Автор:
Омельчук Ніка Романівна
Спеціальність 121 «Інженерія програмного забезпечення»
студентка 42 ІПЗ групи

Науковий керівник:
Старший викладач кафедри комп'ютерної та програмної інженерії
Фещенко Богдан Петрович

РОЗРОБКА UNIT-ТЕСТІВ ДЛЯ ВЛАСНОЇ МОВИ ПРОГРАМУВАННЯ

Анотація. Тестування є дійсно важливим етапом розробки, адже воно допомагає знайти помилки і недоліки в коді, та виправити їх ще до того, як програма вийде в світ. Для того, щоб дійсно якісно перевірити програму, бажано використовувати декілька видів тестування, один з яких називається модульним, та включає в себе розробку unit-тестів. Вони автоматизовано перевіряють правильність роботи кожної окремої частини коду, не торкаючись решти програми.

Ключові слова: розробка програмного забезпечення, тестування, мова програмування, функції, синтаксис, Unit-тести.

Вступ. Модульне тестування — це процес перевірки роботи окремих частин програми, які називають модулями.

На сьогоднішній день модульне тестування є досить актуальним у розробці програмного забезпечення, оскільки якість програм напряму залежить від правильного функціонування та відповідності наданим вимогам, а за допомогою даних тестів є змога це перевірити. Використання компонентного тестування дозволяє знаходити помилки ще на ранніх етапах розробки ПЗ, коли їх набагато легше виправити.

Крім того, дане тестування автоматизовано перевіряє окремі частинки коду, допомагаючи нам впевнитися, що кожен модуль виконує свою функцію правильно. Саме завдяки такому розбиттю тестів можна легше побачити, в якій саме частині проекту знаходиться помилка, та яким чином її можна виправити.

Постановка задачі. Оскільки задача даного дослідження полягає у розробці модульних тестів до власної мови програмування, то варто звернутися до інформаційних джерел та досліджень, які надають необхідні відомості щодо створення unit-текстів та правильного додавання їх у саме середовище розробки. Опираючись на ці відомості вже можна розпочинати створення тестів. В результаті дослідження маємо отримати розроблені модульні тести до мови програмування UaNiki, які будуть подані у вигляді бібліотеки класів вбудованої у сам програмний продукт.

Метою дослідження є розробка та впровадження модульних тестів для перевірки окремих частин програмного коду.

Основна частина. Загалом тестування являє собою процес перевірки роботи програми, який дозволяє нам переконатися, що всі її частини працюють коректно. Це дійсно важливий етап розробки, адже він допомагає знайти помилки та недоліки в коді та виправити їх до того, як відбудеться реліз програмного засобу. Його метою є виявлення помилок у роботі програмного забезпечення, їх виправлення, а також забезпечення стабільної та якісної роботи системи.

Ідея модульного тестування полягає в тому, щоб перевіряти кожну частину окремо, і таким чином переконатися, що функції працюють правильно самі по собі, без взаємодії з іншими модулями. Наприклад, таким чином можна виявити проблеми з обчисленнями, умовами, циклами або неправильним використанням змінних. Такий підхід є дуже важливим, тому що він дозволяє знайти й виправити більшість помилок ще до того, як

програма буде повністю зібрана. Це робить розробку більш ефективною та підвищує якість кінцевого продукту.

Ще однією особливістю даного тестування є те, що воно автоматизоване, тобто більшість процесів виконуються спеціальними програмами без постійної участі людини. Завдяки автоматизації можна значно зменшити час тестування і отримати швидкий та якісний результат. Крім того, автоматизовані тести виконуються точно й без помилок, які часто виникають під час ручного тестування.

Модульні тести зазвичай створюють як окремий проєкт поруч із основним. У методах тестування є конкретний набір вхідних даних, які є фіксованими. Це мають бути тестові приклади того, що є на вході та що має бути на виході.

Щоб створити unit-тести, в програмі-компілятор «UaNiki», створюються проєкт модульного тестування. Даний додаток матиме назву «UaNiki Tests». У тестовому проєкті потрібно створити unit-тести, додати необхідні бібліотеки, та написати тестові класи та методи з очікуваними результатами.

