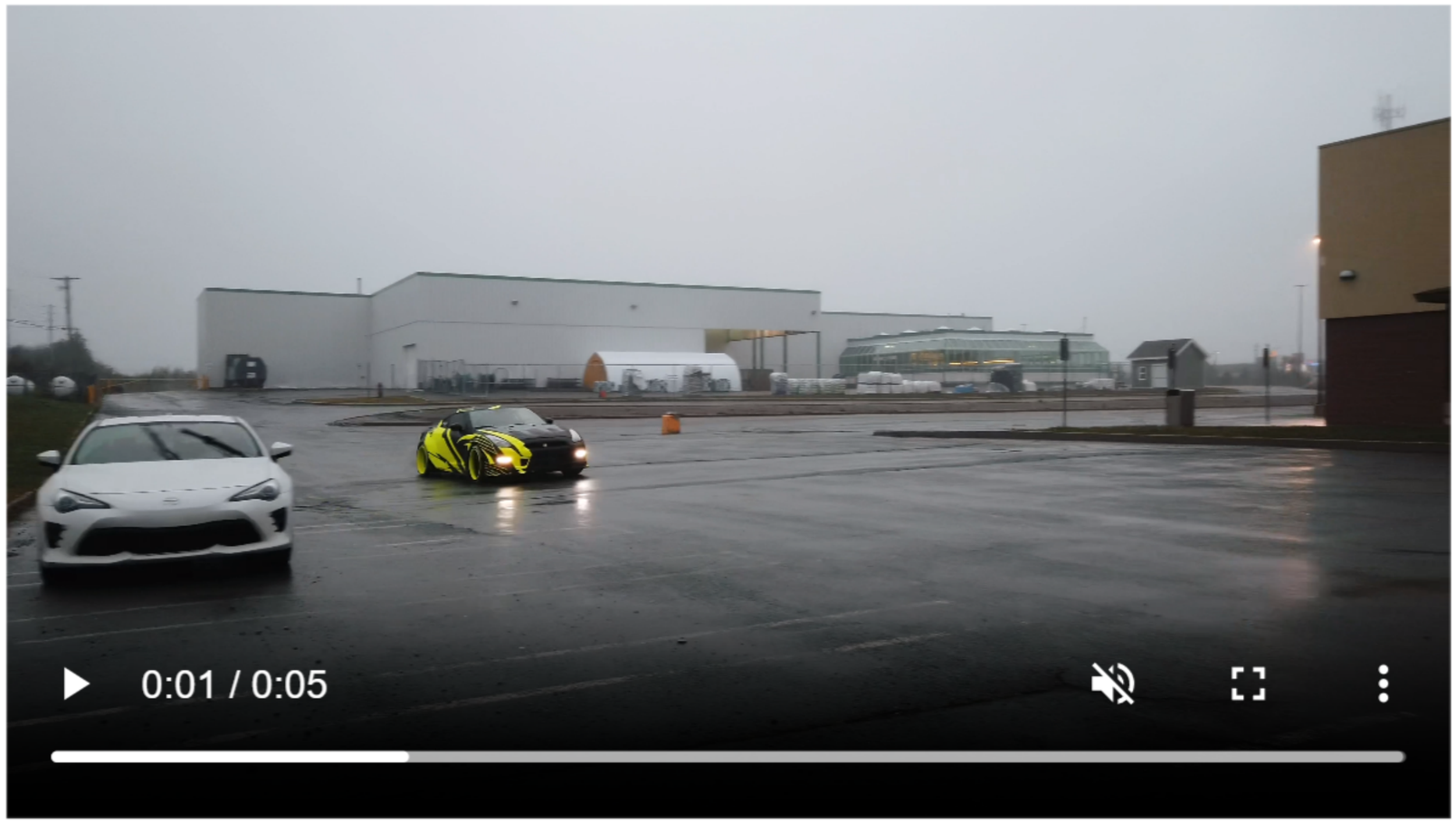
Slide Note

- Before putting a new car out in market, it is tested for various aspects, such as speed, durability, fuel efficiency, safety checks etc.

What Do Car Manufactures Do Before Rolling Out a New Car Model?



Slide Note

A mobile battery manufacturing has to ensure that specifications are provided only after thorough testing.

Breach in doing so can bring bad reputation to business and hence the losses.

What Do Mobile Companies Do…

Before announcing the battery life of its handsets?

Slide Note

Vaccines are life saviors.

