ДОДАТОК А

Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії

6. Shubin I. , Kozyriev A. Method for Solving Quantifier Linear Equations for Formation of Optimal Queries to Databases. 2023 7th International Conference on Computational Linguistics and Intelligent Systems, Computational Linguistics Workshop, Kharkiv,Ukraine,8-20 April 2023. 21 April 2023.,URL: https://doi.org/10.1109/picst47496.2019.9061407 (дата звернення: 20.04.2024).

8. Hybrid system of computational intelligence based on bagging and group method of data handling / Y. Bodyanskiy et al. System research and information technologies. 2024. No. 1. P. 75–85. URL: https://doi.org/10.20535/srit.2308-8893.2024.1.06 (date of access: 24.04.2024).

12. Models of adaptive integration of weighted interval data in tasks of predictive expert assessment / I. Ruban et al. Eastern-European Journal of Enterprise Technologies. 2022. Vol. 5, no. 4(119). P. 6–15. URL: https://doi.org/10.15587/1729-4061.2022.265782 (date of access: 06.05.2024).

ДОДАТОК Б

Слайди презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

NURE ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Дослідження методів механізмів тестування та мокування(Mocking)

Богун Владислав Миколайович., ІПЗм-22-5

Науковий керівник: доц. Голян В.В.

SE software engineering

1

Дослідження

Актуальність та стан розвитку галузі: Розробники можуть не розуміти всі різновиди тестування, такі як тестування одиниць, інтеграційне тестування, прийомочне тестування та інше. Це може призвести до невірного вибору фреймворку для конкретного виду тестування.

Чітке визначення напряму дослідження: Напрям дослідження полягає у вивченні методів створення механізмів тестування та мокування (Mocking) програмного забезпечення на платформі .Net. Це дослідження має на меті вирішення труднощів, пов'язаних з вибором підходу та стратегії тестування програмного забезпечення.

Об’єкт дослідження: Об’єктом дослідження є методи створення механізмів тестування та мокування (Mocking) програмного забезпечення.

SE software engineering

2

Постановка задачі

У ході даної роботи нами буде досліджено предметну область побудови тестових механізмів за допомогою фреймворків та підходів до тестування на платформі .Net. Ми проведемо порівняльний аналіз та аналіз призначення кожного фреймворку та підходу до тестування. Ця робота спрямована на створення дорожньої карти для розробників щодо знаходження найефективнішого та комплексного підходу до тестування застосунків. Реалізація наукового дослідження складається з наступних етапів:

- аналіз предметної області;аналіз методів тестування;
- розглядання особливостей NUnit, xUnit, MSTest;
- порівняння атрибутів методів побудови механізмів тестування;
- аналіз мокування;
- формування висновків.

Методологія

Опис використаних методів дослідження: Дослідження проводилось за багатьма матеріалами в мережі та аналізувались патерни і результати, також ми задіяли персональні навички та досвід у побудові архітектури подібних систем та їх використанні.

Інструментарій та технології, використані в роботі:

- Платформи: .Net
- Мова програмування: C#
- Фреймворки для тестування: NUnit, xUnit, MSTest
- Бібліотеки для мокування: Moq, NSubstitute, FakeItEasy

Дослідження проблематики

- У сучасній розробці програмного забезпечення існує велика кількість фреймворків для тестування, кожен з яких має свої переваги та недоліки залежно від специфіки проекту.
- Розробники часто стикаються з проблемою вибору правильного інструменту для конкретного типу тестування, наприклад, юніт-тестування або інтеграційного тестування.
- Крім того, надійність фреймворків залежить від активності спільноти та можливості інтеграції з іншими інструментами, такими як CI/CD системи. Неправильний вибір інструментів може призвести до низької якості тестування та додаткових витрат на виправлення помилок. Ці питання актуальні для розробників, і їх вирішення сприятиме підвищенню ефективності та якості розробки програмного забезпечення

