МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ

КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ  
  
  
  
  
  
  
  
  
  
Дисципліна «Автоматизоване тестування»

Лабораторна робота № 2

Виконала: студентка гр. 8.1213  
 Абдуллаєва Севда   
 Самандар Кизи

Перевірив: Лебідь Максим   
 Володимирович

Запоріжжя  
2023

**Завдання**

1. Реалізувати підхід TDD при розробці бібліотеки операцій над матрицями.
2. Для модульного тестування використовувати засоби Mocha, Chai.
3. Створити github репозиторій з відображенням процесу TDD.

Підхід Test-Driven Development (TDD) є доброю практикою при розробці бібліотеки операцій над матрицями, оскільки він допомагає забезпечити високу якість та надійність коду, зменшує кількість помилок та полегшує роботу з бібліотекою. Підхід TDD допоможе забезпечити те, що бібліотека працюватиме, як очікується, та те, що вона залишається стабільною при розвитку і змінах.

**Код програми:**

function addMatrices(matrix1, matrix2) {

    const result = [];

    for (let i = 0; i < matrix1.length; i++) {

        const row = [];

        for (let j = 0; j < matrix1[i].length; j++) {

            row.push(matrix1[i][j] + matrix2[i][j]);

        }

        result.push(row);

    }

    return result;

}

module.exports = {

    addMatrices

};

const { expect } = require('chai');

const { addMatrices } = require('../src/matrixopera');

describe('Matrix Operations', () => {

    it('should add two matrices', () => {

        const matrix1 = [[1, 2], [3, 4]];

        const matrix2 = [[5, 6], [7, 8]];

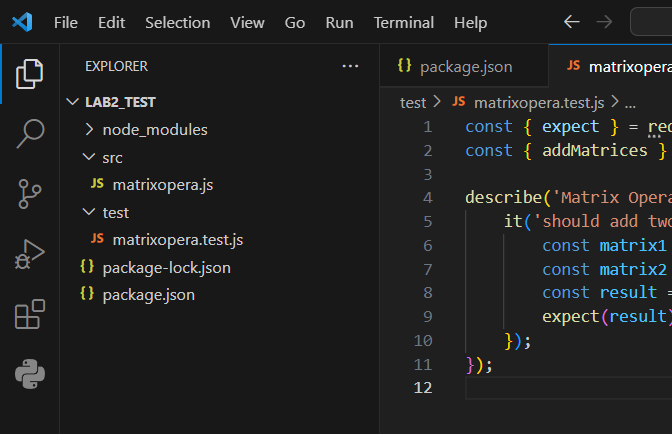
        const result = addMatrices(matrix1, matrix2);

        expect(result).to.deep.equal([[6, 8], [10, 12]]);

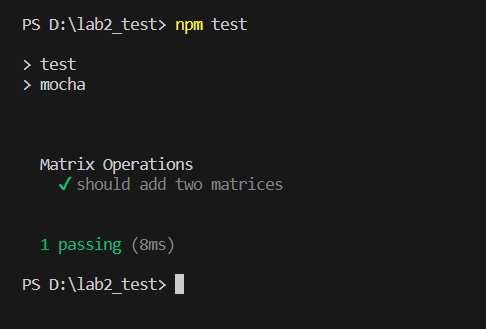
    });

});

**Структура файлів:**

****

**Результати роботи програми:**

****

**Посилання на репозиторій:**

<https://github.com/SevdaAbdullaeva/TDD_lab2.git>

**Контрольні запитання**

1. **Наведіть особливості TDD підходу.**

Test-Driven Development (TDD) - це методологія розробки програмного забезпечення, яка передбачає виконання тестування перед написанням коду функціональності. Основні особливості TDD включають:

