https://doi.org/10.31891/2307-5732-2025-355-23
УДК 004.4:004.4`2

ЄВСЄЄНКО ОЛЕГ
Національний технічний університет «Харківський політехнічний інститут»
https://orcid.org/0000-0001-5432-1211
e-mail: oleg.yevseienko@gmail.com
ЗУЄВ АНДРІЙ
Національний технічний університет «Харківський політехнічний інститут»
https://orcid.org/0000-0001-8206-4304
e-mail: andrii.zuiev@khpi.edu.ua

ОГЛЯД МОЖЛИВОСТЕЙ КАСТОМІЗАЦІЇ ФРЕЙМВОРКУ XUNIT ДЛЯ МОДУЛЬНОГО ТЕСТУВАННЯ .NET-ЗАСТОСУНКІВ

У статті досліджується ефективність застосування фреймворку xUnit для модульного тестування програмних рішень на платформі .NET. Актуальність теми обумовлена зростаючою потребою в якісному тестовому покритті коду в умовах стрімкого розвитку веб- і десктопних застосунків, а також активного впровадження компонентно-орієнтованих архітектур, зокрема Blazor. Проведено порівняльний аналіз xUnit з іншими популярними фреймворками, такими як NUnit і MSTest, з урахуванням критеріїв зручності використання, продуктивності, розширюваності та сумісності з CI/CD-пайплайнами. Описано типові сценарії використання xUnit у проєктах різного масштабу, включно з тестуванням API, класів бізнес-логіки та компонентів користувацького інтерфейсу. У результаті встановлено, що xUnit відзначається високою гнучкістю, широкою інтеграцією з сучасними інструментами розробки. Водночас деякі функціональні можливості, такі як повторний запуск тестів або інтеграція з зовнішніми джерелами даних, не реалізовано за замовчуванням і потребують сторонніх розширень або додаткової конфігурації. Огляд літератури показав, що автоматизація тестування є важливим елементом забезпечення якості програмного забезпечення, а також необхідним кроком у контексті розвитку складних програмних продуктів із великою кількістю залежностей. У практичній частині запропоновано підходи до розширення функціональності xUnit, зокрема використання кастомних атрибутів, власних механізмів категоризації тестів. Окрему увагу приділено створенню інфраструктури категоризації тестів із застосуванням атрибутів Trait, можливостям їх фільтрації у середовищі Visual Studio та виконання з командного рядка (CLI). Стаття може бути корисною як для початківців, так і для досвідчених розробників, які прагнуть оптимізувати процес модульного тестування в середовищі .NET.

Ключові слова: юніт-тест, автоматизоване тестування, фреймворк, xUnit, програмування, програмні засоби.

YEVSEIENKO OLEH
ZUEV ANDREY
National Technical University «Kharkiv Polytechnic Institute»

OVERVIEW OF XUNIT FRAMEWORK CUSTOMIZATION CAPABILITIES FOR UNIT TESTING .NET APPLICATIONS

The article is devoted to exploring the effectiveness of using the xUnit framework for unit testing software solutions on the .NET platform. The relevance of the topic is driven by the increasing need for high-quality code coverage in the context of rapid development of web and desktop applications, as well as the widespread adoption of component-oriented architectures, particularly Blazor. A comparative analysis of xUnit with other popular frameworks such as NUnit and MSTest is presented, taking into account criteria such as ease of use, performance, extensibility, and compatibility with CI/CD pipelines. Typical use cases of xUnit are described for projects of varying scales, including API testing, business logic validation, and user interface component testing. The findings indicate that xUnit offers high flexibility and broad integration with modern development tools. At the same time, some features — such as test retries or integration with external data sources — are not available by default and require third-party extensions or additional configuration. The literature review confirms that automated testing is a key element in ensuring software quality and a necessary step in the development of complex software systems with a high number of dependencies. In the practical section, approaches to extending xUnit functionality are proposed, including the use of custom attributes and custom test categorization mechanisms. Particular attention is paid to the implementation of test categorization infrastructure using the Trait attribute and the options for filtering tests in Visual Studio and through the command-line interface (CLI). The article may be useful for both novice and experienced developers seeking to optimize the unit testing process in the .NET environment.

Keywords: unit test, automated testing, framework, xUnit, programming, software tools.

