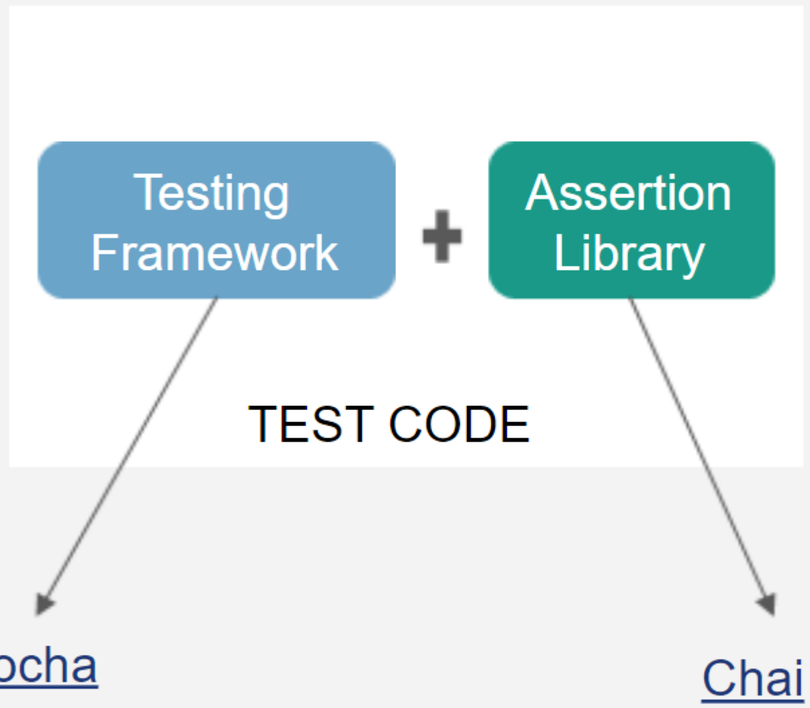
- Performed by software program
- Very fast
- Highly accurate
- **Highly Reliable**

How to Automate Unit Testing?



Automated unit testing requires developer to:

- Write the test code
  - Test code should refer to JavaScript function that needs to be tested
- Run the test code
  - The execution of test code generates test results
  - Test results state whether actual results match with the expected results
  - If actual result matches expected result, then the test has passed
  - If actual result does not match expected result, then the test has failed



Design Test Code

- To automate testing, test code needs to be written.
- Test frameworks and assertion libraries provide predefined functions.
- The functions provided by test frameworks help to define test code.
- Within the test code, functions provided by assertion libraries are used to write test scripts.
  - The assertion scripts help to compare actual results with the expected results.

Mocha – Test Framework

- Mocha is a JavaScript based test framework.
- This test framework provides functions that help define test code for testing JavaScript functions.
- It runs on Node.js and in the browser
  - Node.js provides runtime environment to run JavaScript code
  - Installing Node.js is a pre-requisite for installing Mocha
- Click here to visit the official site for Mocha.

Note: In this program, Mocha will be installed to run on Node.js.

Chai – Assertion Library

- Chai is an assertion library for node and the browser.
- This assertion library can be delightfully paired with any JavaScript testing framework such as Mocha.
- It provides various assertion styles that help write test scripts and they are:
  - Assert
  - Expect
  - Should
- Installing Node.js is a pre-requisite for installing Chai.
- Click here to visit official site for Chai.

Note: In this program, Chai will be installed to run on Node.js.

Node.JS And Node Package Manager (NPM)

- Node.js is a JavaScript runtime environment built on Chrome's V8.
- It allows you to execute a JavaScript code on server-side
- One of the major factors of Node's success is NPM-Node Package Manager
- npm is the default package manager for JavaScript's runtime Node.js
- npm gets installed along with Node.js installation
- npm consists of two main parts:
  - CLI (command-line interface) tool for publishing and downloading packages
  - Online repository that hosts JavaScript packages – npm.js
- Click here to visit official site for Node.js.

About the Test Code

```
const chai = require('chai');
const expect = chai.expect;

const display = require('../solution/script');

describe('',function() {
    it('',function() {
        ;
    });
});
```

- The test code uses `require()` function to access
  - Assertion function from `Chai` library
  - JavaScript function(s) that need to be tested
- The test code contains calls to `describe()` and `it()` functions to define test code.
  - These functions are provided by `Mocha` framework

The describe() Function

```
describe('Script', function() {

});
```

- `describe()` is called to group tests in `Mocha`.
- `describe()` takes two arguments:
  - the first is the name of the test group
  - the second is a callback function that includes test cases.

Note: Callback function is a function passed as a parameter to the called function.

The `it()` Function

```
describe('Script',function() {
    it('should have function display()',function() {
        // test code
    });
});
```

- The function `it()` is called to define a particular test case.
- It takes two arguments:
  - the first is the string explaining purpose of the test.
  - the second is a callback function which contains actual test code.
- The `describe()` function can have multiple calls to `it()` function, each representing a particular test case.

Mocha Hooks

- `Mocha` provides methods termed as "hooks" that help write compact and maintainable code.
- These hooks are (in the order of execution):
  - `before()` – runs once before the first test in this block
  - `beforeEach()` – runs before each test in this block
  - `afterEach()` – runs after each test in this block
  - `after()` – runs once after the last test in this block
- `before()` and `beforeEach()` help setup preconditions
  - Preconditions help set up pre-requisites that are required to execute before the test script runs
- `after()` and `afterEach()` help perform clean up after tests
  - Clean up tasks help manage memory that are allocated during test case execution and are no longer required.

Using Assertion Library

```
describe('Script',function() {
    it('should have function display()',function() {
        expect(typeof display).to.be.equal('function');
    });
});
```

Actual result — **Language Chains** — **Expected result**

Matches actual result with expected result and generates test result

- Inside `it()` function, `expect()` function and language chains provided by the `Chai`'s assertion library are used to construct the assertion statement.
- Language chains help build simple English like assertion statement making the code readable to even non-tech readers.
- Chaining is a technique whereby output from a function is provided as an input to the other function.

expect() with Message Parameter

```
describe('Script',function() {
    it('should have function display()',function() {
        expect(display, "function with the name
                         display is not defined")
        .to.be.equal("function");

    });
});
```

Message Parameter

```
Script
    1) should have function display()

  0 passing (8ms)
  1 failing

  1) Script
       should have function display():
     AssertionError: function with the name display is not defined: expected [Function
displays] to equal 'function'
      at Context.<anonymous> (test\script-spec.js:7:16)
      at processImmediate (node:internal/timers:466:21)
```

- The expect() function can optionally take an additional string parameter.
- This string parameter value is displayed as a message on console when the test case fails.

More on Assertions

- Assertions are statements written to compare expected result with actual result.
- The Chai's assertion library provides three different assertion styles for writing the assertion statements:
  - Assert
    - Uses classic assert-dot notation similar to any JavaScript's object-dot notation
    - For e.g., `assert.typeOf(display, "function");`
  - Expect
    - Uses chainable language constructs for creating readable assertion statement
    - For e.g., `expect(typeof display).to.be.equal('function');`
  - Should
    - Also uses chainable language constructs for creating readable assertion statement.
    - Uses `should` property of the object to be tested to start the chain
    - For e.g., `display.should.be.a("function");`

Language Chains

- In `Chai, expect()` is a function which takes a single argument, the value under test.
- All expectations against the value are either properties or methods.
- These properties and methods can be chained together to create the "expectation chain"
  - The expectations are added or chained to the output of the `expect()` function.
- The "expectation chain" help build a language chain making the assert statement readable like a natural English language statement.
- In the below examples, `to`, `be`, `true` are the properties and `greaterThan()`, `a()`, `an()` are the methods used to create language chains.
  - `expect(result).to.be.true;`
  - `expect(value).to.be.greaterThan(5);`
  - `expect('foo').to.be.a('string');`
  - `expect('foo').to.not.be.an('array');`

Testing Considerations

- Testing is an integral process in software development. It is therefore critical to ensure testing is done thoroughly.
- Ensure all requirements are considered and mapped to different test scenarios.
- Requirements should be tested using positive and negative test scenarios.
  - Positive test scenarios test whether the function returns the expected result for the valid inputs provided.
  - Negative test scenarios test whether the function returns appropriate message for the invalid inputs provided.
- Each function in the solution code should be tested for all the above-mentioned requirements.
  - Failing to do so can leave a defect undetected resulting into a major concern later.
  - **If required, code should be refactored to ensure all the test requirements are fulfilled.**

Planning Test Cases

- Before writing test cases, plan the test cases.
- Test cases should be written for each and every function to ensure that the function
  - Exists with the specified name
  - Generates expected output for the input provided
  - Generates appropriate message for the invalid / incorrect / insufficient input provided

Self Check

Which of the below statements are true for automated unit testing?

1. The testing is done by software code

2. Is more accurate than manual testing

3. Is less time consuming than manual testing

4. Is more reliable than manual testing

Self Check: Solution

Which of the below statements are true for automated unit testing?

1. **The testing is done by software code**
2. **Is more accurate than manual testing**
3. **Is less time consuming than manual testing**
4. **Is more reliable than manual testing**

Self Check

Which of the below assertion library is used to perform unit testing on JavaScript code?

1. Mocha
2. Chai
3. Node.js
4. NPM

Self Check: Solution

Which of the below assertion library is used to perform unit testing on JavaScript code?

1. Mocha
2. **Chai**
3. Node.js
4. NPM

Self Check

Identify the correct code that tests whether function with name `displayTotal` is defined?

1. `expect(typeof displayTotal).toBe.Equal("function");`

2. `expect(typeof displayTotal).to.be.Equal("function");`

3. `expect(typeof displayTotal).to.be.equal("function");`

4. `expect(typeof displayTotal).tobe.equal("function");`

Self Check: Solution

Identify the correct code that tests whether function with name displayTotal is defined?

1. `expect(typeof displayTotal).toBe.Equal("function");`

2. `expect(typeof displayTotal).to.be.Equal("function");`

3. **`expect(typeof displayTotal).to.be.equal("function");`**

4. `expect(typeof displayTotal).tobe.equal("function");`