* Цикл тестування-розробки-рефакторингу: TDD базується на коротких ітераціях, які передбачають створення тесту, написання коду, необхідного для проходження цього тесту, і потім рефакторингу цього коду. Цей цикл повторюється для кожної нової функціональності.
* Спрощення проблем: TDD допомагає розробникам розглядати проблему як послідовність тестів і обговорення їх замість намагання вирішити всі аспекти проблеми одразу. Це сприяє розкладанню завдання на менші, керовані тестами кроки.
* Відсутність зайвого коду: TDD спонукає розробників писати код тільки для вирішення конкретних тестів. Це допомагає уникнути зайвого або непотрібного коду.
* Забезпечення належної якості: Тести, що створюються під час TDD, служать в якості автоматизованого тестування для перевірки функціональності програми, і це допомагає запобігти виходу на ринок програмного продукту з помилками.
* Рефакторинг: Після написання коду, що проходить тест, розробник може виконати рефакторинг для поліпшення якості коду без страху внести зміни, які порушать функціональність, оскільки тести надійно перевіряють її стабільність.
* Підтримка документації: Тести, що створюються під час TDD, можуть служити в якості специфікації для функціональності програми, що полегшує розуміння, як користувачам, так і розробникам.
* Впровадження змін: TDD полегшує внесення змін у програму, оскільки наявні тести допомагають виявити помилки після внесення змін, і це сприяє збереженню стабільності програми під час розвитку.

Загалом, TDD сприяє покращенню якості програмного забезпечення та спрощує розробку через систематичний тестовий підхід та ітеративність.

1. **Як модульне тестування використовується у методології TDD?**

* Модульне тестування (unit testing) є важливою складовою методології Test-Driven Development (TDD). В методології TDD розробка починається з написання тестів для індивідуальних модулів (функцій або класів) програми перед написанням самого коду модуля. Ось як модульне тестування використовується у TDD:
* Визначення функціональності: Розробник визначає, яку функціональність повинен мати модуль (функція або клас), який він збирається розробляти.
* Написання тесту: Розробник створює тест, який перевіряє очікуваний результат для цієї функціональності. Тест має бути автоматизованим і повинен бути простим і конкретним.
* Запуск тесту: Перед написанням самої функціональності, розробник запускає тест. Як очікується, тест не пройде, оскільки функціональність ще не реалізована.
* Написання коду: Розробник розробляє мінімально необхідний код, щоб зробити тест пройденим. Мета полягає в тому, щоб код виконував лише те, що необхідно для проходження тесту, без зайвих функцій.
* Повторення: Розробник перезапускає тест. Якщо тест пройшов успішно, це означає, що розроблена функціональність працює як очікується.
* Рефакторинг: Після успішного проходження тесту розробник може виконати рефакторинг коду, щоб покращити якість та читабельність, при цьому забезпечуючи, що тести залишаються пройдені.
* Повторення для інших модулів: Цей цикл повторюється для кожного іншого модуля, який розробляється в рамках проекту.

Модульні тести створюються для кожного окремого модуля або функції, і вони допомагають забезпечити, що кожен модуль працює коректно. Це дозволяє ізолювати помилки і забезпечити надійність програми в цілому. Коли всі модульні тести проходять успішно, це дає впевненість в тому, що програма в цілому працює правильно.

1. **Як враховується специфікація програмного забезпечення та на якому етапі?**

Специфікація програмного забезпечення є важливим етапом у розробці програми і вона враховується на різних етапах розробки. Специфікація програмного забезпечення (іноді називається технічною специфікацією) описує те, що програма має робити, і як це має бути зроблено. Вона враховується наступним чином:

* Етап аналізу (Analysis): Специфікація розробляється або уточнюється на цьому етапі. Розробники та бізнес-аналітики взаємодіють, щоб визначити вимоги до програми, описати функціональність і негласні правила.
* Етап проектування (Design): Після визначення вимог програмація насправді не починається. На цьому етапі відбувається проектування програми. Це означає визначення архітектури, інтерфейсів та інших технічних аспектів програми. Специфікація використовується для визначення того, як програма буде виглядати і як буде взаємодіяти з іншими системами.
* Етап розробки (Development): На цьому етапі розробники використовують специфікацію для написання коду програми. Вони використовують специфікацію для визначення функціональних і нефункціональних вимог, а також для побудови тестів, які перевіряють відповідність реалізації вимогам.
* Етап тестування (Testing): На цьому етапі тести використовуються для перевірки того, чи відповідає програма специфікації. Якщо програма проходить всі тести, це означає, що вона відповідає специфікації.
* Етап супроводу (Maintenance): Після випуску програми в експлуатацію специфікація слугує як основа для подальших змін і покращень програми. Вона допомагає розробникам розуміти, які зміни можуть бути внесені без порушення існуючої функціональності.

Специфікація програмного забезпечення є важливим інструментом для забезпечення відповідності розробки програми її вимогам, зменшення ризику помилок та забезпечення зрозумілості для всіх учасників процесу розробки.