Unit Testing with JS

Unit Testing, Modules, Mocha & Chai

SoftUni Team

Technical Trainers

Software University

https://softuni.bg

You Have Questions?

Software University

sli.do

#QA-BackEnd

Table of Contents

1. Unit Testing
2. JS Modules
3. Mocha & Chai

Unit Testing

Unit Testing Overview

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

- 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 , Jest
- Assertion frameworks (perform checks)
  - Chai, Assert.js, Should.js
- Mocking frameworks (mocks and stubs)
  - Sinon, JMock, Mockito, Moq

JS 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() { },
};
```

ECMAScript Modules (ESM)

- **ESM** == **official standard format** to package JS code
  - Became standard with ES6 (ECMAScript 2015)
- Uses the **`import`** / **`export`** syntax
- Supports **asynchronous** loading
  - More suitable for modern web development
- Natively supported in browsers
- **Node.js** added **support** for ESM
  - Integration is still evolving

ESM – import

- import is used to import modules

```
import express from 'express'
// For NPM packages
```

```
import { myFunction, myVariable } from './myModule.js'
// For importing specific exports from a an internal file
```

```
import * as myUtils from './utility.js'
// For importing everything from a file as an object
```

ESM – import

- **import** statements are processed **before** the module's code runs
- ESM syntax
  - Default import

```
import defaultExport from 'module-name'
```

  - Named import

```
import { export1 } from 'module-name'
```

  - Import everything

```
import * as name from 'module-name'
```

ESM – export

- **export** is used to **expose items** from a module

```
export const myVariable = 42;
// Exporting a constant
```

```
export function myFunction() {…}
// Exporting a function
```

```
export default class MyClass {…}
// Exporting a class as the default export
```

ESM – export

- When the **imported value changes** in the **exporting module**, it also **updates** in the **importing module**
- ESM syntax
  - Default export

```
export default myFunctionOrClass;
```

  - Named export

```
export { myFunctionOrClass };
```

  - Aggregating modules (doesn't include the default export)

```
export * from 'module-name';
```

CommonJS

- **CommonJS** == **official standard format** to package JS code
  - Older, but still **widely used**
    - Especially in existing Node.js projects
- Uses the **`require()`** / **`module.exports`** syntax
- Supports **synchronous** loading
  - Modules are loaded one by one
- **Transitioning** from CommonJS to ESM takes **time** and **effort**
  - There are still dependencies **only** available as CommonJS modules

CommonJS – require()

- **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

CommonJS – module.exports

- 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
};
```

package.json

- Serves as a **manifest**
  - Organizes the project's **metadata**
    - Project's name
    - Project's version
    - Etc.
  - Manages its **dependencies**
    - Lists the **packages** the project uses
      - Specifies versions
  - Lists all **scripts** that the project needs

dependencies vs devDependencies

- **`dependencies`**
  - **Libraries** that are necessary for the app to run and function correctly in **production**
    - Frameworks
    - Utility libraries

- **`devDependencies`**
  - **Libraries** that are necessary for the app **development**
    - Testing frameworks
    - Build tools
  - Not included in production build

Managing Dependencies and Versions

- **package.json** is used for specifying versions of each package
  - Uses semantic versioning (semver) syntax
    - Three-part version notation `Major.Minor.Patch`
- Specify exact versions or use symbols to allow for updates
  - `"libraryName": "1.0.0"` → pins the version to exactly 1.0.0
  - `"libraryName": "^1.0.0"` → allows updates to any 1.x.x version
  - `"libraryName": "~1.0.0"` → allows updates to any 1.0.x version

Installing Libraries with NPM

- To install a library and add it to the **'dependencies'** in the package.json, open the **terminal** in VS Code and write the following command

```
npm install <library_name> --save
```

- To install a library as a **development dependency**, use the following command

```
npm install <library_name> --save-dev
```

- Running these commands, **modifies** the **package.json** file

Mocha and Chai

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);
  });
});
```

Installation

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

