

ОПТИМИЗИРАНЕ НА АВТОМАТИЗИРАНО ТЕСТВАНЕ НА БИЗНЕС ЛОГИКА В .NET ЧРЕЗ AI BUSINESS LOGIC UNIT TESTING OPTIMIZATION IN .NET USING AI

Maria Marzovanova¹,

Veska Mihova²

e-mail: mmarzovanova@unwe.bg¹,

e-mail: vmihova@unwe.bg²

Абстракт

Автоматизираните тестове за бизнес логика са от критична важност при разработката на бизнес информационни системи. В съвременната среда с постоянно променящи се и нарастващи изисквания, автоматизираното тестване е един от инструментите за гарантиране на качество и стабилност на софтуерните решения. Традиционният начин на създаване на тестове е бавен и податлив на човешки грешки, но с развитието на изкуствения интелект (AI) пред разработчиците и QA специалистите се отварят множество възможности за оптимизация. Този доклад представя едно изследване на ползите от AI като помощно средство за генериране на тестови случаи базирани на анализ на програмния код, генериране на тестове, подбор и приоритизиране на тестове с цел олекотяване на целия процес в .NET приложения.

Abstract

Unit testing the business logic is crucial in the development of business information systems. In a dynamic and increasingly demanding environment, unit testing is one of the tools for assuring quality and stability of software solutions. The traditional way of adding unit tests to a business logic layer is a slow and prone to errors process, but with the evolving artificial intelligence assistance, software developers and QA specialists have more optimization opportunities. This paper presents research results on the benefits of using AI assistance in generating test cases, based on code analysis, generating tests, test selection and prioritization, improving the efficiency and effectiveness of the testing process in .NET application.

Ключови думи: Unit Testing optimization, AI in Unit testing, Business logic.

JEL: C88, L86

Увод

Бизнес информационните системи са пряко зависими от правилността на бизнес логиката заложена в тях. Тестовите написани, за да упражняват слоя с бизнес логиката, са ключови за ранно откриване на регресия в кода и подsigуряване на очакваното поведение на системата. В същото време, писането и поддържането на тестове често се пренебрегва, защото е трудоемко и изисква дълбоки познания в областта на приложение на информационната система, както и в нейната имплементация. Последните версии на инструментите за разработчици с AI асистенция предлагат различни възможности за автоматично генериране на тестови случаи, препоръчват входни данни за тях и приоритизират тестовите случаи с цел по-ефективно и фокусирано тестване. Настоящия

¹ Гл. ас. д-р в катедра „Информационни технологии и комуникации“, УНСС

² Гл. ас. д-р в катедра „Информационни технологии и комуникации“, УНСС

доклад разглежда използването на AI помощ в целия процес на тестването при приложения разработени в .NET.

Съвременната облачна среда, масивите от данни и усложнените зависимости изискват нов подход за подпомагане на QA, тестове и архитектура [5].

Възприемането на автоматизирани и адаптивни подходи при разработването на информационни системи е в съзвучие с тенденциите за системи с висока степен на конфигурируемост и персонализация [6]. Тези принципи могат успешно да бъдат приложени и при внедряването на AI асистенция в процеса на тестване на бизнес логика.

Тестване на кода

По дефиниция от Microsoft [1] тестването е чудесен начин да се подсигури, че едно приложение прави точно това, което неговия автор е целял. В статията се разглеждат няколко вида тестване – единични тестове (Unit tests), интеграционни тестове (Integration tests) и тестове за натоварване (Load tests). Единичните тестове се упражняват върху изолирани единици от кода и целят да подсигурят, че всяка единица от логиката работи както е по дизайн и допринася за цялостната надеждност на системата. Другите видове тестове са също от изключителна важност, но настоящото изследване се фокусира върху единичните тестове и в изложението при споменаване на тестове и тестване се има предвид единично тестване.

За .NET се предлагат различни среди за създаване на тестове, библиотеки с различни възможности и улеснения за верифициране на очакваните резултати от изпълнение на единиците код. Сред тях са NUnit, xUnit и MSTest. NUnit [2] е вече зряла платформа с широк диапазон на опции за проверка. xUnit [3] е по-нова платформа, с повече възможности за разширяване и модуларност. MSTest [4] е вградена Microsoft, която лесно се интегрира с Visual Studio.