Стаття надійшла до редакції / Received 06.05.2025
Прийнята до друку / Accepted 06.06.2025

Постановка проблеми

Модульне тестування, або юніт-тестування – це підхід до тестування програмного забезпечення, який передбачає ізольовану перевірку окремих функціональних одиниць програми (зазвичай методів або класів). Основною метою юніт-тестів є виявлення помилок на ранніх етапах розробки та перевірка правильності роботи кожного модуля незалежно від інших компонентів системи.

Процес написання юніт-тестів дуже схожий на створення звичайного коду. На основі вимог або очікувань до певної функції створюється окремий метод у спеціальному тестовому класі. Цей метод викликає тестований код, передає в нього певні вхідні дані та перевіряє, чи отриманий результат відповідає очікуваному. Для цього здебільшого використовуються твердження (assert), які автоматично фіксують, якщо результат відрізняється від заданого.

Існують усталені практики, яких рекомендується дотримуватись при написанні юніт-тестів. Вони охоплюють різні аспекти: від оформлення коду, найменування методів і змінних до структури та логіки побудови самих тестових функцій. Однією з ключових концепцій є підхід AAA (Arrange, Act, Assert), згідно з яким кожен тест повинен мати чітку структуру: спочатку виконується підготовка необхідних об'єктів та даних, далі – виклик тестованого методу, а потім – перевірка результату на відповідність очікуваному.

Важливо також пам'ятати про принципи, закладені в акронімі FIRST. Він нагадує, що тести мають бути швидкими у виконанні, незалежними від інших тестів, повторюваними з однаковим результатом, самоперевірними (тобто такими, що мають чіткий критерій успішності) та своєчасними – написаними до або під час реалізації функціоналу.

Додаткову увагу варто приділяти правилам найменування, адже зрозумілі імена тестів суттєво полегшують їх підтримку та читання. Для цього використовуються неймінг-конвенції, які вимагають, щоб назви тестових методів та змінних були змістовними і відображали суть перевірки. Зазвичай ім'я тесту містить назву методу, що тестується, початкові умови або вхідні дані, а також очікуваний результат. Такий підхід дозволяє вже з назви тесту зрозуміти, що саме перевіряється в тесті, без необхідності заглядати в його код. Окрім цього, добре написані тести виконують роль неформальної документації до програмного коду: вони показують, як саме має працювати та чи інша функція, які вхідні значення вона приймає і який результат очікується. Таким чином, тести допомагають новим розробникам швидше зрозуміти логіку застосунку, а також виступають як засіб контролю правильності змін у вже реалізованому функціоналі.

Історично першим фреймворком для модульного тестування в середовищі .NET став MSTest. Він був створений як вбудований інструмент Microsoft для підтримки автоматизованого тестування в середовищі Visual Studio. Початково MSTest був закритим і тісно інтегрованим з екосистемою Microsoft, що обмежувало його гнучкість. Проте згодом було представлено MSTest v2. Це дало змогу спільноті впливати на розвиток фреймворку та інтегрувати його з іншими інструментами.

Іншим важливим кроком у розвитку .NET тестування стало створення NUnit – адаптації популярного Java-фреймворку JUnit. NUnit швидко набув популярності завдяки простоті використання, відкритому коду та більшій гнучкості порівняно з MSTest.

Однак з часом команда розробників NUnit дійшла висновку, що додавання нових функцій до існуючого коду ускладнює його підтримку і розвиток. Було ухвалено стратегічне рішення створити новий фреймворк з нуля – так народився xUnit.net. Його розробили ті ж розробники, що і NUnit, але вже з урахуванням минулого досвіду та нових підходів до тестування.

Основними ідеями, що лягли в основу xUnit, були спрощення архітектури, зменшення кількості атрибутів, усунення надлишкового коду під час ініціалізації тестів, поліпшення ізоляції – зокрема, створення нового екземпляра класу для кожного тесту для запобігання взаємному впливу. Також xUnit використовує інтерфейс IDisposable для очищення ресурсів після виконання тесту, замість традиційних атрибутів типу [TearDown] чи [TestCleanup].

З розвитком .NET з'явилась і нова платформа для розробки вебінтерфейсів – Blazor. Вона дозволяє створювати інтерактивні компоненти на C#, які виконуються як на клієнті, так і на сервері. Однак звичайні фреймворки для юніт-тестування не були призначені для перевірки UI-компонентів у Blazor, оскільки не підтримують рендеринг, взаємодію з DOM чи перевірку змісту HTML. У відповідь на цю потребу і з'явився bUnit – тестовий фреймворк спеціально для Blazor-компонентів, створений на основі xUnit.

Навіть маючи потужні підходи до написання тестів і широкий вибір інструментів, існуючі фреймворки не завжди повністю покривають усі потреби розробників. У деяких випадках стандартної функціональності, яку пропонує, наприклад, xUnit, може бути недостатньо. Саме тому виникає потреба в розширенні фреймворку додатковими можливостями – як це реалізовано в bUnit, що базується на xUnit, але спеціально адаптований для роботи з Blazor-компонентами.

Розширення xUnit або інтеграція з додатковими бібліотеками можуть вирішити низку типових проблем. Наприклад, вони дозволяють покращити читабельність вихідних повідомлень у разі невдачі тесту, оскільки стандартний вивід іноді буває занадто стиснутим і неінформативним. Крім того, паралельне виконання тестів, увімкнене за замовчуванням, може призводити до неочікуваних помилок, якщо тести мають спільні ресурси, що потребують ізоляції. Ще одним важливим аспектом є необхідність групування тестів за категоріями чи тегами для кращої організації запуску, а також отримання вбудованої статистики про час виконання кожного тесту для подальшої оптимізації.

Аналіз досліджень та публікацій

Оскільки сфера розробки програмного забезпечення продовжує розвиватися з точки зору складності та масштабів, інтеграція автоматизованого тестування стає важливим елементом для забезпечення якості та надійності програмного забезпечення [1]. Реалізація автоматизованих тестових фреймворків відіграє ключову роль у спрощенні процесу тестування, однак вибір відповідної категорії фреймворку являє собою виклики та вимагає обґрунтованих компромісів [1].

У роботі [2] розглядаються методи автоматизованого тестування програмного забезпечення та надаються рекомендації щодо доцільності використання відповідних інструментів для тестування різних типів програмного забезпечення. Аналізується ефективність застосування автоматизованого тестування для перевірки якості програмних продуктів. У висновках робиться акцент на необхідності розробки

автоматизованих тестів і наданні їм переваги порівняно з ручним тестуванням.

Стаття [3] присвячена реалізації автоматизованого тестування вебзастосунків за допомогою фреймворків Selenium і NUnit, з особливим акцентом на їх інтеграції та продуктивності в реальних умовах. Автори демонструють приклад тестування складного застосунку, описують переваги інструментів, такі як підтримка кросбраузерного тестування, параметризація та можливості CI. Водночас у статті відсутній глибокий аналіз обмежень або недоліків Selenium і NUnit, таких як технічні межі чи специфічні виклики в масштабуванні в дуже великих проектах.

Роботи [4, 5] зосереджені на порівняльному аналізі фреймворків з метою пошуку комплексного та ефективного підходу до підвищення якості й надійності програмних продуктів.

У [6] зазначається, що до недоліків класичного юніт-тестування належить те, що функції, які змінюють стан складного компонента, важко тестувати індивідуально. Іноді неможливо визначити, чи є певний внутрішній стан правильним. Деякі тести вимагають складного контексту, який важко налаштувати. Інші тести сильно залежать від модулів, що все ще перебувають у розробці. На противагу цим твердженням можна зазначити, що призначення юніт-тестів – це тестування окремої функції чи одиниці. Можливо, варто переглянути вихідний код проекту на відповідність архітектурним принципам і практикам написання чистого коду.

Проблему неможливості використання існуючих рішень для забезпечення необхідної гнучкості та адаптивності до специфічних вимог проекту було розв'язано в роботі [7]. Було запропоновано створення окремого фреймворку для тестування. Такий підхід дозволив уніфікувати процес написання тестів і зменшити час на їх створення. Крім того, він забезпечив легку інтеграцію з CI/CD, спростив адаптацію нових членів команди та підвищив загальну якість програмного забезпечення.

У статті [8] було проведено аналіз інструментів для тестування програмного забезпечення з узагальненням їх по рівнях тестування. Основною метою дослідження було вивчення та класифікація методів тестування програмного забезпечення для допомоги фахівцям у виборі правильних інструментів, що забезпечить якість розроблюваного продукту. Автори звертають увагу на труднощі ручного тестування через зростаючу складність програмних продуктів і скорочення циклів розробки, що робить автоматизацію тестування критично важливою. Висновки свідчать, що неправильний вибір інструментів тестування може призвести до неадекватної оцінки якості або необхідності заміни інструментів під час проекту, що впливає на кінцевий результат програмного продукту. Запропоновані класифікації інструментів тестування можуть стати корисними для швидкого орієнтування в доступних варіантах та допомогти фахівцям, особливо тим, хто не є експертами в цій галузі. Загалом, стаття надає важливі рекомендації щодо використання інструментів тестування на різних етапах життєвого циклу розробки програмного забезпечення і зазначає, що автоматизація тестування є важливим елементом для забезпечення якості продуктів в умовах сучасної розробки.

У роботі [9] розглянуто недоліки використання модульного тестування як основної технології перевірки програмного забезпечення. Автори наголошують, що попри поширеність цього підходу, він не завжди є ефективним, зокрема у випадках складної логіки, відсутності точного очікуваного результату, взаємодії з апаратним забезпеченням, а також при потребі оцінки інтеграційних зв'язків і продуктивності. Підкреслено, що без належної культури програмування та підтримки документації модульне тестування не дозволяє повною мірою забезпечити якість програмного продукту, тому його доцільно застосовувати в поєднанні з іншими методами.

У роботі [10] вирішувалась проблема оптимізації вибору системи автоматизації тестування програмного забезпечення відповідно до вимог замовника. Зокрема, це стосувалося вибору найкращої системи автоматизації тестування серед доступних варіантів, враховуючи критерії замовника, такі як платформи, типи додатків, час виконання тестів, інтеграція з іншими системами, бюджет тощо. У результаті було створено підсистему, яка за допомогою алгоритму вибору автоматизованої системи тестування допомагає визначити оптимальний варіант з урахуванням зазначених критеріїв. Це дозволяє ефективно вибирати систему для автоматизації тестування програмних продуктів, що відповідає вимогам замовника, та оптимізує час і витрати на виконання тестування. Таким чином, більшість робіт зосереджуються на необхідності поступової відмови від ручного тестування на користь автоматизованого, що зумовлено зростаючою складністю програмних продуктів та вимогами до якості. У дослідженнях також здійснюється порівняння існуючих фреймворків з метою вибору найбільш придатного рішення залежно від специфіки проекту, наявних ресурсів і вимог замовника.

Мета і задачі дослідження

Метою статті є розширення можливостей використання фреймворку xUnit для автоматизованого тестування компонентів шляхом впровадження додаткових механізмів умовного пропуску тестів, покращення інформативності виводу, а також реалізації підтримки роботи з зовнішніми джерелами даних. Це дозволить адаптувати xUnit до специфічних вимог складних проєктів, оптимізувати процес розробки тестів, знизити часові витрати на налагодження та підвищити ефективність тестування в екосистемі .NET.

Виклад основного матеріалу

Згідно з оглядом літератури, розробникам доступно декілька фреймворків для автоматизованого тестування, серед яких необхідно обрати той, що найкраще задовольняє потреби конкретного проекту. У

більшості випадків перевагу надають xUnit через його розширюваність, меншу кількість атрибутів, а також простий, чистий і зручний для обслуговування код. Завдяки цим перевагам на базі xUnit було створено bUnit – фреймворк, спеціально адаптований для тестування Blazor-компонентів.

Однак у процесі роботи з тестами можуть виникати вузькоспеціалізовані потреби, які базовий функціонал xUnit не покриває. До таких недоліків належать:

1) xUnit має обмежену інформативність виводу при падінні тесту. У багатьох випадках, особливо при використанні базових перевірок, важко швидко зрозуміти причину збою, оскільки вивід не завжди містить достатньо контексту. Це може ускладнити налагодження та сповільнити роботу з тестами, особливо у великих проектах.

2) xUnit не підтримує умовний пропуск тестів на основі логіки, яка виконується під час виконання тесту.

3) Немає прямої підтримки завантаження тестових даних із зовнішніх джерел (наприклад, JSON, XML, CSV) на рівні атрибута, як це реалізовано у NUnit через TestCaseSource. Хоча існують механізми на кшталт MemberData або ClassData, вони вимагають додаткової реалізації логіки імпорту даних у коді, що ускладнює написання тестів при використанні зовнішніх джерел.

