Unit Testing and Error Handling

Error Types, Modules, Unit Testing, Mocha & Chai

SoftUni Team

Technical Trainers

Software University

https://softuni.bg

Table of Contents

Have a Question?

Software University

sli.do

#js-advanced

Error Handling

Concepts, Examples, Exceptions

Error Handling

- The fundamental **principle** of error handling says that a function (method) should either:
  - Do what its **name** suggests
  - Indicate a **problem**
  - Any other behavior is **incorrect**

Error Handling

A function failed to do what its name suggests should:

- Return a special value (e.g. **undefined** / **false** / **-1**)
- Throw an **exception** / **error**
- Exceptions are the **object-oriented way** for errors

```
let str = "Hello, SoftUni";
console.log(str.indexOf("Sofia")); // -1
// Special case returns a special value to indicate "not found"
```

Types of Errors

There are **three types** of errors in programming:

- **Syntax Errors** - during parsing
- **Runtime Errors** - occur during execution
  - After compilation, when the application is running
- **Logical Errors** - occur when a mistake has been made in the logic of the script and the expected result is incorrect
  - Also known as bugs

Error Handling – Exceptions (Errors)

- **Exception** - a function is unable to do its work (**fatal error**)

```
let arr = new Array(-1);   // Uncaught RangeError
```

```
let bigArr = new Array(9999999999); // RangeError
```

```
let index = undefined.indexOf("hi"); // TypeError
```

```
console.log(George);   // Uncaught ReferenceError
```

```
console.print('hi');   // Uncaught TypeError
```

Error Handling – Special Values

```
let sqrt = Math.sqrt(-1); // NaN (special value)
```

```
let sub = "hello".substring(2, 1000); // llo
let sub = "hello".substring(-100, 100); // hello
// Soft error - substring still does its job: takes
all available chars
```

```
let invalid = new Date("Christmas"); // Invalid Date
let date = invalid.getDate(); // NaN
```

Problem : Sub Sum

- Sum a **range** of elements in **array** from **startIndex** to **endIndex**
  - Receive three parameters: **array**, **startIndex**, **endIndex**
- Handle **special cases**:
  - First parameter is **not** array → return **NaN**
  - **startIndex** < 0 → assume **startIndex** = 0
  - **endIndex** > **array**.length-1 → assume **endIndex** = **array**.length-1

Solution: Sub Sum

```
function solve(array, startIndex, endIndex) {
  if (Array.isArray(array) == false) {
    return NaN;
  }
  if (startIndex < 0) {startIndex = 0; }
  if (endIndex > array.length - 1) {
    endIndex = array.length - 1;
  }

  return array
    .slice(startIndex, endIndex + 1)
    .map(Number)
    .reduce((acc, x) => acc + x, 0);
}
```

Throwing Errors (Exceptions)

- The **throw** statement lets you create custom errors
  - **General Error** - throw new Error("**Invalid state**")
  - **Range Error** - throw new RangeError("**Invalid index**")
  - **Type Error** - throw new TypeError("**String expected**")
  - **Reference Error** - throw new ReferenceError("**Missing age**")

```
throw new Error('Required');
  // generates an error object with the message
```

Try – Catch

- The **try** statement tests a block of code for **errors**
- The **catch** statement **handles** the error
- **Try** and **catch** come in pairs

```
try {
  // Code that can throw an exception
  // Some other code - not executed in case of error!
} catch (ex) {
  // This code is executed in case of exception
  // Ex holds the info about the exception
}
```

Exception Properties

- An **Error object** with properties is be created

```
try {
    throw new RangeError("Invalid range.");
    console.log("This will not be executed.");
  } catch (ex) {
    console.log("Exception object: " + ex);
    console.log("Type: " + ex.name);
    console.log("Message: " + ex.message);
    console.log("Stack: " + ex.stack);
}
```

Live Demonstration

Lab Problem **2**

Unit Testing

Definition, Structure, Examples, Frameworks

Unit Testing

- A **unit test** is a piece of code that checks whether certain functionality **works as expected**
- Allows developers to see **where** & **why errors occur**

```
function sortNums(arr) {
    arr.sort((a,b) => a - b);
}
```

```
let nums = [2, 15, -2, 4];
sortNums(nums);
if (JSON.stringify(nums) === "[-2,2,4,15]") {
    console.error("They are equal!");
}
```

Unit Testing

- Testing enables the following:
- **Easier maintenance** of the code base
  - Bugs are found ASAP
- **Faster development**
  - The so called "Test-driven development"
  - Tests before code
- **Automated way to find code wrongness**
  - If most of the features have tests, running them shows their correctness