```
npm init -y
```

```
npm install chai
```

```
npm install mocha
```

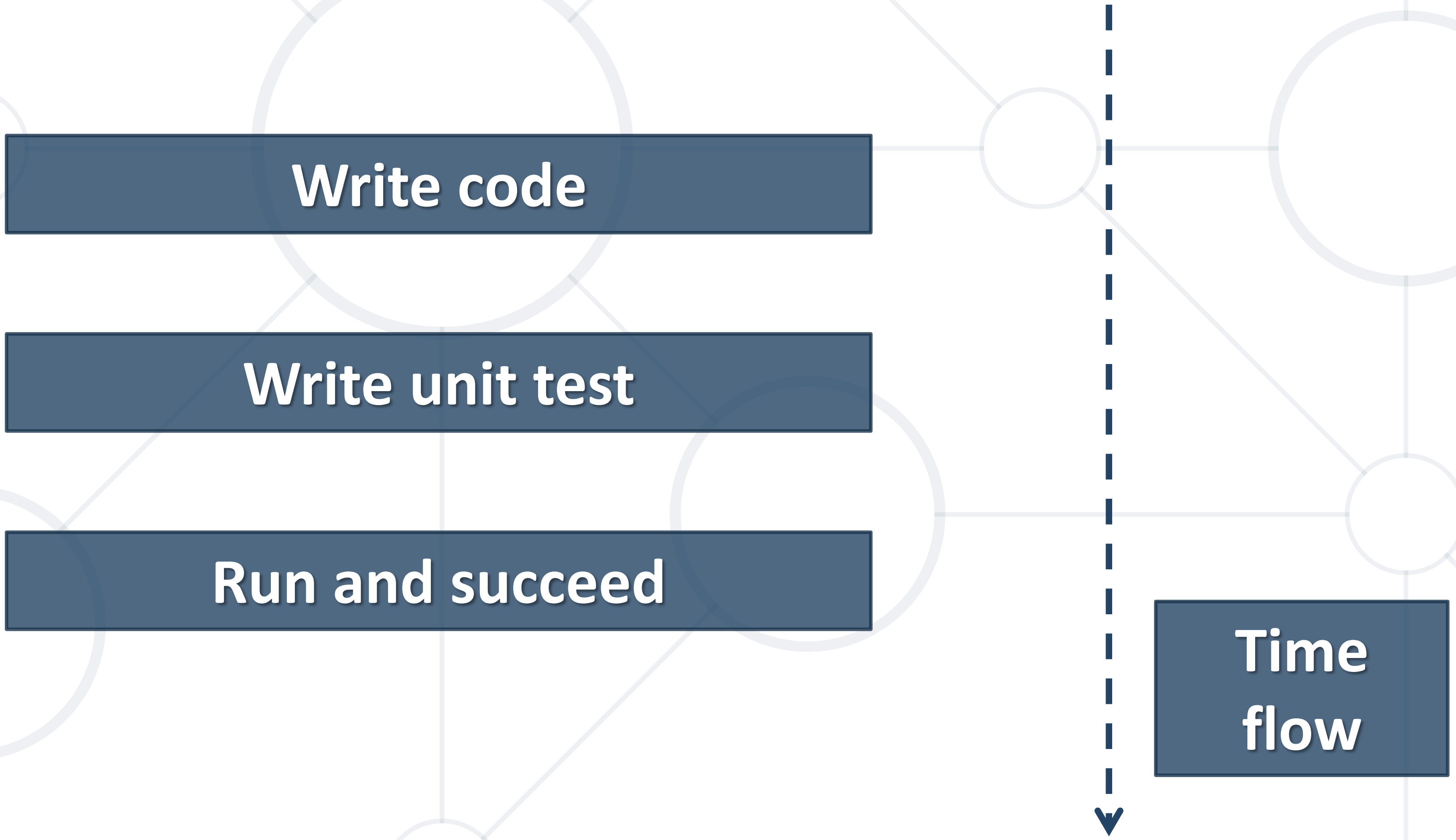
```
npm init -y
```