4) Неможливість зручно групувати тести за категоріями чи тегами для вибіркового запуску.

5) Відсутність вбудованого механізму вимірювання часу виконання тестів, що ускладнює виявлення повільних або неефективних тестів.

Для вирішення деяких із цих проблем існують сторонні розширення. Наприклад, бібліотека для тестування FluentAssertions дозволяє записувати перевірки в більш читабельному та наочному вигляді, що частково розв'язує проблему неінформативного виводу. Проте залежність від сторонніх розширень не завжди доцільна, особливо з огляду на відсутність офіційної підтримки Microsoft у більшості з них.

Інша помітна відмінність – механізм повторного запуску тестів. У NUnit для цього є вбудований атрибут [Retry(кількість повторень)]. У xUnit повторний запуск тестів потребує додаткової реалізації: створення кастомного атрибута або використання сторонніх бібліотек, наприклад, xunit.retry. Інші підходи містять використання MessageSink для обробки подій під час виконання тестів або навіть розширення XunitTestFramework, що є досить складним для більшості команд.

Як наслідок, вибір фреймворку для юніт-тестування має ґрунтуватися не лише на його популярності, а передусім – на відповідності його можливостей специфіці проекту. Наприклад, якщо критично важливо динамічно підставляти дані з зовнішніх джерел, краще обрати NUnit. Якщо ж пріоритетом є легкість обслуговування та інтеграція з Blazor – xUnit або bUnit.

Розв'язання проблеми умовного пропуску тестів в xUnit

Щоб мати змогу умовно пропускати тести в залежності від зовнішніх факторів (наприклад, значення змінної середовища), можна реалізувати власний механізм умовного виконання тестів:

1. Створити інтерфейс умови:

```
public interface ITestCondition
{
bool ShouldRun { get; }
    string SkipReason { get; }
}
```

Цей інтерфейс дозволяє визначити, чи повинен запускатися тест (ShouldRun), та причину пропуску (SkipReason).

2. Створити клас з реалізацією умови:

```
public class EnvironmentVariableCondition : ITestCondition
{
    public bool ShouldRun =>
        Environment.GetEnvironmentVariable("ENABLE_SPECIAL_TESTS") == "true";

    public string SkipReason => "Тест пропущено, оскільки ENABLE_SPECIAL_TESTS != true";
}
```

Цей клас перевіряє, чи встановлена змінна середовища ENABLE_SPECIAL_TESTS у значення "true".

3. Створити атрибут ConditionalFactAttribute:

```
public class ConditionalFactAttribute : FactAttribute
{
    public ConditionalFactAttribute(Type conditionType)
    {
        if (!typeof(ITestCondition).IsAssignableFrom(conditionType))
            throw new ArgumentException("Тип має реалізовувати ITestCondition");

        var condition = (ITestCondition)Activator.CreateInstance(conditionType)!;
        if (!condition.ShouldRun)
        {
```

```
                Skip = condition.SkipReason;
            }
        }
    }
```

Цей атрибут динамічно створює екземпляр умови, перевіряє її і в разі невиконання – позначає тест як пропущений.

4. Використати атрибут у тесті:

```
public class UnitTests
{
    [ConditionalFact(typeof(EnvironmentVariableCondition))]
    public void TestRunsOnlyWhenEnabled()
    {
        // Цей тест запуститься лише якщо ENABLE_SPECIAL_TESTS == "true"
        Assert.True(true);
    }
}
```

5. Налаштувати значення змінної у файлі test.runsettings.

Щоб передати змінну середовища при запуску тестів, можна створити файл test.runsettings у корені рішення:

```
<?xml version="1.0" encoding="utf-8"?>
<RunSettings>
  <RunConfiguration>
<EnvironmentVariables> <ENABLE_SPECIAL_TESTS>true</ENABLE_SPECIAL_TESTS>
</EnvironmentVariables>
  </RunConfiguration>
</RunSettings>
```

6. Підключити test.runsettings до середовища запуску тестів. Для цього необхідно обрати файл test.runsettings Test → Configure Run Settings Visual Studio.