Unit Tests Structure

- The **AAA** Pattern: **Arrange**, **Act**, **Assert**

```
// Arrange all necessary preconditions and inputs
let nums = [2, 15, -2, 4];
// Act on the object or method under test
sortNums(nums);
// Assert that the obtained results are what we expect
if (JSON.stringify(nums) === "[-2,2,4,15]") {
    console.error("They are equal!");
}
```

Unit Testing Frameworks

- JS Unit Testing:
  - Mocha, QUnit, Unit.js, Jasmine
- Assertion frameworks (perform checks):
  - Chai, Assert.js, Should.js
- Mocking frameworks (mocks and stubs):
  - Sinon, JMock, Mockito, Moq

Modules

Definition, Import, Export

Modules

- A **set of functions** to be included in applications
- Group related behavior
- Resolve naming collisions
  - `http.get(url)` and `students.get()`
- Expose only public behavior
  - They do not populate the global scope with unnecessary objects

a module for loading indicator

```
const loading = {

  show() { },
  hide() { },

};
```

Node.js Modules

- **require()** is used to **import** modules

```
const http = require('http');
// For NPM packages
```

```
const myModule = require('./myModule.js');
// For internal modules
```

- **Internal** modules need to be **exported before** being required
- In **Node.js** each file has its own scope

Node.js Modules

- Whatever value has **module.exports** will be the value when using **require**

```
const myModule = () => {...};
module.exports = myModule;
```

- To **export more than one** function, the value of **module.exports** will be an **object**

```
module.exports = {
  toCamelCase: convertToCamelCase,
  toLowerCase: convertToLowerCase
};
```

Unit Testing with Mocha and Chai

Installation, Configuration, Approaches

What is Mocha?

- Feature-rich JS test framework
- Provides common testing functions including **it**, **describe** and the **main function** that runs tests

```
describe("title", function () {
    it("title", function () { … });
});
```

- Usually used together with **Chai**

What is Chai?

- A library with many assertions
- Allows the usage of a lot of different assertions such as **assert.equal**

```
let assert = require("chai").assert;
describe("pow", function() {
   it("2 raised to power 3 is 8", function() {
    assert.equal(pow(2, 3), 8);
  });
});
```

Global Installation

- To install **frameworks** and **libraries globally**, use the CMD
  - Installing **Mocha** and **Chai** through **npm**

```
npm install –g mocha
```

```
npm install –g chai
```

- Check if Mocha is installed

```
mocha --version
```

NODE_PATH Configuration

- By default Node.js does not find its globally installed modules
- You need to set the **NODE_PATH** environment variable

```
rem for any future sessions
setx NODE_PATH %AppData%\npm\node_modules
rem for current session
set NODE_PATH=%AppData%\npm\node_modules
```

- You may need to restart your IDE after changing **NODE_PATH**

Usage and Examples

- To load a library, we need to **require** it

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

```
describe("Test group #1", function () {
    it("should… when…", function () {
        expect(actual).to.be.equal(expected);
    });
    it("should… when…", function () { … });
});
describe("Test group #2", function () {
    it("should… when…", function () {
        expect(actual).to.be.equal(expected);
    });
});
```