But the manufacturing process requires lot of R&D.

The testing is equally crucial and often before releasing it out for humans, it is tested on mice.

Breach in testing can lead to loss of life.

What Do Laboratories Do…

Before announcing vaccine release?

Slide Note

Word of mouth is best publicity for good business returns.

For the restaurant chef, ensuring the customers are served with delicious food consistently is no less stressful.

Before serving food is tasted to ensure it lives up to customer's satisfaction.

Breach in doing so can lead to customer dissatisfaction which can further lead to business loss.

What Do Chefs Do…

- Before serving food dishes to guests?

Slide Note

Should Websites Also Be Tested Before Their Launch?

Slide Note

In October 2014, Filpkart, India based e-commerce giant, sent a note to its customers apologizing for the glitches that took place on the Big Billion Day Sale. The site encountered a heavy rush that it couldn't manage resulting in cancellation of orders, delayed delivery, and much more that was beyond them to manage. While the sale helped the ecommerce giant garner a billion hits in a day, it was certainly a PR nightmare for the brand.

2. In December 2015, UK government found out that its online calculator for estimating the spouse's financial worth got hit with a Form E fault, where calculations went wrong for thousands of couples who had got divorced over the past 20 months. Though the issue was prevalent since April 2014, it got noticed only in December 2015. The damage caused is yet to be estimated.

3. In July 2015, New York Stock Exchange stopped trading due to an undisclosed 'internal technical issue', where all open orders were cancelled, the traders were alerted and informed that they would receive information later. While responding to the shut down, NYSE announced that there was no cyber breach within the system and it resumed operations after 4 hours.

4. Ola, India's largest taxi aggregator faced major security flaws within their system. The software bugs detected helped basic programmers to enjoy unlimited free rides – at the expense of Ola and at the expense of users. The issue went public when customers brought up the weaknesses in the system. Ola tried to fix bugs when the complaints soared up and it was alarming for the brand's reputation in the market.

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 fiascos.

What Can Go Wrong If Websites Malfunction?

October 2014, Flipkart apologizes for Big Billion Day sale fiasco

Source: Livemint.com

Dec 2015, UK government's online calculator glitch

Source: Cio-asia.com

July 2015, New York Stock Exchange halts trading

Source: Money.cnn.com

Ola Mobile App Software Security Flaws Revealed

Source: Economictimes.indiatimes.com

Slide Note

Would You Not Want to Test Your Websites and Prevent Disaster?

Slide Note

Implement Unit Testing Using Mocha and Chai

Slide Note

Learning Objectives

- 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

Slide Note

Software testing is a process

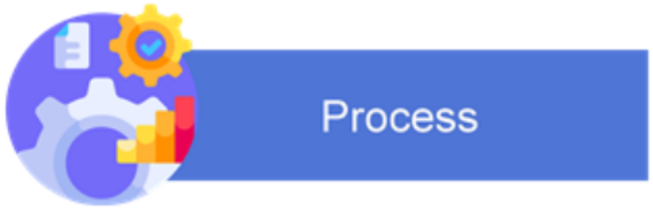
This process verifies and validates the code under test to ensure that it:

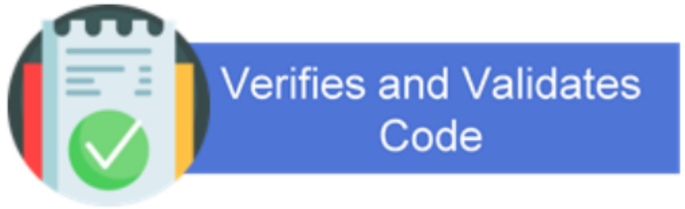
1. is defect or bug free

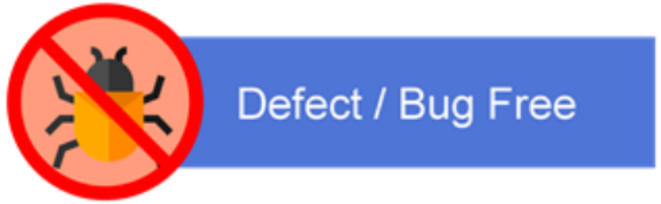
2. meets the requirements stated by the user

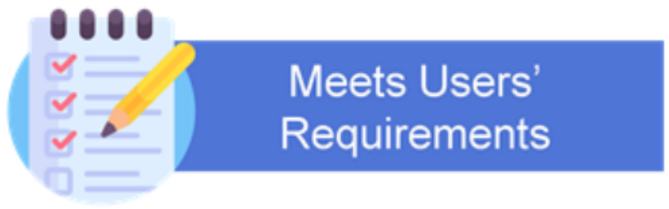
3. complies with the technical specifications laid down in the design document

What is Software Testing?









Menu

00:09:00:00:44

10/ 45

Slide Note

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

Slide Note

Think and Tell

- Software solution code can be quite complex.
- For a quality product, complete solution code should be tested to ensure:
  - Each chunk of code works correctly
  - The different chunks of code when put together, should also work correctly
  - The complete solution should work correctly on the system it is designed and developed for
  - And finally, the solution should be acceptable to the user for whom it is developed
- **How can we ensure that all the above requirements are met by the testing process?**

Slide Note

Examples of

Unit testing: Testing add() function of a calculator app

Integration Testing: Testing all the functions of a calculator app

System testing: Testing calculator app works correctly on desktop browser as well as mobile browser

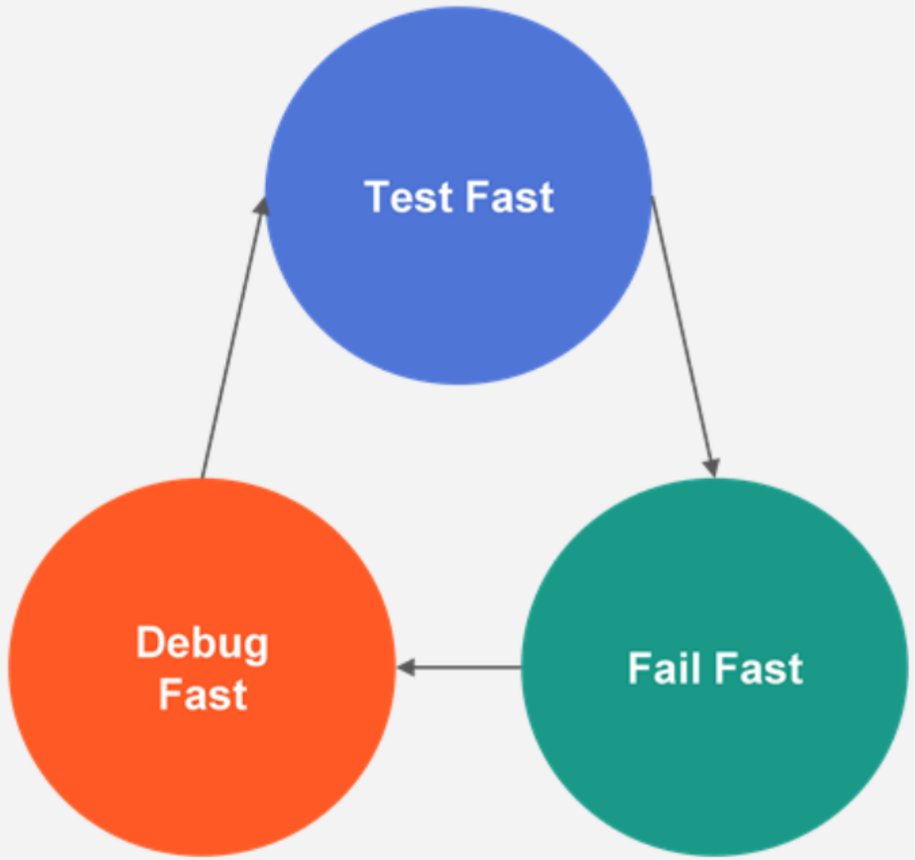
Acceptance testing: Testing performed by end user to ensure calculator app meets all the expected requirements

Levels of Testing

Unit Testing	• Test single unit of code such as a JavaScript function
Integration Testing	• Test group of functions when put together
System Testing	• Test complete software for system compliance
Acceptance Testing	• Test complete software to ensure compliance with the users' requirements

- Software testing should be done at different levels.
- Levels of testing helps make software testing process structured and efficient.

Slide Note



When to Perform Unit Testing?