Завантаження параметрів із зовнішніх джерел (JSON, XML, API)

xUnit не має вбудованої підтримки атрибута для параметризації тестів TestCaseSource, як, наприклад, NUnit. Це створює складність при бажанні підставити параметри з зовнішніх файлів: JSON, XML, CSV. Щоб реалізувати подібну функціональність, можна створити власний атрибут для підвантаження даних із файлу. Для розв'язку цієї проблеми необхідно (приклад для JSON формату):

1. Створити атрибут JsonDataAttribute.

Цей клас читає дані з JSON-файлу та повертає їх у форматі IEnumerable<object[]>, який очікує [Theory]:

```
public class JsonDataAttribute : DataAttribute
{
    private readonly string _path;

    public JsonDataAttribute(string path)
    {
        _path = path;
    }

    public override IEnumerable<object[]> GetData(MethodInfo testMethod)
    {
        var json = File.ReadAllText(_path);

        var root = JsonDocument.Parse(json).RootElement;

        if (root.ValueKind != JsonValueKind.Array)
            throw new ArgumentException("JSON root must be an array.");

        foreach (var item in root.EnumerateArray())
        {
            if (item.ValueKind != JsonValueKind.Array)
                throw new ArgumentException("Each test case must be an array.");

            var parameters = item.EnumerateArray().Select(ToClrObject).ToArray();
            yield return parameters;
        }
```

```
        }

        private object ToClrObject(JsonElement element)
        {
            return element.ValueKind switch
            {
                JsonValueKind.Number when element.TryGetInt32(out var i) => i,
                JsonValueKind.Number => element.GetDouble(),
                JsonValueKind.String => element.GetString()!,
                JsonValueKind.True => true,
                JsonValueKind.False => false,
                _ => throw new ArgumentException($"Unsupported JSON value kind: {element.ValueKind}")
            };
        }
    }
```

2. Використати JSON-дані у тесті:

```
public class MyTests
{
        [Theory]
        [JsonData("testcases.json")]
        public void TestWithJsonData(int a, int b, int expected)
        {
                Assert.Equal(expected, a + b);
        }
}
```

3. Створити файл testcases.json, який може мати такий вигляд:

```
[
  [1, 2, 3],
  [5, 7, 12],
  [-1, 1, 0]
]
```

Додавання категорій до тестів

У xUnit є можливість позначати тести певними категоріями, що допомагає в подальшому фільтрувати їх за заданими характеристиками, такими як тип тесту чи пріоритет. Це дуже корисно, коли потрібно організувати тестування на основі різних категорій, таких як UI, Integration, Unit, Performance тощо. Для цього ми можемо створити власний атрибут та спеціальний клас для обробки категорій тестів:

1. Створити атрибут для категорій CategoryAttribute.

Атрибут CategoryAttribute буде використовуватись для позначення методів тестів, які належать до певної категорії. Цей атрибут дозволяє додавати інформацію про категорію, яку можна пізніше використовувати для фільтрації тестів.

```
[AttributeUsage(AttributeTargets.Method, AllowMultiple = true)]
[TraitDiscoverer("ProjectTests.CategoryDiscoverer", "ProjectTests")]
public class CategoryAttribute : Attribute, ITraitAttribute
{
        public string Name { get; }

        public CategoryAttribute(string name)
        {
                Name = name;
        }
}
```

2. Створити клас для обробки категорій CategoryDiscoverer:

```
public class CategoryDiscoverer : ITraitDiscoverer
{
        public IEnumerable<KeyValuePair<string, string>> GetTraits(IAttributeInfo traitAttribute)
        {
                var category = traitAttribute.GetConstructorArguments().First().ToString();
                yield return new KeyValuePair<string, string>("Category", category);
        }
}
```

3. Використати категорії у тестах.

Тепер можна позначити тести певною категорією, використовуючи атрибут Category. Приклад

використання цього механізму:

```
public class UnitTests
{
    [Category("UI")]
    [Fact]
    public void UITest()
    {
        // Тест для UI компонентів
    }

    [Category("Integration")]
    [Fact]
    public void IntegrationTest()
    {
        // Тест для інтеграційних компонентів
    }
}
```

4. Фільтрація тестів за категоріями.

Після того, як тести будуть позначені категоріями, можна запускати лише ті тести, які належать до конкретної категорії. Це робиться за допомогою фільтрації через команду: dotnet test --filter "Category=UI".