Огляд існуючих рішень

Було розглянуто :

Nunit
- Легкість у використанні
- Підтримка паралельного виконання тестів
- Інтеграція з CI/CD

xUnit
- Сучасний підхід до тестування
- Різноманітні атрибути для налаштування тестів
- Легка міграція з інших фреймворків

MSTest
- Офіційний фреймворк від Microsoft
- Тісна інтеграція з Visual Studio
- Підтримка функціональних та інтеграційних тестів

Мокування (mocking)
- Імітація поведінки реальних об'єктів
- Використання патернів «Адаптер» та «Фасад»
- Бібліотека Harmony для перехоплення викликів статичних методів

Порівняння атибутів

Порівняння атрибутів у NUnit, xUnit і MSTest важливе для розуміння того, як кожен із цих фреймворків юніт-тестування використовує атрибути для організації та виконання тестів. Атрибути дозволяють фреймворкам ідентифікувати тестові методи та класи, керувати процесом тестування, налаштовувати тестове середовище та інші.

Щоб продемонструвати використання атрибутів і послідовність, у якій виконується тестовий код NUnit, ми розглянемо приклад простої реалізації тесту

Журнал виконання тесту

```
Inside SetUp
Inside TestMethod Test_1
Inside TearDown

Inside SetUp
Inside TestMethod Test_2
Inside TearDown
```


Відмінності між NUnit, xUnit, MSTest

Ізоляція тестів

- xUnit: Висока ізоляція тестів, кожен тест виконується у власному контексті.
- NUnit та MSTest: Тести виконуються у межах одного класу, що може призводити до взаємозалежностей.

Розширюваність

- xUnit: Більша розширюваність через використання [Fact] і [Theory], новий екземпляр класу для кожного тесту.
- NUnit: Багато атрибутів для налаштування тестів ([TestFixture], [TestFixtureSetup]).
- MSTest: Використання [ClassInitialize], [ClassCleanup] для налаштування.

Інтеграція та зручність

- MSTest: Тісна інтеграція з Visual Studio, зручність для користувачів Visual Studio.
- NUnit: Гнучкість та розширені можливості для управління тестами.
- xUnit: Висока чистота коду, підходить для проектів з високими вимогами до ізоляції тестів.

Параметризація тестів

- xUnit: Використання [Theory] для параметризованих тестів.
- NUnit: Підтримка параметризації через атрибути [TestCase].
- MSTest: Підтримка Data-Driven тестів.

Підхід до винятків

xUnit: Використання Assert.Throws.NUnit та MSTest: Використання [ExpectedException].

Вибір фреймворку

NUnit є оптимальним фреймворком для мокування завдяки своїй гнучкості, підтримці паралельного виконання тестів та розширеним можливостям параметризації.

По-перше, NUnit надає різноманітні атрибути, такі як [TestCase] та [Theory], що спрощує створення та управління тестовими випадками. Це забезпечує точність і контроль над тестами.

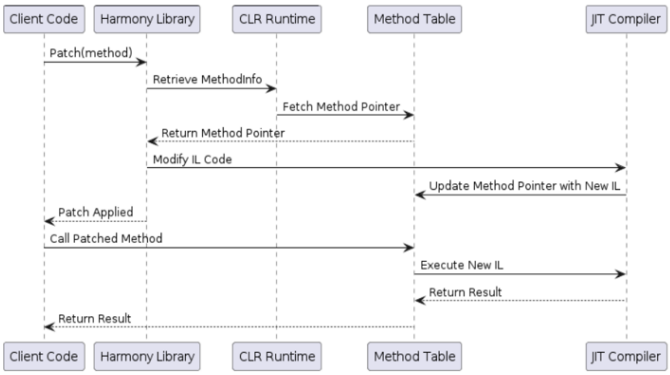
По-друге, підтримка паралельного виконання тестів в NUnit дозволяє значно скоротити час тестування, що є критично важливим для великих проектів.
Крім того, NUnit має широку підтримку спільноти, що полегшує інтеграцію у різні проекти та спрощує навчання.