- Testing is an iterative process
- Testing is performed until the actual results match with the expected results
- For better cost efficiency, first write the 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

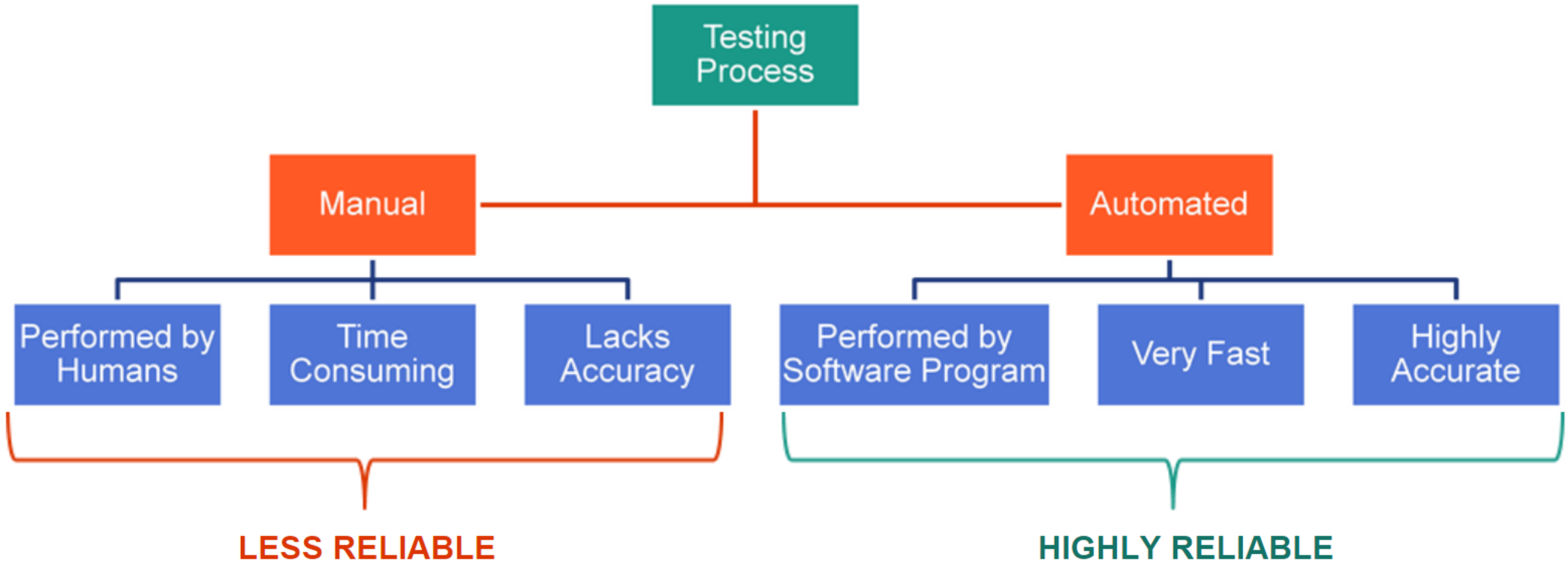
Slide Note

Testing process can be manual or automated.

Manual testing is done by humans which makes it time consuming and lacks accuracy. Hence it is less reliable.

Automated testing is done by software programs and hence will be performed at a high speed, is more accurate and highly reliable.

How to Test?



Slide Note

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

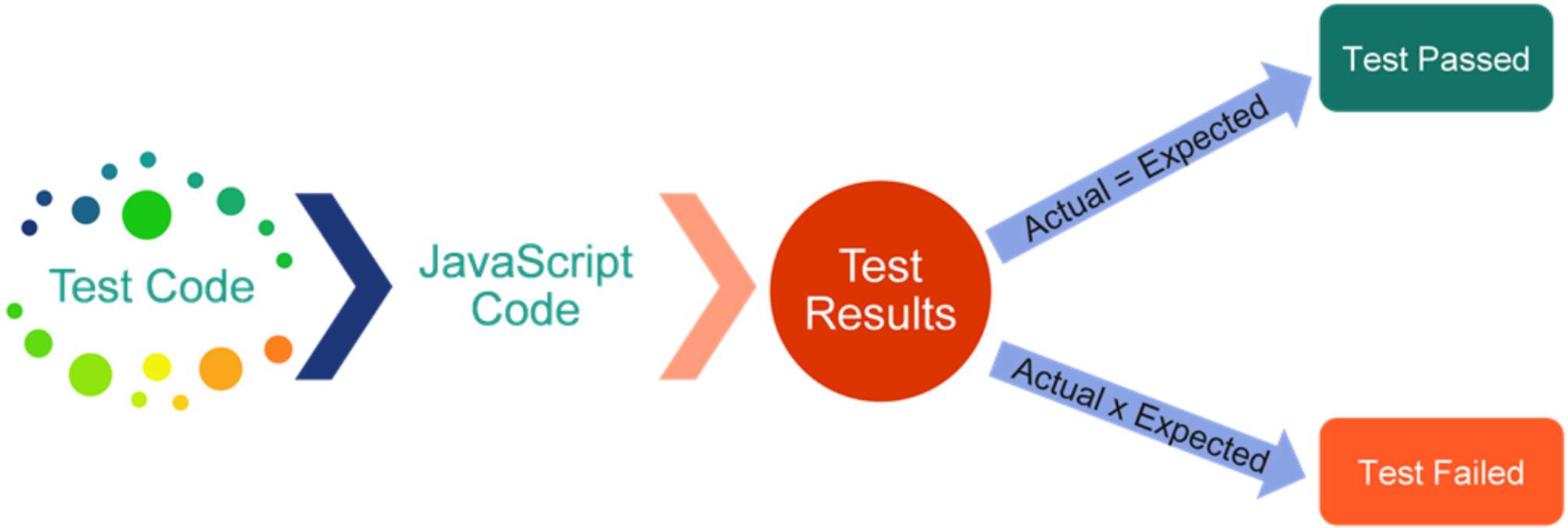
The execution of test code generates test results