Въпреки че тези платформи предоставят солидна основа за тестване в .NET, традиционните подходи към създаването и поддържането на тестове често се сблъскват с предизвикателства. Ръчното писане на тестови случаи е трудоемко, податливо на човешка грешка и може да не обхване всички възможни сценарии или гранични случаи.

За да преодолеем тези ограничения и да постигнем по-ефективен и всеобхватен процес на тестване, ще представим адаптивен подход, базиран на използването на инструменти с изкуствен интелект. Изкуственият интелект е вече в голяма помощ на разработчиците на софтуер, безспорно е неговата полза и при тестването. Предстои да разгледаме подход с ясни и последователни стъпки, следването на които ще доведе до намаляване на времето и възможните грешки при създаването на тестове в .NET среда.

Подход за тестване с AI асистенция

Когато разработчикът избере да използва инструменти с AI възможности при създаването на тестове, добра практика е да се премине през следните стъпки:

- Анализ на кода
- Генериране на тестове
- Подбор и приоритизиране на тестове

- Изпълнение на тестовете и анализ на покритието



Фигура 1: Стъпки в процеса на тестване с AI асистенция

Анализ на кода

Това е процес на семантичен анализ на кода с цел да се определи какво трябва да се тества, как да се изолират отделни единици за тестване, какви входни данни, изходни данни, странични ефекти и зависимости са от значение.

Използват се техники като AST (абстрактно семантично дърво) за извличане на структурата и семантиката на кода. Control flow анализ за идентифициране на възможните пътища при изпълнение на кода. В резултат на това ще бъдат определени единици от бизнес логиката, които е необходимо да бъдат обхванати от тестовете, както и зависимости в кода от свързани методи, външни API.

Един добър инструмент за извършване на анализ на кода е Roslyn, .NET платформа, която предлага API за анализиране на C# код, който поддържа и последните версии на .NET за разлика от други инструменти.

Генериране на тестове

Въз основа на извлечените от анализа на кода метаданни, фазата на генериране на тестове цели автоматичното създаване на цялостен пакет от unit тестове. В зависимост от избрания инструмент за генериране се прилагат различни комбинации от AI техники, включително търсене-базирано тестване и разпознаване на модели, за генериране на разнообразни тестови случаи, насочени към различни аспекти от функционалността на кода. Например, използваме генетични алгоритми за еволюиране на тестове, които максимизират покритието на кода и разкриват потенциални грешки. Генерираните тестове са проектирани да бъдат съвместими с популярни .NET платформи за тестване, като xUnit и NUnit.

В тази фаза може да се използва и GitHub Copilot, за който трябва да се добавят метаданните в кода като коментари. Трябва да се обърне внимание тези подсказки да бъдат ясни и точни, разбираеми за асистента. Този инструмент е много полезен за бързо създаване на скелет на тестове, но изисква внимателен преглед и проверка.

Избор на тестове и приоритизиране

За да се оптимизира процеса на тестване и да се гарантира, че най-критичните тестове се изпълняват първи, се използват AI-базирани техники за селекция и приоритизация на тестовете. Алгоритми за машинно обучение анализират данни за исторически изпълнения на тестове, промени в кода и метрики за сложност на кода, за да предвидят кои тестове е най-вероятно да се провалят или да разкрият нови дефекти. Това ни позволява да приоритизираме тестовете на база на техния риск и потенциално въздействие, позволявайки на програмистите да се фокусират върху най-важните области от кодовата база.

Изпълнение на тестовете

Генерираните и приоритизирани тестове след това се интегрират безпроблемно в .NET средата за разработка. Използват се стандартни .NET платформи за тестване (xUnit, NUnit, MSTest) за изпълнение на тестовете и преглед на резултатите. В добавка може да се разработи собствен инструмент за стартиране на тестове, на който да се укаже приоритизираният ред, гарантирайки че най-критичните тестове се изпълняват първи. Резултатите от тестовете се анализират, за да се идентифицират дефекти и да се предостави обратна връзка на програмистите.

Оценка

За целите на демонстрацията и оценката на описания подход е изготвен един примерен C# клас с четири метода за работа с текст. Самите методи нямат практическо приложение или претенции да изпълняват сложна бизнес логика, създадени са само за целите на демонстрация на подхода. Върху класа се упражнени стъпките в представената последователност. Като краен резултат ще използваме информацията от Visual Studio представяща нивото на покритие на кода с тестове.