На відміну від MSTest, NUnit пропонує більшу гнучкість і можливості налаштування, що робить його кращим вибором для складних тестових сценаріїв, де важлива ізоляція та точний контроль.

Вибір бібліотеки для проблеми статики

Проблема мокування статики:
Статичні методи та властивості є частиною класу, а не екземпляру, що ускладнює контроль та ізоляцію.
Статичні методи мають глобальний стан, який впливає на всю програму, що робить мокування викликів складнішим.
Вибір бібліотеки Harmony дає можливість мокування статичних методів. Harmony дозволяє модифікувати та перехоплювати виклики статичних методів за допомогою IL (Intermediate Language).
Гнучкість: Harmony надає можливість вставляти префікси та постфікси до існуючих методів, дозволяючи змінювати вхідні дані або результати.
Рантайм-патчинг: Harmony вносить зміни безпосередньо під час виконання програми, без необхідності зміни вихідного коду.

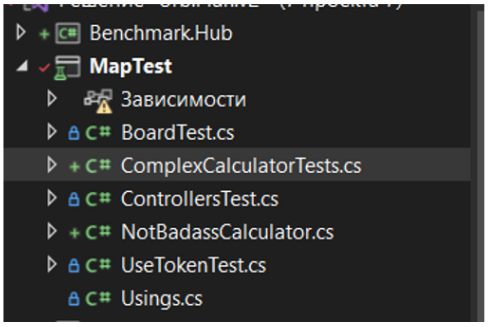
Взаємодія Harmony із рантаймом



Розробка методу мокінгу статики

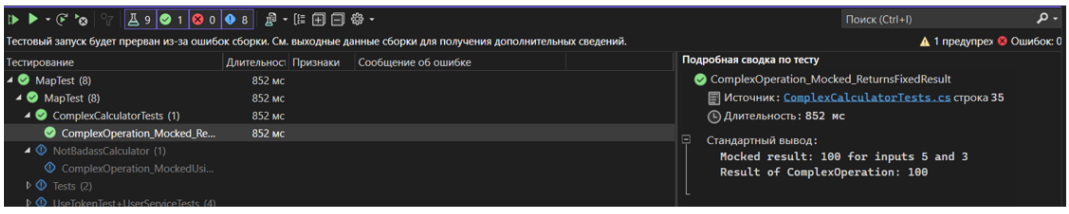
Для вирішення проблеми – нами було написано функціонал який через бібліотеку та її методи дозмолить «мокати» статику.
ComplexCalculatorTests використовує бібліотеку Harmony для патчингу статичного методу ComplexOperation у класі ComplexCalculator.
Ініціалізація Harmony:Створення екземпляра Harmony з унікальним ідентифікатором для уникнення конфліктів.
Цей підхід дозволяє ефективно мокувати статичні методи, забезпечуючи контрольоване та ізольоване тестування

Файлова структура тестів



Аналіз результатів

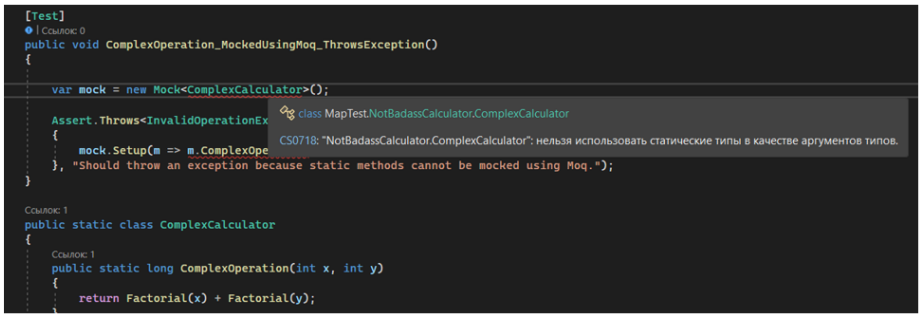
Застосування TestContext.WriteLine у тесті дозволяє вивести результат у консоль тестового середовища, що робить можливим візуалізацію змін у поведінці методу. Це підтверджує успішне мокування.
Результати тесту виводяться у стандартний вивід, де вказано: "Mocked result: 100 for inputs 5 and 3", що підтверджує правильність мокування та роботи патча, а також "Result of ComplexOperation: 100", що відображає кінцевий результат виконання тестованого методу. Ці повідомлення забезпечують зрозумілість і перевірку того, що метод поводиться відповідно до очікувань після застосування модифікації.