Test results state whether actual results match with the expected results

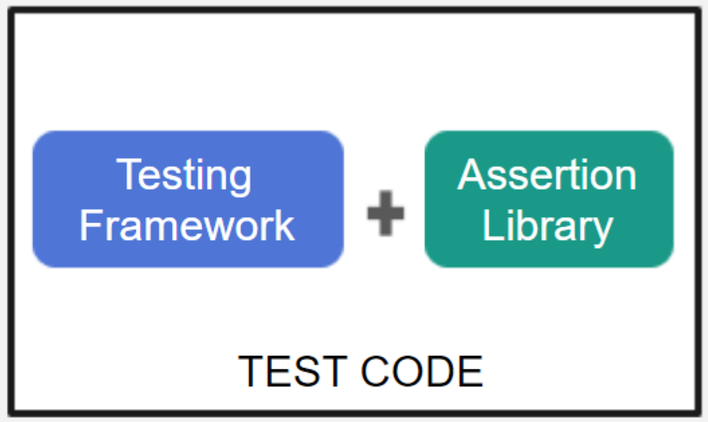
If actual result matches expected result, then the test has passed

If actual result does not match expected result, then the test has failed

How to Automate Unit Testing?



Slide Note



Design Test Code

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

Slide Note

Testing in Action

Trigger Execution

Slide Note

Think and Tell

Which test framework and assertion library should be used to define the test code for testing JavaScript code?

Slide Note

JavaScript code can run at the client side and the server side.

For client-side execution, JavaScript code runs inside browser.

Browser provides necessary runtime environment for execution

For Server-side execution, Node.js provides the necessary runtime environment

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.

Slide Note

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 to write test scripts. These 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.

Slide Note

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.

Slide Note

Create Test Enabled JavaScript Application

Create JavaScript application and install Mocha and Chai to enable automated testing.

Note: Node.js is a pre-requisite for installing Mocha and Chai.

Click here for demo solution.

DEMO

Slide Note

Summarizing Demo

Pre-requisites

install `Node.js`

`npm` tool also gets installed

Command
`npm init -y`

transforms current folder to `npm` based JavaScript project

creates `package.json` file

Command
`npm install mocha chai`

installs `Mocha` and `Chai` and are listed as dependencies in `package.json` file

test property of scripts object in `package.json` should be modified with the value "mocha"

Application folder structure

contains `solution` folder, `test` folder and `package.json` file

`solution` folder contains application source code

`test` folder contains test code

`package.json` file contents

```
{
  "name": "demo-1",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "mocha"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "chai": "^4.3.7",
    "mocha": "^10.1.0"
  }
}
```

Slide Note

JavaScript Solution Code

```
function display() {

}

module.exports = display;
```

Code to export function

- Solution code is the JavaScript code.
- Each function in the solution code should be tested.
- The function should be discoverable to the external code (here, test code).
  - Test code creates reference to the function under test, calls the function and generates actual result.
- Function to be tested should be exported so that it is accessible to the test code.

Slide Note

JavaScript 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

Slide Note

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.

Slide Note

The it() function contains one more assertion statement.

Test code contains 1 or more it() function calls.

Each it() function represents one test case.

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.