Четири метода в клас са както следва:

- Метод за анализ на чувствата в текст (AnalyzeSentiment), който приема като входни данни текст и представя на изхода си в текстов резултат оценка – позитивен, негативен или неутрален.
- Метод за превод на текст (TranslateText), който приема като входни данни текст и целеви език и в резултат връща преведения текст.
- Метод за генериране на изречение с подадена дума (GenerateText), който приема като входни данни дума и представя на изхода си изречение, в което се съдържа тази дума.
- Метод за съкращаване на текст (SummarizeText), който приема като входни данни текст и в резултат връща по-кратък текст.

Първата стъпка е анализ на кода, избран е Roslyn като инструмент за анализ на кода. За използването му е създадено приложение, което достъпва директно целевия клас и извършва семантичен анализ.

От извършения анализ получаваме резултати като представения на фигура 2. В тях се съдържа информация за достъпността на метода, входни параметри, тип на връщания резултат, списък с методи, които се извикват, дали включва try catch конструкции, дали има изрично хвърляне на грешки, както и предложения за ключови тестови случаи.

Този анализ би помогнал и при ръчно създаване на тестове, тъй като съдържа полезна информация, която да насочва разработчикът и да подсигури, че няма да бъдат пропуснати важни елементи от логиката и зависимостите в тествания метод.

```
{
  "Name": "AIGeneratedUnitTestsExample.AICapabilities.AnalyzeSentiment(string)",
  "Accessibility": "Public",
  "ReturnType": "string",
  "Parameters": [
    {
      "Name": "text",
      "Type": "string",
      "IsNullable": false
    }
  ],
  "InvokedMethods": [
    {
      "Name": "string.IsNullOrEmpty(string?)",
      "ContainingType": "string"
    },
    {
      "Name": "string.Split(char, System.StringSplitOptions)",
      "ContainingType": "string"
    },
    {
      "Name": "string.ToLower()",
      "ContainingType": "string"
    },
    {
      "Name": "string.ToLower()",
      "ContainingType": "string"
    }
  ],
  "BranchCount": 5,
  "Throws": false,
  "HasTryCatch": false,
  "SuggestedEdgeCases": [
    "text == null",
    "text is empty (if collection/string)",
    "exercise true/false branches"
  ]
},
```

Фигура 2: Резултати от семантичен анализ с Roslyn

Резултатите записваме в json файл, за използването им в следващата стъпка.

За генерирането на самите тестове използваме асистенцията на GitHub Copilot. Един от възможните начини е да се добави съкратена информация от резултатите от анализа като коментар на всеки метод. След това да се подаде искане към GitHub Copilot да генерира тестове съобразявайки се с тази информация. В примерния клас това не би било затруднение, защото съдържа само няколко метода и не би било толкова времеемко. С оглед на това, че целите на изследването са да утвърди подход приложим и при големи информационни системи, избираме да съхраним информацията като помощен файл добавен в проекта и да насочим AI асистента към този файл при генерирането на тестовете.

Подаваме следното искане в разговора с асистента: *I have a RoslynResults.json file in the project, can you generate unit tests for the project considering those Roslyn analyzer results?*

В резултат получаваме неговите предложения за тестове, общо 14, включващи насоките дадени от анализа.

За показания на фигура 2 метод с резултатите от неговия анализ са генерирани 4 теста включващи ситуацията:

- Подаден е празен текст, в който няма позитивни и негативни думи. Резултатът се очаква да бъде „неутрален“.
- Подаден е текст с повече позитивни думи, отколкото негативни. Резултатът се очаква да бъде „позитивен“.
- Подаден е текст с повече негативни думи, отколкото позитивни. Резултатът се очаква да бъде „негативен“.
- Подаден е текст с равен брой позитивни и негативни думи. Резултатът се очаква да бъде „неутрален“.

```
0 references
public class AICapabilitiesTests
{
    [Fact]
    0 references
    public void AnalyzeSentiment_ReturnsNeutral_ForNullOrEmpty()...

    [Fact]
    0 references
    public void AnalyzeSentiment_ReturnsPositive_WhenMorePositiveWords()...

    [Fact]
    0 references
    public void AnalyzeSentiment_ReturnsNegative_WhenMoreNegativeWords()...

    [Fact]
    0 references
    public void AnalyzeSentiment_ReturnsNeutral_OnTie()...
```

Фигура 3: Генерирани тестове с помощта на GitHub Copilot

Следващата стъпка е селекция и приоритизиране на тестове. Проучването показва, че няма отделен или интегриран инструмент с AI възможности съвместим с Visual Studio, или поне не безплатен такъв. Но GitHub Copilot се оказва полезен и за тази стъпка.