Результат мокування статичного методу

Аналіз результатів

Завдяки цим результатам, ми маємо доказ того, що бібліотека Harmony ефективно дозволяє мокувати статичні методи, що не підтримується більшістю традиційних бібліотек мокування. Це надає значних переваг у тестуванні коду, де необхідно ізолювати методи від їхніх залежностей або змінювати їхню поведінку на етапі виконання для перевірки різних сценаріїв.

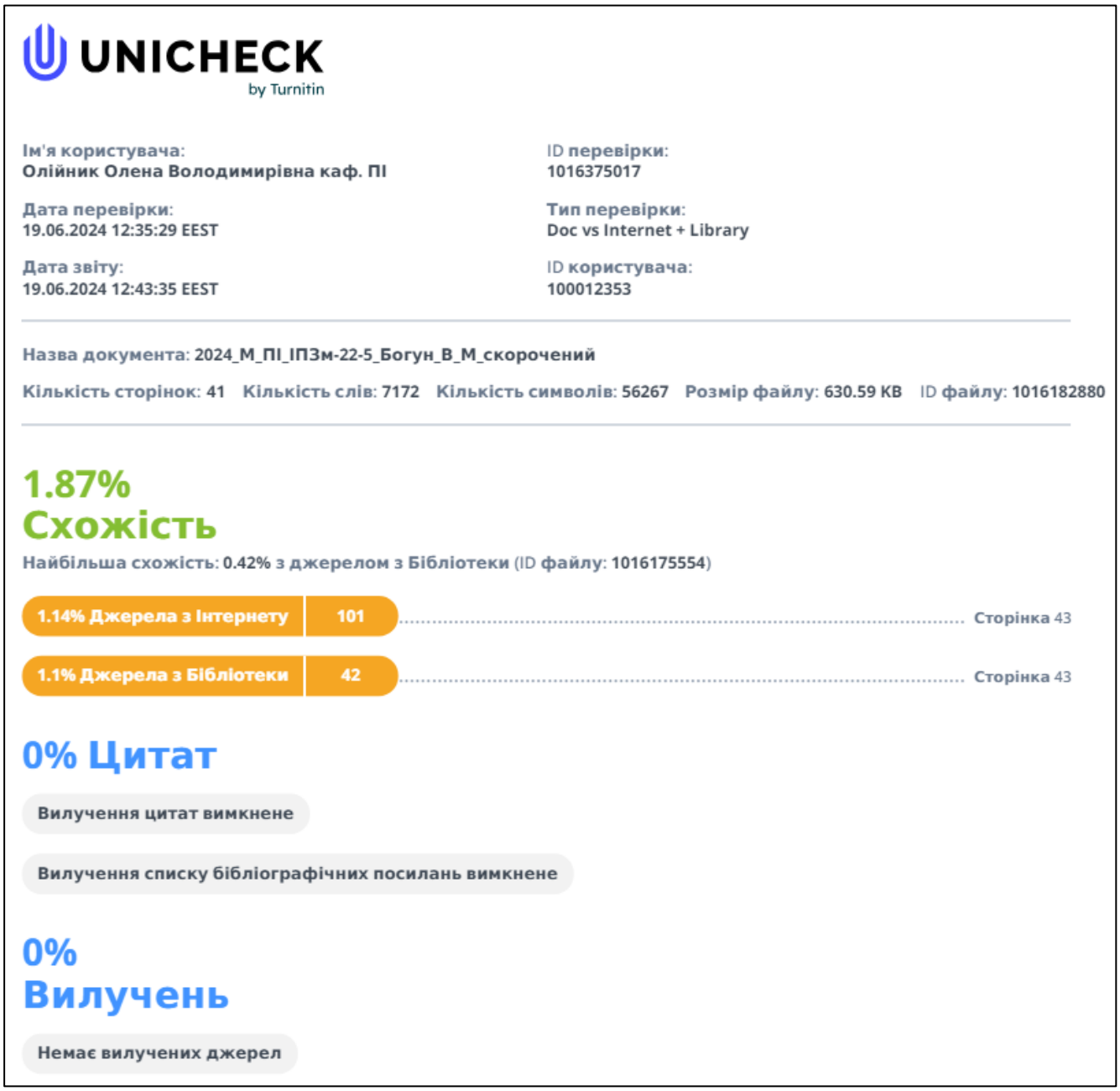


Висновки

- У цій роботі ми провели всебічний аналіз трьох провідних фреймворків для юніт-тестування в екосистемі .NET: NUnit, xUnit та MSTest. Кожен з цих фреймворків має свої унікальні особливості, переваги та можливі сценарії використання, що робить їх підходящими для різних типів проектів та потреб розробників.
- NUnit відзначається своєю гнучкістю та розширеними можливостями, що робить його ідеальним для складних тестових сценаріїв.
- MSTest, як інтегрований компонент Visual Studio, забезпечує легкість використання та зручність для користувачів Visual Studio. Цей фреймворк є гарним вибором для проектів, що вже інтегровані з екосистемою Microsoft та потребують тісної інтеграції з інструментами Visual Studio.
- xUnit пропоЗагалом щоб остаточно визначити найефективніший метод – потрібно додатково провести автоматизовані тести через локалні сітки та більш детальне крос браузерне тиестування.
- В кінці нами було досліджено підходи різних бібліотек до мокування статичних методів та зясували що безкоштовні бібліотеки подібне робити не в змозі, тому за допомогою бібліотеки Harmony – ми через рантайм код змогли замокати статичний метод та продемонструвати працездатність підходу.

ДОДАТОК В

Результат проходження на академічний плагіат