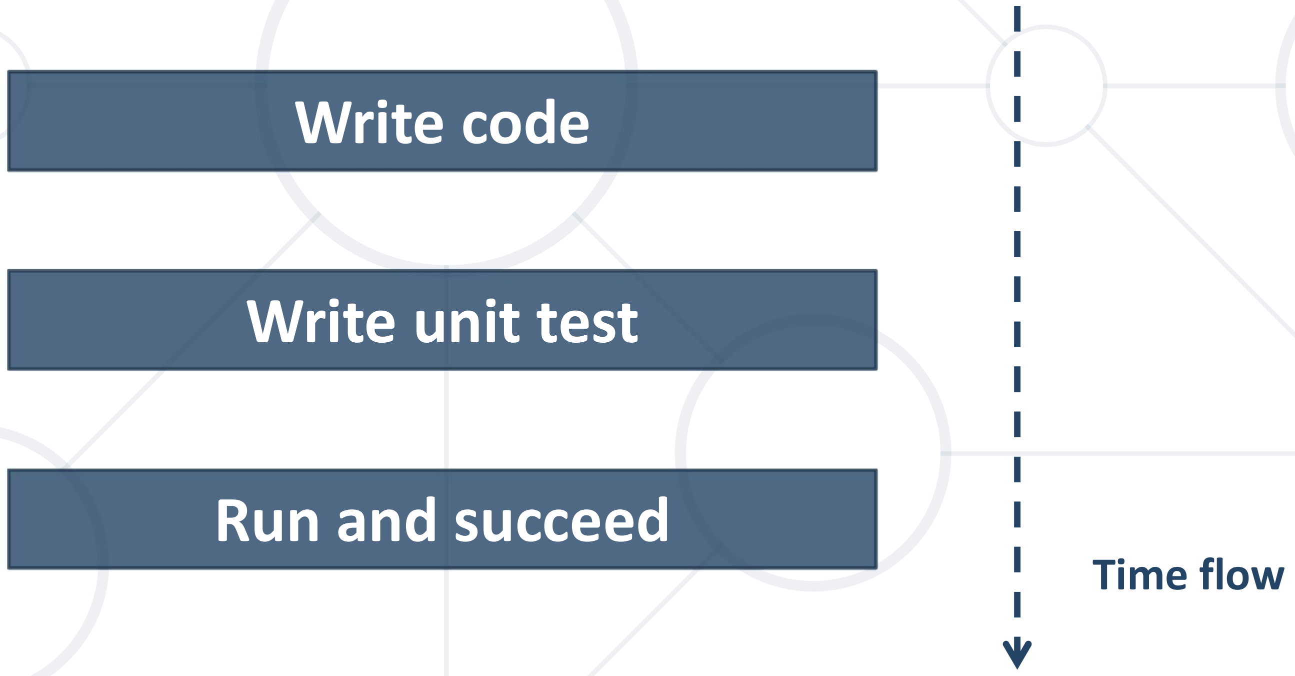
Live Demonstration

Lab Problems **5** and **6**

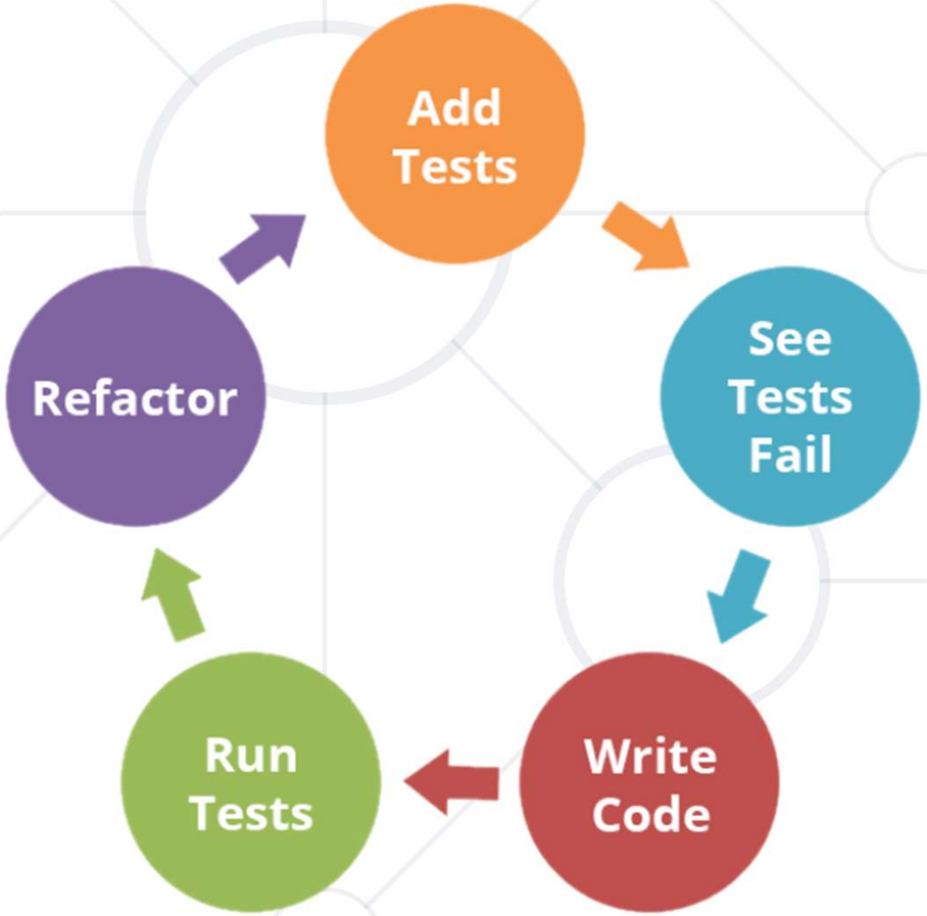
Unit Testing Approaches

- "**Code First**" (code and test) approach
  - Classical approach
- "**Test First**" approach
  - **T**est-**d**riven **d**evelopment (**TDD**)