Коли проєкт створено, автоматично додається один клас та один метод для тестування. Для того, щоб наші класи і методи відобразились у списку тестів, вони мають бути відмічені атрибутами «[TestClass]» та «[TestMethod]».

Виконувати перевірку результату будемо за допомогою об’єкта «Assert», що має багато методів для проведення перевірок. Основний із вбудованих методів, який було використано для тестування, це метод «Assert.AreEqual». Даний метод має багато перевантажень залежно від вхідних параметрів: очікуваних, фактичних і повідомлення, яке буде виведено, якщо тест буде негативним.

В нашому випадку можна використати його для перевірки коректності компіляції коду. Маємо розроблюваний тестовий метод «Compile_PrintHelloWord()», в якому ми будемо тестувати компілювання найпростішого коду - виведення на екран фрази «Привіт світ». Для цього ми спочатку створюємо змінну comp (скорочення від слова компілятор), в яку передається клас мови програмування «UaNiki» - var comp = new UaNiKi_Class.UaNiKi();, та змінну code з кодом, що необхідно скомпілювати (див. рис. 1).

```
string code = @"
головний метод
{
написати(""Привіт світ"")
}
";
```

Рисунок 1 – Змінна code

Далі створюємо змінну exOut (expected output) - string exOut = "Привіт світ\n"; , яка зберігає правильний очікуваний результат, що має вивести результат після виконання. В даному випадку це текст "Привіт світ" і перехід на новий рядок (\n). І наостанок ми маємо створити змінну res (result), в якій викликається основний метод з бібліотеки UaNiKi - Compile, що повертає результат обробки коду у вигляді інформації виведеної у консоль - string res = comp.Compile(code);. Після цього викликається метод Assert.AreEqual(exOut, res); , в який ми передаємо наші змінні, і порівнюємо очікуване з результатом.

Також є ще й інший метод написання тестів, в якому передаються змінні «code» та «res» через атрибути. Таким чином можна записати декілька значень змінних, що зробить код більш лаконічним та допоможе ще більш ефективно протестувати програму.

Таким чином можна створити тестові рішення із різними вхідними даними для виклику методів і перевірки їхньої коректності, та чим більше тестових наборів, тим ефективніше.

Висновок. За результатами виконаної роботи було створено проєкт з unit-тестами для власної мови програмування «UaNiki». Завдяки модульному тестуванню вдалося

перевірити, коректність роботи окремих частини програмного засобу. Дана стратегія допомагає підвищити ефективність тестування.

Для реалізації даної стратегії було використано мову програмування C# та створено окремий проєкт із тестами, в якому розроблені класи та методи що перевіряють основні функції «UaNiki». Наведеним прикладом було висвітлено тестування команди виводу словосполучення «Привіт світ».

Під час розробки, були покращені знання та навички, що до створення unit-тестів. Дані тести - це зручний і ефективний спосіб перевірити якість коду. Вони допомагають знайти помилки заздалегідь, полегшують перевірку програми та дають впевненість, що кожна частина працює як було заплановано.

В результаті створені тести, змогли перевірити код програми, і тим самим дали змогу переконатися, що компіляція, змінні, математичні операції, цикли, умови, коментарі та функції працюють коректно.

Список літератури

1. Омельчук Н. Р. "СТВОРЕННЯ ВЛАСНОЇ МОВИ ПРОГРАМУВАННЯ ТА КОМПІЛЯТОРА ДЛЯ НЕЇ." *Інформаційно-комунікаційні технології в освіті* 12. 2024.
2. Омельчук Н. Р. "СТВОРЕННЯ ВЛАСНОЇ МОВИ ПРОГРАМУВАННЯ." *Інформаційно-комунікаційні технології в освіті* 13. 2024.
3. Get started with unit testing - Visual Studio (Windows). Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/visualstudio/test/getting-started-with-unit-testing?view=vs-2022&tabs=dotnet,mstest
4. Test a .NET class library using Visual Studio - .NET. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/dotnet/core/tutorials/testing-library-with-visual-studio