

ПРО ВПЛИВ РОЗВИТКУ ТЕХНОЛОГІЙ ШТУЧНОГО ІНТЕЛЕКТУ НА ЗМІСТ ПІДГОТОВКИ З ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ БАКАЛАВРІВ ЗА ФАХОМ ПРОГРАМНА ІНЖЕНЕРІЯ

Суслов С.В., к.т.н., доцент¹; Лебедь Ю.М., інженер²

¹ Національний університет кораблебудування імені адмірала Макарова; ² CCSI Ltd

¹ Україна, Миколаїв; ² Ірландія, Дублін

¹ sergiy.suslov@nuos.edu.ua; ² yuriy.lebed@ccsi.ie

***Анотація.** З метою адекватного врахування у навчальному матеріалі дисципліни «Якість та тестування програмного забезпечення» для бакалаврів спеціальності програмна інженерія на тестових задачах досліджено сучасні можливості штучного інтелекту (ШІ) при застосуванні до тестування програм. Результати вказують на доцільність більшої уваги загальній методології тестування*

***Ключові слова:** тестування програмного забезпечення; штучний інтелект; підготовка фахівців.*

Вступ Розвиток галузі програмної інженерії породжує все більший обсяг знань, які потрібні розробникам програмного забезпечення для ефективної роботи. Це, у свою чергу, формулює задачу для закладів освіти, що готують фахівців для галузі, стосовно визначення оптимального змісту навчальних дисциплін при обмеженнях у часі і швидкості засвоєння студентами знань. До бурхливого зростання нових технологій, пов'язаних з хмарними обчисленнями, за останні два роки додався ще більш бурхливий розвиток технологій штучного інтелекту, особливо великих мовних моделей (LLM), таких як GPT і LLaMA .

Крім появи задач розроблення нового класу програмного забезпечення, технології ШІ надали розробникам новий вид програмних інструментів, які, згідно очікуванням, можуть у значній мірі автоматизувати процеси створення ПЗ, починаючи зі специфікацій вимог і до кодування та тестування. У фахівців галузі на цей час склалось неоднозначне ставлення до ефективності застосування цих інструментів – від можливості майже повністю замінити розробників рівня “junior” до заперечення суттєвої користі. Незаперечно, що ця невизначеність викликана різноманітністю технологій, задач і досвіду застосування. Попри це, але з урахуванням реалій, освітянам доводиться визначати конкретний зміст підготовки майбутніх фахівців, що дозволить з меншими втратами почати професійну діяльність.

Міжнародні об'єднання вчених й інженерів ІТ галузі ACM і IEEE при залученні відомих університетів розробили рекомендації з підготовки бакалаврів за спеціальністю програмна інженерія, остання версія яких SE2014 [1] видана у 2015 році і не враховує умови сучасного стану застосування ШІ. У розробленій цими ж організаціями з залученням фахівців з ШІ програмі з комп'ютерних наук CS2023 [2] детально враховано сучасний розвиток ШІ, але не приділено достатньої уваги застосуванню у процесах розроблення ПЗ.

Оприлюднені дослідження зосереджені, головним чином, на технології застосуванні ШІ у тестуванні ПЗ без освітньої спрямованості. Підходи до навчання студентів пошуку дефектів у кодах, згенерованих ШІ розглянуто у статті [3], але стосовно створення відповідних тренувальних і тестових задач.

Мета роботи – з'ясувати доцільний спосіб врахування у наявних умовах спроможності ШІ з розв'язання задач тестування у навчальній дисципліні «Якість та

тестування програмного забезпечення» програми підготовки бакалаврів за спеціальністю програмна інженерія.

Основна частина Дослідження можливостей ІІІ при створенні тестів проведено на ряді завдань відповідних за рівнем навчальному курсу з тестування.

Моделі GPT-4 перше завдання було сформульоване наступним чином:

«Could you generate unit tests for c# code?

```
public interface ISave<TEntity> where TEntity : class
{
    TEntity Save(TEntity entity);
}
```

Отримано наступну відповідь.

«Certainly! Below are some unit tests for the ISave<TEntity> interface using a mock implementation. We'll use a testing framework like NUnit along with a mocking library such as Moq to demonstrate how to create unit tests for the Save method.»