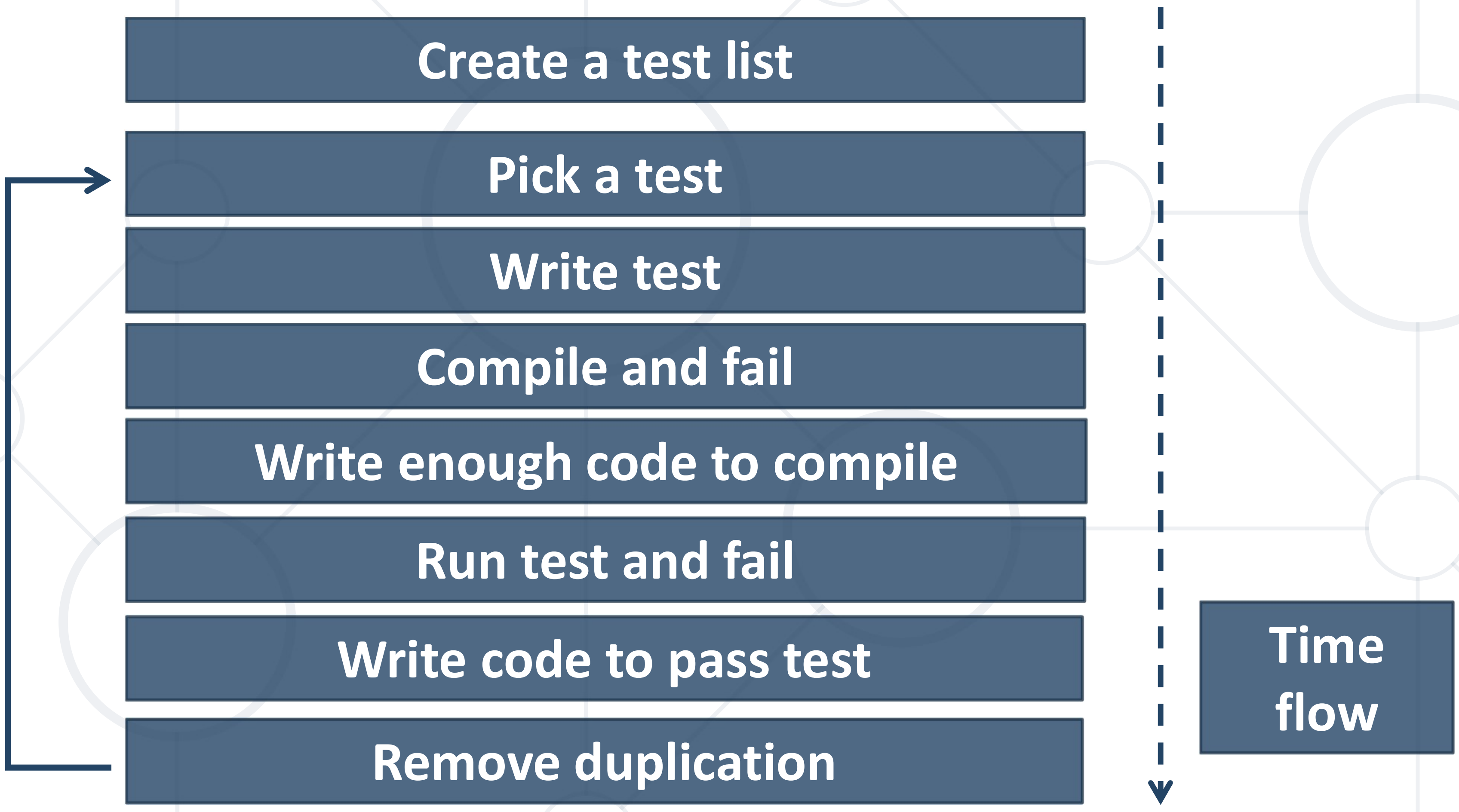
The Code and Test Approach



The Test-Driven Development Approach



Test-Driven Development (TDD)



Why TDD?

- TDD helps find design issues early
  - Avoids reworking
- Writing code to satisfy a test is a focused activity
  - Less chance of error
- Tests will be more comprehensive than if they are written after the code

Summary

- Errors in JavaScript
  - Types & **try/catch** statement
- Modules are a **set of functions** to be included in applications
- Unit tests **check** if certain functionality **works as expected**
- Mocha is a feature-rich **JS testing framework**
- Chain is an **assertion** library
- Different testing approaches

Questions?

Software University

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

License

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg