Developing Secure Software – CMP 6045B/7038B

Lab -Software Testing – Unit testing

**Objectives:**

The aim of this lab is to learn how to learn how to write unit tests in Node with Mocha and Chai, and complete a test plan template. These processes are an essential part of assignment 2’s Technical Design Document.

**Overview:**

You will need to refer to the Week 12 lecture content, which explains about testing. This will enable you to learn how to run unit tests, debug errors and complete a test plan template. It is important you complete this lab as these processes are part of assignment 2.

**Step 1 - Create a new Node project**

* Create a new file called car.js
* Run npm init in a command prompt (visual studio code terminal, or powershell) in the directory with your new js file.
* Follow the npm instructions to set up your project (the default settings should create the project for car.js)
* Install the prompt-sync library by running npm install prompt-sync.
* Type the following code into car.js

import promptSync from 'prompt-sync';

import { fileURLToPath } from "url";

const prompt = promptSync();

export default class Car{

    constructor(speed=0){

        this.speed = speed;

        this.odometer = 0;

        this.time = 0;

    }

    say\_state(){

        console.log("I'm going "+this.speed+"kph!");

    }

    accelerate(){

        this.speed += 5;

    }

    brake(){

        this.speed -= 5;

    }

    step(){

        this.odometer += this.speed;

        this.time += 1;

    }

    average\_speed(){

        if(this.time != 0){

            return this.odometer / this.time;

        }

    }

}

if (process.argv[1] === fileURLToPath(import.meta.url)) {

    let this\_car = new Car();

    console.log("I'm a car!");

    while(true){

        let  action = prompt("What should I do? [A]ccelerate, [B]rake, show [O]dometer, or show average [S]peed?").toUpperCase();

        switch(action){

            case 'A':

                this\_car.accelerate();

                break;

            case 'B':

                this\_car.brake();

                break;

            case 'O':

                console.log("The car has driven "+this\_car.odometer+" kilometres");

                break;

            case 'S':

                console.log("The car's average speed was "+this\_car.average\_speed()+" kph.");

                break;

            default:

                console.log("I don't know how to do that");

                continue;

        }

        this\_car.step();

        this\_car.say\_state();

    }

}

* Text

  Description automatically generatedRun your car.js file by running node car.js in your terminal. Something similar to below should appear in your console when you input A, B, O and S

**Step 2 – Writing and running unit tests**

* Create a new folder called test in the same directory as your car.js file.
* Create a new file in your test folder named car.test.js.
* You now need to install mocha and chai – run npm install mocha and npm install chai to install these into your project.
* Now type the following code in your car.test.js file.

import Car from '../car.js';

import assert from 'assert';

var test\_car = new Car();

describe('Testing initial values of car', function() {

    it("Testing initial speed", function(){

        assert.equal(test\_car.speed, 0);

    });

    it("Testing initial odometer", function(){

        assert.equal(test\_car.odometer, 0);

    });

    it("Testing initial time", function(){

        assert.equal(test\_car.time, 0);

    });

});

describe("Testing acceleration functions of car", function(){

    it("Testing accelerate from 0", function(){

        test\_car.accelerate();

        assert.equal(test\_car.speed, 5);

    });

    it("Testing multiple accelerates", function(){

        test\_car = new Car();

        for(let i=0; i<3; i++){

            test\_car.accelerate(5);

        }

        assert.equal(test\_car.speed, 15);

    });

});

describe("Testing brake functions of car", function(){

    it("Test should not allow negative speed", function(){

        test\_car = new Car();

        test\_car.brake();

        assert.equal(test\_car.speed, 0);

    });

    it("Testing brake once", function(){

        test\_car = new Car();

        test\_car.accelerate();

        test\_car.brake();

        assert.equal(test\_car.speed, 0);

    });

    it("Testing multiple brakes", function(){

        test\_car = new Car();

        for(let i=0; i<5; i++){

            test\_car.accelerate();

        }

        for(let i=0; i<3; i++){

            test\_car.brake();

        }

        assert.equal(test\_car.speed, 10);

    });

    it("Test multiple brakes at zero", function(){

        test\_car = new Car();

        for(let i=0; i<3; i++){

            test\_car.brake();

        }

        assert.equal(test\_car.speed, 0);

    });

});

* Now we need to run our tests using mocha. Mocha will automatically scan the test folder for all unit test files within. Run npm test in your terminal.
* This will show 7 passing 2 failing in the output from your test, as well as details on individually failed tests.

Graphical user interface, text, application, email

Description automatically generated

**Step 3 – Understanding Mocha Output**

* If one or more of your unit tests fail, mocha will give you details of the failed tests.
* When writing your unit tests, you separate your tests with describe messages, and name individual tests with it messages. These messages will show up in the mocha output if a test has failed, enabling you to easily identify it.
* Look at your unit test output. The first failed test should be identified as follows:

Text

Description automatically generated

* Now go to your unit test file, car.test.js. Find the test with the describe message “Testing brake functions of car”, and the name “Test should not allow negative speed”… it should look like this:

Text, letter

Description automatically generated

* Now you know what’s causing the error you need to change brake function in car.js to:

    brake(){

        if(this.speed < 5){

            this.speed = 0;

        }else{

            this.speed -= 5;

        }

    }

* Run npm test again. This time the tests should all pass and show a green pop up box confirming this

Graphical user interface, text, application, email

Description automatically generated

* You have now successfully completed a full unit test and debug process.

**Step 4 – Fill in a Test Plan**

* If you haven’t already done so, download the agile-test-plan-template from Blackboard
* Fill in the test plan for each of the functions in the car.js file
* Remember to show you fixed and re-tested the brake function. Should show as both fail and pass
* Show the completed test plan to your lab leaders so you can get formative feedback

**Step 5 – Add comments to all the above code**

* This will help you understand what the code is doing