UNICHECK
by Turnitin

Ім'я користувача:
Олійник Олена Володимирівна каф. ПІ

ID перевірки:
1016375017

Дата перевірки:
19.06.2024 12:35:29 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
19.06.2024 12:43:35 EEST

ID користувача:
100012353

Назва документа: 2024_М_ПІ_ІПЗм-22-5_Богун_В_М_скорочений

Кількість сторінок: 41 Кількість слів: 7172 Кількість символів: 56267 Розмір файлу: 630.59 KB ID файлу: 1016182880

1.87%
Схожість

Найбільша схожість: 0.42% з джерелом з Бібліотеки (ID файлу: 1016175554)

1.14% Джерела з Інтернету 101 ..... Сторінка 43

1.1% Джерела з Бібліотеки 42 ..... Сторінка 43

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

ДОДАТОК Г

Апробація результатів роботи

УДК 004.415.3

ДОСЛІДЖЕННЯ МЕТОДІВ ТЕСТУВАННЯ ТА МОКУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ПЛАТФОРМІ .NET

Богун В. М.
Науковий керівник – к.т.н., доц. Голян В. В.
Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна
e-mail: vladyslav.bohun@nure.ua

This work is devoted to the research of methods of creating mechanisms of testing and mocking in the context of software development on the .NET platform. It includes an analysis of the most common unit testing frameworks – NUnit, xUnit, and MSTest – focusing on their key features, attributes, and usage characteristics. The purpose of the work is to identify the optimal testing tools according to the specific conditions of the project and the environment, as well as to analyze the effectiveness of frameworks for the most productive code mocking. The main focus is on the comparative analysis of frameworks in order to find a comprehensive and effective method for improving the quality and reliability of software products.

У процесі дослідження предметної області ми можемо встановити деякі проблеми із підходом до тестування. Вибір фреймворку та підходу до побудови механізмів тестування може бути обумовлений технічними вимогами проекту.

Тип та характер проекту можуть вимагати різних підходів до тестування. Наприклад, для одиничного тестування компонентів може підходити один фреймворк, тоді як для інтеграційних тестів інший. Ця проблема актуальна через різницю у специфікаціях проектів.

Фреймворк з активною спільнотою і підтримкою може бути надійнішим вибором. Спільнота забезпечує швидше виявлення та виправлення помилок, а також забезпечує доступ до ресурсів та допомоги.

Можливість інтегрувати фреймворк з іншими інструментами, такими як системи автоматизованої збірки (CI/CD), також важлива для розробників.

Розробники можуть не розуміти всі різновиди тестування, такі як тестування одиниць, інтеграційне тестування, прийомне тестування та інше.

Саме тому наше дослідження методів створення механізмів тестування та мокування(Mocking) програмного забезпечення, на платформі .Net – є актуальним, адже вирішить вище зазначені труднощі розробників, та закриє багато питань з вибору підходу та стратегії до тестування ПЗ [1].

У ході даної роботи необхідно було дослідити предметну область побудови тестових механізмів за допомогою фреймворків та підходів до тестування на платформі .Net.

Провести порівняльний аналіз та аналіз призначення кожного фреймворку та підходу до тестування.

Провести порівняльні експерименти за певними метриками для визначення більш оптимальної бібліотеки у різних сценаріях тестування програмного забезпечення.

Ця робота спрямована на створення дорожньої карти для розробників щодо знаходження найефективнішого та комплексного підходу до тестування застосунків.

Реалізація наукового дослідження складається з наступних етапів:

– аналіз предметної області;
– аналіз методів тестування;
– розглядання особливостей NUnit, xUnit, MSTest;
– порівняння атрибутів методів побудови механізмів тестування;
– аналіз мокування;
– формування висновків.

Було розглянуто декілька бібліотек тестування, вони представляють собою комплексні методи тестування. Їх порівняння проводиться емпіричними експериментами які містять певні метрики які відповідають різним структурним аспектам порівняння, ось приклади із описом метрик за якими було порівняно бібліотеки:

– час виконання тестів: вимірювання часу, який потрібний для виконання одного або кількох наборів тестів в кожному з фреймворків;

– час запуску тестів: вимірювання часу, який потрібен для запуску фреймворка та завантаження тестових сценаріїв;

– обсяг пам'яті, використовуваний фреймворком: визначення кількості пам'яті, яку використовує фреймворк для запуску тестів та їх виконання;

– стабільність тестового середовища: вимірювання кількості помилок або збоїв, які виникають при запуску тестів у кожному з фреймворків;

– масштабованість: оцінка можливості фреймворка ефективно працювати з великим обсягом тестів та різними конфігураціями проекту;

– підтримка паралельного виконання тестів: визначення швидкості та ефективності паралельного виконання тестів у кожному з фреймворків;

– кількість доступних ресурсів та інструментів: аналіз кількості інструментів та ресурсів, які доступні для автоматизації, звітності та аналізу результатів тестування.

За результатами можна зазначити, що NUnit надає гнучкість через різноманітні атрибути, які можуть бути використані для деталізованого управління тестовими сценаріями. Наприклад, атрибути [TestCase] та [Theory] дозволяють легко впроваджувати параметризовані тести.