Slide Note

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.

Slide Note

An assertion library is a tool to verify things are correct - It's what actually verifies the test results.

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.

Slide Note

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');`

Slide Note

Quick Check

The command to install `Mocha` and `Chai` together is_____.

1. `npm install mocha, chai`

2. `npm install –mocha, -chai`

3. `npm install –mocha –chai`

4. `npm install mocha chai`

Slide Note

Quick Check: Solution

The command to install Mocha and Chai together is_____.

1. `npm install mocha, chai`

2. `npm install –mocha, -chai`

3. `npm install –mocha –chai`

4. **`npm install mocha chai`**

Slide Note

Compute Grades – Enable Testing

A JavaScript program is written that computes grades based on the average marks obtained by the students to help teachers of grades 1 – 9 prepare annual report.

Separate functions are defined for calculating total marks, average scores and determining grades based on the average scores.

Write test cases that test these functions to ensure the code is defect free and generates the expected results.

Click here for demo solution.

DEMO

Slide Note

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.

`Compute Grades` Demo - Test Cases

- The table shown below lists down the test cases written for testing functions of `Compute Grades` demo:

Target	Test Requirement
Solution	Function `calculateTotalMarks()` is defined
	Function `calculateAverateMarks()` is defined
	Function `calculateGrade()` is defined
Function `calculateTotalMarks()`, Function `calculateAverageMarks()`, Function `calculateGrade()`	Function should return expected value for the valid inputs provided

- In all, **8** test cases are written to fulfil all the stated test requirements.

Slide Note

Quick Check

Which of the below testing frameworks can be used to perform unit testing on JavaScript?

1. Mocha

2. Chai

3. Node.js

4. NPM

Slide Note

Quick Check: Solution

Which of the below testing frameworks can be used to perform unit testing on JavaScript?

1. Mocha

2. Chai

3. Node.js

4. NPM

Slide Note

The solution to compute grades has how many functions? - 3

How many test cases were written for each of these functions? - 17

How to determine test cases for the functions? (Answer on the upcoming slide)

Think and Tell

- Were the test cases written to test whether these functions return error messages for insufficient / invalid or incorrect inputs?
- If not, what would happen if these test cases are added?
  - Does the code handle such error conditions?
- Should not the code be refactored to ensure it handles errors for all improper inputs?

Slide Note

Compute Grades – Refactor Code

The existing solution to compute grades is tested to ensure that the required functions exist, and they return expected result for the valid inputs provide.

However, the code needs to be refactored to ensure that these functions return error messages for the invalid inputs passed.

Write test cases and refactor code to ensure that the code provides appropriate error messages for the invalid inputs passed.

Click here for demo solution.

DEMO

Slide Note

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.

`Compute Grades` Demo - Test Cases for Error Handling

- The table shown below, lists down the test cases written for testing functions of `Compute Grades` demo for insufficient / invalid and incorrect inputs:

Target	Test Requirement
Function `calculateTotalMarks()`, Function `calculateAverageMarks()`, Function `calculateGrade()`	Function should return appropriate error message for insufficient inputs.
	Function should return appropriate error message for invalid type of inputs.
	Function should return appropriate error message for negative inputs.

- In all **17** test cases are written to fulfil all the stated test requirements.

Slide Note

Testing Considerations

- Testing is an integral process in software development. It is therefore critical to ensure testing is done thoroughly.
- Ensure all the 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.**

Slide Note

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.

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
    - In the demo, `typeof` operator was used to test this requirement.
  - Generates expected output for the input provided
    - In the demo, each of the functions was called with sample inputs and the actual result was compared with the expected result.
  - Generates appropriate message for the invalid / incorrect / insufficient input provided
    - In the demo, each of the inputs was checked to ensure it was
      - Not undefined (not insufficient)
      - Of type Number (not invalid)
      - Not Negative (not incorrect)

Slide Note

Quick Check

Identify the correct assertion statement from the below given statements:

1. `expect(greetings)=="hello world"`

2. `expect(greetings).to.be("hello world");`

3. `expect(greetings).to.be.equalTo("hello world");`

4. `expect(greetings).to.be.equal("hello world");`

Slide Note

Quick Check: Solution

Identify the correct assertion statement from the below given statements:

1. `expect(greetings)=="hello world"`

2. `expect(greetings).to.be("hello world");`

3. `expect(greetings).to.be.equalTo("hello world");`

4. `expect(greetings).to.be.equal("hello world");`