Це дозволяє запускати тільки тести, які належать до категорії UI, що зручно для тестування специфічних частин додатку. Також у Visual Studio можна фільтрувати тести за категоріями безпосередньо у Test Explorer.

Після розробки власних класів та атрибутів для категоризації тестів у xUnit можна зібрати їх у NuGet-пакет. Це дозволить зручно використовувати їх не тільки в локальних проектах, а й поширювати через nuget.org, щоб інші розробники могли використовувати ці функціональні можливості у своїх проектах.

Висновки

Дослідження присвячене аналізу можливостей фреймворку модульного тестування xUnit у контексті розробки програмного забезпечення на платформі .NET. Проведене порівняння із фреймворками NUnit та MSTest засвідчило, що xUnit має низку переваг, серед яких – активна підтримка спільноти, інтеграція з сучасними технологіями (зокрема Blazor через bUnit). Однак дослідження також виявило низку недоліків xUnit. Серед основних – відсутність вбудованих механізмів для підключення зовнішніх джерел даних (JSON/XML/CSV), умовного пропуску тестів на основі кастомної логіки, а також неінформативний вивід при падінні тестів. Крім того, повторний запуск тестів потребує використання сторонніх бібліотек або ручної реалізації. Запропоновані підходи щодо усунення недоліків, зокрема створення кастомних атрибутів для категоризації тестів, використання власних класів для обробки метаінформації, а також налаштування повторних запусків і підключення зовнішніх даних, дозволяють розширити функціональність xUnit без втрати сумісності чи зниження ефективності тестування. Таким чином, вибір фреймворку для тестування має базуватись не лише на його популярності, а й на специфіці проекту. У випадках, де ключову роль відіграє швидке масштабування та модульність, xUnit може бути оптимальним рішенням. Водночас проекти з жорсткими вимогами до параметризованих тестів і повторного запуску можуть скористатися перевагами NUnit.

Література

1. Khankhoje R. An In-Depth Review of Test Automation Frameworks: Types and Trade-offs / R. Khankhoje // International Journal of Advanced Research in Science, Communication and Technology (IJARSCT). – 2023. – Vol. 3, Iss. 1. – P. 55–64. – DOI: 10.48175/IJARSCT-13108.

2. Огляд програмних засобів та методологій для реалізації систем автоматизованого тестування / Л. І. Д'яченко, А. В. Ілін, Л. М. Шумиляк, К. П. Газдюк, О. Ю. Тарновецька // Вчені записки ТНУ імені В. І. Вернадського. Серія: Технічні науки. – 2021. – Т. 32, ч. 1, № 2. – С. 122–129. – DOI: 10.32838/2663-5941/2021.2-1/20.

3. Jayaraman K. D. Automated Testing Frameworks: A Case Study Using Selenium and NUnit / K. D. Jayaraman, A. Shrivastav // International Journal of Research in Humanities & Social Sciences. – 2025. – Vol. 13, Iss. 1. – P. 16–31.

4. Василевський В. О. Порівняльний аналіз фреймворків юніт-тестування в середовищі .Net [Електронний ресурс] / В. О. Василевський, О. В. Романюк // XLIX Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (18–29 травня 2020 р., Вінниця). – Вінниця : ВНТУ, 2020. – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2020/ paper/view/9024.

5. Богун В. М. Дослідження методів тестування та мокування програмного забезпечення на платформі .NET / В. М. Богун // Радіоелектроніка та молодь у XXI столітті : матеріали 28-го Міжнар. молодіж. форуму, Харків, Україна, – Харків, 2024. – Т. 6. – С. 562–564.

6. Sami D. Automated Unit Testing Tool for .Net Framework / D. Sami, H. Abdel-Galil, M. Sami // International Journal of Computer Applications. – 2013. – Vol. 71, No. 7. – P. 11–15. – DOI: 10.5120/12369-8712.

7. Ремінний О. А. Створення фреймворку автоматизованого тестування графічних користувацьких інтерфейсів / О. А. Ремінний // Вісник Хмельницького національного університету. – 2013. – № 5. – С. 212–218.

8. Zasornova I. Study of software testing tools according to the testing levels / I. Zasornova, T. Hovorushchenko, O. Voichur // Computer Systems and Information Technologies. – 2023. – № 1. – P. 38–46. – DOI: 10.31891/csit-2023-1-5.

9. Василевський В. О. Недоліки використання модульного тестування як основної технології тестування / В. О. Василевський, О. В. Романюк // Інформаційно-комп'ютерні технології – 2020 (ІКТ–2020) : тези доп. XI Міжнар. наук.-техн. конф., м. Житомир. – Житомир, 2020. – С. 14–15.

10. Popov A. Choosing the test automation system according to customer requirements / A. Popov, M. Momot, A. Yelizieva // Innovative Technologies and Scientific Solutions for Industries. – 2022. – No. 1(19). – P. 40–46. – DOI: 10.30837/ITSSI.2022.19.040.

References

1. Khankhoje R. An In-Depth Review of Test Automation Frameworks: Types and Trade-offs / R. Khankhoje // International Journal of Advanced Research in Science, Communication and Technology (IJARSCT). – 2023. – Vol. 3, Iss. 1. – P. 55–64. – DOI: 10.48175/IJARSCT-13108.

2. Ohliad prohramnykh zasobiv ta metodolohii dlia realizatsii system avtomatyzovanoho testuvannia / L. I. Diachenko, A. V. Ilin, L. M. Shumyliak, K. P. Hazdiuk, O. Yu. Tarnovetska // Vcheni zapysky TNU imeni V. I. Vernadskoho. Seriia: Tekhnichni nauky. – 2021. – T. 32, ch. 1, № 2. – S. 122–129. – DOI: 10.32838/2663-5941/2021.2-1/20.

3. Jayaraman K. D. Automated Testing Frameworks: A Case Study Using Selenium and NUnit / K. D. Jayaraman, A. Shrivastav // International Journal of Research in Humanities & Social Sciences. – 2025. – Vol. 13, Iss. 1. – P. 16–31.

4. Vasylevskyi V. O. Porivnialnyi analiz freimvorkiv yunit-testuvannia v seredovyshchi .Net [Elektronnyi resurs] / V. O. Vasylevskyi, O. V. Romaniuk // XLIX Naukovo-tekhnichna konferentsiia fakultetu informatsiinykh tekhnolohii ta kompiuternoi inzhenerii (18–29 travnia 2020 r., Vinnytsia). – Vinnytsia : VNTU, 2020. – Rezhym dostupu: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2020/ paper/view/9024.

5. Bohun V. M. Doslidzhennia metodiv testuvannia ta mokuvannia prohramnoho zabezpechennia na platformi .NET / V. M. Bohun // Radioelektronika ta molod u XXI stolitti : materialy 28-ho Mizhnar. molodizh. forumu, Kharkiv, Ukraina, – Kharkiv, 2024. – T. 6. – S. 562–564.

6. Sami D. Automated Unit Testing Tool for .Net Framework / D. Sami, H. Abdel-Galil, M. Sami // International Journal of Computer Applications. – 2013. – Vol. 71, No. 7. – P. 11–15. – DOI: 10.5120/12369-8712.

7. Reminnyi O. A. Stvorennia freimvorku avtomatyzovanoho testuvannia hrafichnykh korystuvatskykh interfeisiv / O. A. Reminnyi // Visnyk Khmelnytskoho natsionalnoho universytetu. – 2013. – № 5. – S. 212–218.

8. Zasornova I. Study of software testing tools according to the testing levels / I. Zasornova, T. Hovorushchenko, O. Voichur // Computer Systems and Information Technologies. – 2023. – № 1. – P. 38–46. – DOI: 10.31891/csit-2023-1-5.

9. Vasylevskyi V. O. Nedoliky vykorystannia modulnoho testuvannia yak osnovnoi tekhnolohii testuvannia / V. O. Vasylevskyi, O. V. Romaniuk // Informatsiino-kompiuterni tekhnolohii – 2020 (IKT–2020) : tezy dop. KhI Mizhnar. nauk.-tekhn. konf., m. Zhytomyr. – Zhytomyr, 2020. – S. 14–15.

10. Popov A. Choosing the test automation system according to customer requirements / A. Popov, M. Momot, A. Yelizieva // Innovative Technologies and Scientific Solutions for Industries. – 2022. – No. 1(19). – P. 40–46. – DOI: 10.30837/ITSSI.2022.19.040.