Сам код тестів через обсяг тут не наведено, але його зміст зрозумілий з наступних пояснень у наданій відповіді

«### Explanation

1. ****SetUp Method****: Initializes a mock of `ISave<string>` before each test.
2. ****Save_ValidEntity_ReturnsSameEntity****: Tests that when a valid entity is saved, the same entity is returned.
3. ****Save_NullEntity_ThrowsArgumentNullException****: Tests that saving a null entity throws an `ArgumentNullException`.

Running the Tests

Make sure you have NUnit and Moq installed in your project to run these tests. You can execute them using a test runner compatible with NUnit.

Feel free to modify or expand the tests based on your specific requirements!»

Аналіз відповіді звертає увагу на наступне.

1. ІІІ обрано, запропоновано і використано певну технологію тестування на основі фреймворка NUnit і бібліотеки Moq.

2. Створено 2 тести – один на правильні, а другий на неправильні дані, яких у реальному випадку при конкретній реалізації класу TEntity може бути недостатньо. На це звернено увагу в останньому реченні наведених вище пояснень, де запропоновано модифікувати і розширювати тести відповідно специфічним вимогам.

3. ІІІ надано відповідну імені методу Save – «зберегти» трактовку його призначенню, хоча таке явно не визначено у специфікації. У загальних випадках таке довільне трактування може бути неоднозначним, що спричинить спотворення тестів.

Для наступного завдання було використано задачу, яка запропонована Дж. Майерсом у книзі [4] для перевірки рівня володіння методологією тестування у читача. Задача полягає у створення тестів для програми, що визначає тип трикутника за довжиною сторін. Результати оцінюються по 14 пунктам за наявністю створених тестів певної спрямованості.

Для GPT-4 було запропоновано «Створи тести для методу, який визначає тип трикутника за довжиною сторін на C#»

Нижче наведено доречний фрагмент отриманої відповіді за деякими виключеннями, що є неважливими для висновків дослідження. Чітке форматування коду у оригіналі відповіді по рядках тут змінено для зменшення їх числа.

«Щоб створити тести для методу, який визначає тип трикутника за довжиною сторін на C#, спочатку необхідно реалізувати сам метод. Потім ми можемо використовувати фреймворк для тестування, наприклад NUnit або MSTest, для написання тестів.

1. Реалізація методу csharp

```
public enum TriangleType { Equilateral, Isosceles, Scalene, NotATriangle }
public class Triangle
{
    public static TriangleType GetTriangleType(double a, double b, double c)
```

```

    if (a <= 0 || b <= 0 || c <= 0 || a + b <= c || a + c <= b || b + c <= a)
    { return TriangleType.NotATriangle; }
    if (a == b && b == c) { return TriangleType.Equilateral; }
    if (a == b || b == c || a == c) {return TriangleType.Isosceles; }
    return TriangleType.Scalene;
}
}

```

2. Написання тестів. Використовуємо NUnit у цьому прикладі.»

Далі було подано коди 9 тестів. А саме, 1 тест на правильний - Equilateral трикутник з боками (3,3,3), 3 тести на рівнобедрений – Isosceles (3,3,2), (2,3,3), (3,2,3), 2 тести на різносторонній – Scalene (3,4,5), (5,6,7) і 3 тести на неправильні дані (1,2,3), (0,1,1), (-1,1,1)

Ці тести задовольняють 9 з 14 критеріїв. Відсутні тести: 1) з сумою довжин двох сторін меншою за довжину третьої; 2) три перестановки таких тестів; 3) тест з трьома нульовими довжинами; 4) тести з нецілими значеннями; 5) тести з меншим за 3 числом значень.

Відповідно критеріям загально має бути створено більше 20 тестів. Треба зауважити, що серед 9 згенерованих III тестів два - (3,4,5) і (5,6,7) є еквівалентними. Дж. Майерсом у [4] наведено дані, що середнє число балів за цими критеріями у професійних програмістів складає 7 - 8, тобто рівень отриманого розв'язання завдання можна вважати вищим за середній. Втім, Дж. Майерс наголошує, що наведені критерії оцінки засновані на аналізі реальних реалізацій програм для розв'язання такої задачі, а відповідні їм тести не гарантують виявлення всіх можливих дефектів.

За результатами, отриманими у цьому завданні можна зауважити таке.

1. Хоча згенерований набір тестів є достатньо ефективним, він, по суті, є найбільш ймовірним сполученням лексем, складеним на основі даних, використаних при навчанні LLM. Задача про трикутник розглянута у багатьох інформаційних джерелах вже протягом більше 40 років і тому якість отриманого результату не варто узагальнювати.

2. При певних недоліках у змісті набору тестів, отриманий код є зручною основою для подальшого вдосконалення.

Тестування за специфікаціями було перевірено також на прикладі задачі про вид трикутника з пропозицією створити специфікацію для тестів за принципом «чорної скриньки»

Отримана наступна відповідь.

«Принцип чорної скриньки передбачає тестування функціональності системи без знання внутрішньої структури та коду. Специфікація тестів для методу, що визначає тип трикутника по довжині його сторін, включатиме опис вхідних даних, очікувані результати та різні сценарії тестування.

Специфікація тестів для методу визначення типу трикутника.

Опис методу.

GetTriangleType(double a, double b, double c) приймає три параметри:

a — довжина першої сторони трикутника.

b — довжина другої сторони трикутника.

c — довжина третьої сторони трикутника.

Метод повертає тип трикутника:

Equilateral — рівносторонній трикутник.

Isosceles — рівнобедрений трикутник.

Scalene — різносторонній трикутник.

NotATriangle — не є трикутником.

Вхідні дані й очікувані результати:

Рівносторонній трикутник

Вхідні дані: (3, 3, 3). Очікуваний результат: Equilateral.

Рівнобедрений трикутник

Вхідні дані: (3, 3, 2). Очікуваний результат: Isosceles.

Вхідні данні: (2, 3, 3). Очікуваний результат: Isosceles.

Вхідні данні: (3, 2, 3). Очікуваний результат: Isosceles.

Різносторонній трикутник.

Вхідні данні: (5, 6, 7). Очікуваний результат: Scalene.

Не є трикутником.

Вхідні данні: (1, 2, 3). Очікуваний результат: NotATriangle.

Вхідні данні: (0, 1, 1). Очікуваний результат: NotATriangle.

Вхідні данні: (-1, 1, 1). Очікуваний результат: NotATriangle.

Вхідні данні: (1, 10, 12). Очікуваний результат: NotATriangle.

Сценарії тестування.

Позитивні тести.

Тести на коректне визначення рівнобічного трикутника.

Тести на коректне визначення рівнобедреного трикутника.

Тести на коректне визначення різностороннього трикутника.

Негативні тести.

Тести на випадки, коли введені значення не можуть утворити трикутник (наприклад, одне або більше значень менше або дорівнює нулю, або сума двох сторін менша або дорівнює третій стороні).

Граничні умови

Вхідні дані межі допустимих значень (наприклад, мінімальні позитивні довжини, великі числа).»

В сформованій специфікації тестів відсутні у явному вигляді посилання на застосовані критерії «чорної скриньки», що не надає структури для можливого розширення набору тестів, проте на додаток до попереднього набору включено граничні умови.

Висновки Створені LLM тести можуть виявитись не повними, проте корисні в якості основи для доповнення і корегування. Генерація кодів тестів зі зверненням до потрібних фреймворків і бібліотек звільняють людину від суттєвої частини рутинної роботи. Тому при використанні інструментів ШІ увага фахівця має бути спрямована переважно на якість тестів, що підвищує роль підготовки з методології тестування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] *Software Engineering 2014 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*. (23, 15 лютого). Association for Computing Machinery. <https://www.acm.org/binaries/content/assets/education/se2014.pdf>

[2] *Computer Science Curricula 2023 Version Beta*. (2023, березень). CS2023 – ACM/IEEE-CS/AAAI Computer Science Curricula. <https://csed.acm.org/wp-content/uploads/2023/03/Version-Beta-v2.pdf>

[3] Nimisha Agarwal, Viraj Kumar, Arun Raman, & Amey Karkare. (2023). *A Bug's New Life: Creating Refute Questions from Filtered CS1 Student Code Snapshots | Proceedings of the ACM Conference on Global Computing Education Vol 1*. ACM Conferences. <https://doi.org/10.1145/3576882.3617916>

[4] Badgett, T., Myers, G. J., & Sandler, C. (2012). *Art of Software Testing*. Wiley & Sons, Incorporated, John.

Suslov S.V., Lebed Y.M.

On the artificial intelligence technologies development influence on the software testing course content for education of software engineering bachelors

Abstract. For the sake of proper consideration in the software engineering bachelor discipline "Software Quality and Testing" of the artificial intelligence (AI) capabilities applying to software testing, these were investigated on test tasks. The results show to be necessary applying adequate attention to the common testing methodology.

Keywords: software testing; artificial intelligence; specialists training