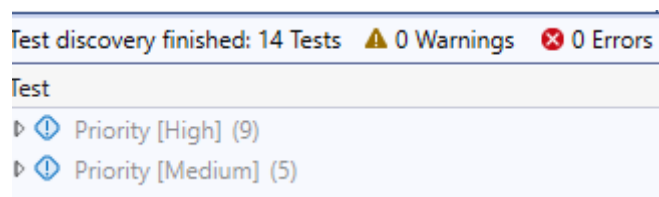
При подадено следното искане : *Can you perform a selection and prioritization of the generated tests and add them in lists?*, са предложени модификации на генерираните по-рано тестове, които отразяват техния приоритет според AI асистента. Добавен е Trait, с име Priority, като са използвани стойностите – висок, среден, нисък.

Според добавеното описание в модификациите, при избора на стойност е използвана следната логика :

- **Висок** приоритет е даден на тестове, които засягат ключово за изпълнението на методите поведение или са базирани на крайни случаи,
- **Среден** приоритет имат тестове за поведение на важни, но не критични, елементи или такива свързани с форматиране например.
- **Нисък** приоритет е даден на тестове свързани със ситуации, които е малко вероятно да се случат в реалността.

И остава да изпълним последната стъпка, да отразим избора и приоритетите при изпълнение на тестовете. Във Visual Studio разполагаме с Test Explorer, в който се извежда пълен списък на всички тестове в системата. Разполагаме с удобно меню за

групиране, с което може да групираме тестовете по Traits. Тъй като имаме само един и той е приоритетът на тестовете, както е показано на фигура 4, лесно се виждат групите и могат да се изпълняват самостоятелно по групи при нужда.



Фигура 4: Групиране на тестове по приоритет в Test Explorer, Visual Studio

При добавяне на повече видове Traits, този подход може и да се усложни, но за такива ситуации може да се изготвят списъци в същия инструмент. Изпълнението им след това е аналогично.

Заклучение

Изследването и демонстративният пример показват обещаващи възможности за бъдещо развитие на AI-базирани решения за тестване в .NET среди. Въпреки че нашето изследване се фокусира върху специфични техники и случаи на употреба, вярваме, че принципите и резултатите могат да бъдат приложени и в други контексти. Бъдещите изследвания ще се фокусират върху по-нататъшното автоматизиране на процеса на тестване, подобряване на точността на приоритизирането на тестовете и разширяване на обхвата на техниките за анализ на кода. Интегрирането на изкуствен интелект в практиките за тестване представлява значителна стъпка към създаването на по-надежден и висококачествен софтуер.

References

1. “Testing in .NET,” Microsoft Learn, 22 Oct. 2025. [Online]. Available: <https://learn.microsoft.com/en-us/dotnet/core/testing/>
2. “What Is NUnit?,” NUnit.org, 2024. [Online]. Available: <https://nunit.org/>
3. “About xUnit.net,” xUnit.net. [Online]. Available: <https://xunit.net/?tabs=cs>
4. “Unit testing C# with MSTest and .NET,” Microsoft Learn, 28 Oct. 2025. [Online]. Available: <https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-csharp-with-mstest>
5. G. Stefanov, “Езера от данни (Data Lakes) като хранилище за Големи данни. Анализ на AWS подхода,” *Научната конференция „Иновативни информационни технологии за дигитализация на икономиката“ (Innovative Information Technologies for Economy Digitalization – IITED)*, УНСС, Sofia, 2023.
6. Milev, P. (2025). Design Solutions for an Information System with User Configuration in a University Environment. *Ikonomiceski i Sotsialni Alternativi*, pp. 110-115.