Також NUnit підтримує паралельне виконання тестів, що дозволяє оптимізувати час виконання тестів і підвищує гнучкість в управлінні ресурсами. xUnit використовує конструктори класів для ініціалізації та

інтерфейс IDisposable для очищення, що дозволяє більшу гнучкість в управлінні станом тестів. Це сприяє написанню чистих та ізольованих тестових сценаріїв, також він використовує [Fact] для не параметризованих тестів і [Theory] для параметризованих, що дозволяє гнучко керувати даними тестів.

MSTest, будучи інтегрованим рішенням в Visual Studio, часто вважається менш гнучким порівняно з NUnit та xUnit, оскільки він менш орієнтований на спільноту та має меншу підтримку з боку сторонніх розробників інструментів.

У висновку NUnit та xUnit пропонують більшу гнучкість у контексті мокування завдяки своїм особливим підходам до ініціалізації, управління станом тесту, та більшому вибору атрибутів та підтримці з боку спільноти.

Це робить їх більш підходящими для складних сценаріїв тестування, де важливо глибоке управління тестовими випадками та ізоляція станів. MSTest, навпаки, краще підходить для інтегрованих сценаріїв у середовищі Visual Studio, де може бути достатньо його базового функціоналу.

NUnit відзначається своєю гнучкістю та розширеними можливостями, що робить його ідеальним для складних тестових сценаріїв. Підтримка параметризації та можливість визначення порядку виконання тестів дозволяє розробникам ефективно адаптувати NUnit до своїх потреб [2].

xUnit пропонує високу ступінь ізоляції тестів та чистоту коду, що робить його відмінним вибором для проектів, де необхідна ізоляція та незалежність тестів. Ефективність у параметризації та унікальний підхід до управління тестовим середовищем роблять xUnit привабливим для багатьох розробників. MSTest, як інтегрований компонент Visual Studio, забезпечує легкість використання та зручність для користувачів Visual Studio. Цей фреймворк є гарним вибором для проектів, що вже інтегровані з екосистемою Microsoft та потребують тісної інтеграції з інструментами Visual Studio. Також він завдяки більш проробленим сценаріям тестування – більше підходить до процесу мокування.

Загалом, щоб остаточно визначити найефективніший метод – потрібно додатково провести автоматизовані тести через локальні сітки та більш детальне крос браузерне тестування.

Список використаних джерел:

1. Редакційний програмний комплекс клієнт-серверної архітектури з «товстими клієнтами», Сокорчук І, ХНУРЕ, 2018.: https://openarchive.nure.ua/entities/publication/427a0cb4-35ba-45b3-9d3c-8e79de13669a (дата звернення: 01.02.2024).

2. Steve Smith. Unit testing C# in .NET using dotnet test and xUnit. 2024: https://learn.microsoft.com/dotnet/core/testing/unit-testing-with-dotnet-test (дата звернення: 01.02.2024).

ДОДАТОК Д

Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008:2015

Експертний висновок результатів перевірки кваліфікаційної роботи

студент (посада) | програмної інженерії (кафедра) | ІПЗм-22-5 (група)

Богун Владислав Миколайович

(прізвище, ім'я, по батькові)

Зауваження

Пункт ДСТУ 3008-2015	Зміст пункту	Сторінка кваліфікаційної роботи
1	2	3
	7.1 Загальні положення	
	7.3 Нумерація сторінок звіту	
	7.4 Нумерація розділів, підрозділів, пунктів, підпунктів	
	7.5 Рисунки	
	7.6 Таблиці	
	7.7 Переліки	
	7.8 Примітки	
	7.9 Виноски	
	7.10 Формули та рівняння	
	7.11 Посилання	
	7.13 Список авторів	
	7.14 Скорочення та умовні познаки	
	7.15 Додатки	

зауважень немає

Експерт _____ (підпис) Олена ОЛІЙНИК (прізвище, ініціали)

20.06.2024