```
npm i chai mocha
```

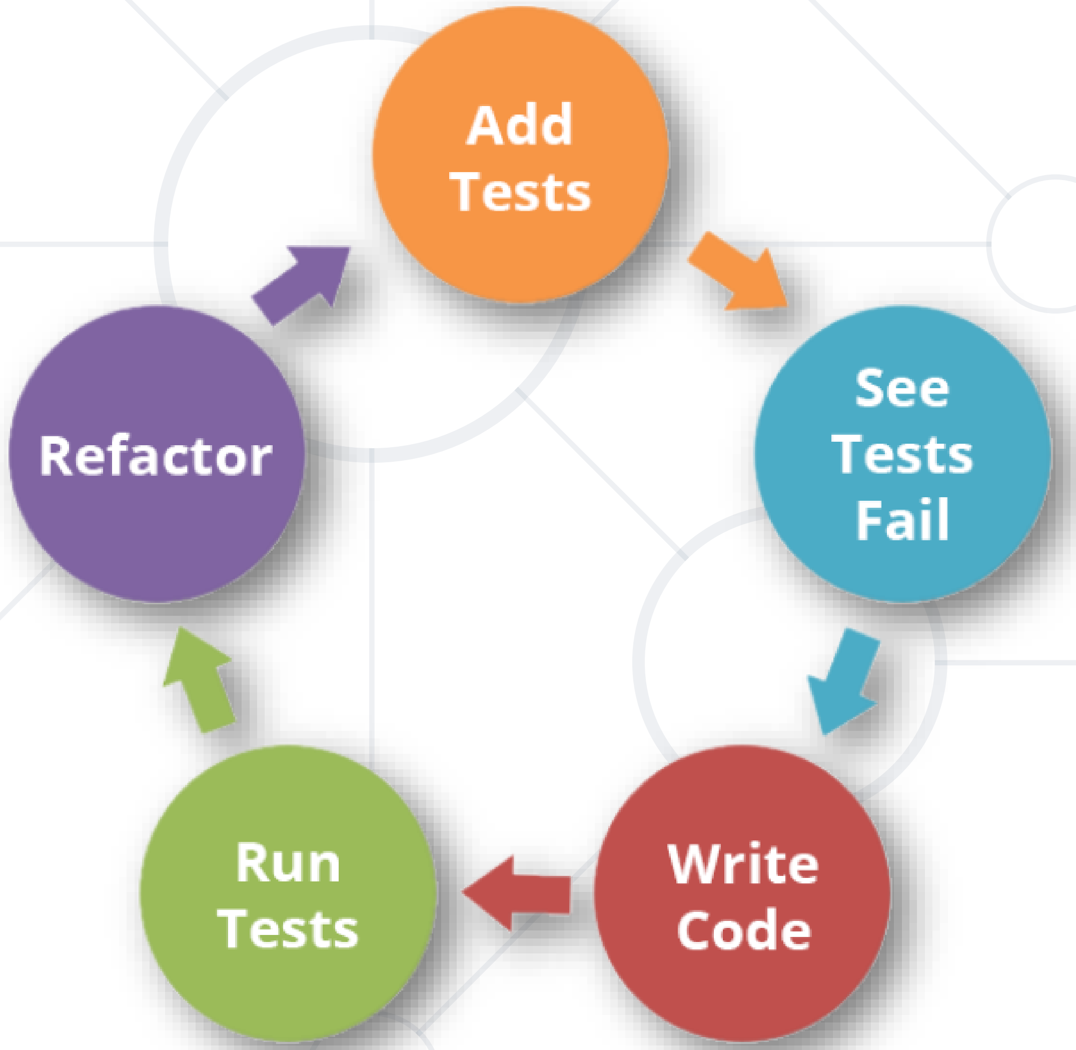
Unit Testing Approaches

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

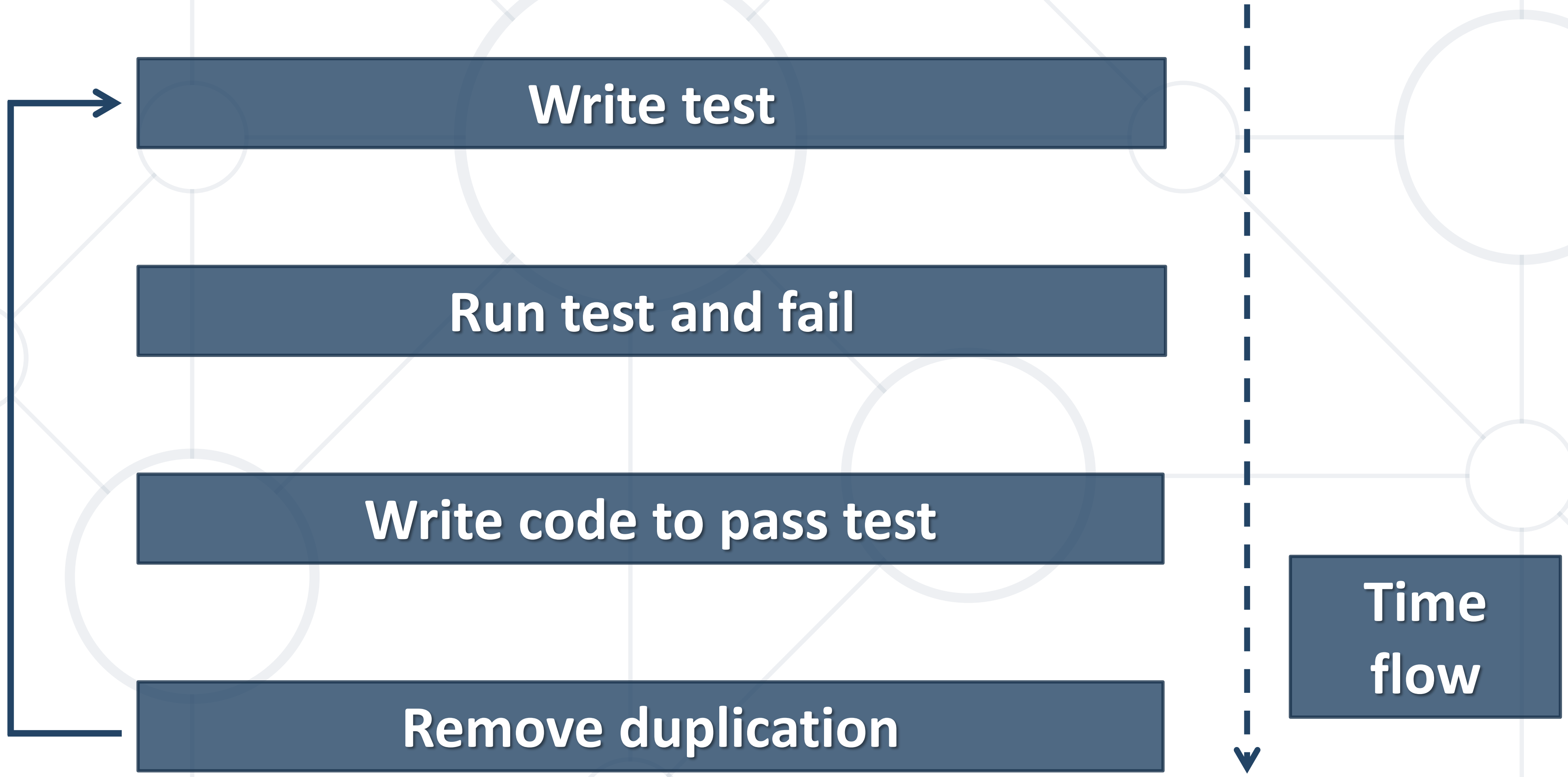
The Code and Test Approach



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

Behavior-Driven Development

- **B**ehavior-**D**riven **D**evelopment (**BDD**) extends TDD
  - Focuses on the system's behavior from the user's perspective
  - Translates the behavior into specifications, using **`describe`** and **`it`** blocks
  - BDD makes tests more readable and user-focused
    - Writing tests that reflect the expected behavior of the application
- Mocha and Chai **incorporate** the **BDD approach**

Live Demo

Lab

Summary

- Modules are a set of functions to be included in applications
- ESM and CommonJS modules
- package.json
- 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

Software University

Questions?

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

СофтУни БУДИТЕЛ

Finance Academy

SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

Решения за твоето утре

DRAFT KINGS

THE CROWN IS YOURS

VIVACOM

AMBITIONED

INDEAVR

Serving the high achievers

PHAR VISION

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

License