Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій

(повна назва)

Кафедра Інформаційно-вимірювальних технологій

(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти другий (магістерський)

Тема: Впровадження штучного інтелекту у процеси забезпечення якості програмного забезпечення

Виконав:
Студент 2го курсу, групи ЗЯм-24-1
Кобрін І. С.
спеціальності 175
Інформаційно-вимірювальні технології
Тип програми освітньо-професійна
освітня програма «Забезпечення якості»

Керівник: доцент кафедри ІВТ, к.т.н.

Дегтярьов О.В.

Допускається до захисту

Зав. кафедри _____ д.т.н., проф. Захаров І.П.

(підпис) (прізвище, ініціали)

Харків 2025р.

Харківський національний університет радіоелектроніки
Факультет_____Інфокомунікацій_____
Кафедра_____Інформаційно вимірювальних технологій_____

Рівень вищої освіти_____другий (магістерський)_____

Спеціальність 175 – __Інформаційно-вимірювальні технології_____

(код і повна назва)

Освітня програма _____«Забезпечення якості»_____

(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____ (підпис)

«_____»_____ 20 _25_ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

Студентові _____Кобріну Івану Сергійовичу_____

1. Тема роботи: ___Впровадження штучного інтелекту у процеси забезпечення якості програмного забезпечення_____

затверджена наказом по університету від «07» __11__ 2025 р. №1011Ст

2. Термін подання студентом роботи (проекту) _____

3. Вихідні данні до роботи

4. Перелік питань, що потрібно опрацювати в роботі:

1. AI в процесах забезпечення якості програмного забезпечення
   1.1 Актуальність впровадження Штучного інтелекту
   1.2 Сучасний стан AI в QA
   1.3 Проблеми застосування
2. Штучний інтелект у тестуванні та забезпеченні якості пз
   2.1 Основні поняття та технології ШІ
   2.2 Моделі і алгоритми ШІ, що застосовуються в QA
   2.3 Огляд сучасних інструментів ШІ для тестування ПЗ
   2.4 Переваги і виклики впровадження ШІ в QA
3. Практичне впровадження ші у процеси забезпечення якості програмного забезпечення
   3.1 Вибір методів та інструментів тестування на основі ШІ
   3.2 Реалізація і налаштування AI-систем у QA
   3.3 Аналіз та інтерпретація результатів впровадження

3.4 Виявлення проблем та шляхів їх подолання

5. Перелік графічного матеріалу із зазначенням креслеників,схем,плакатів,комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)_____
_____

6.Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада,прізвище,ім'я,по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Керівник	доцент кафедри ІВТ, к.т.н. Дегтярьов О.В.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів роботи	Примітка
1	Огляд літератури	9.11.2025	
2	Формування завдань на дослідження	12.11.2025	
3	Розробка основної частини тексту	13.11.2025	
4	Формулювання висновків	13.12.2025	
5	Оформлення тексту	14.12.2025	
6	Побудова презентації	15.12.2025	
7	Подання роботи на перевірку	15.12.2025	

Дата видачі завдання ____01.05.2025_____

Студент_____

(підпис)

Керівник роботи _____ доцент кафедри ІВТ, к.т.н. Дегтярьов О.В.

(підпис) (посада,прізвище,ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи містить 73 сторінки, 11 рисунків, 11 таблиць, 55 джерел.

Об’єктом дослідження виступають процеси забезпечення якості програмного забезпечення в контексті сучасних методологій розробки.

Предметом дослідження є методи, технології та інструменти штучного інтелекту, що застосовуються для автоматизації та оптимізації тестування програмного забезпечення.

Мета роботи: дослідження теоретичних основ та практичних аспектів впровадження технологій штучного інтелекту у процеси забезпечення якості програмного забезпечення, аналізі переваг та викликів такої інтеграції, а також визначенні перспективних напрямків розвитку AI-керованого тестування.

ШТУЧНИЙ ІНТЕЛЕКТ, AI, ЗАБЕЗПЕЧЕННЯ ЯКОСТІ, QA, ТЕСТУВАННЯ, ВЕЛИКА МОВНА МОДЕЛЬ, LLM

ABSTRACT

The explanatory note to the qualification work contains 73 pages, 11 pictures, 1 tables, 55 sources.

The object of the study is software quality assurance processes in the context of modern development methodologies.

The subject of the study is artificial intelligence methods, technologies, and tools used to automate and optimize software testing.

Purpose of the work: to study the theoretical foundations and practical aspects of implementing artificial intelligence technologies in software quality assurance processes, analyze the advantages and challenges of such integration, and identify promising areas for the development of AI-driven testing.

ARTIFICIAL INTELLIGENCE, AI, QUALITY ASSURANCE, QA, TESTING, LARGE LANGUAGE MODEL, LLM

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

1. AI - Artificial Intelligence.
2. API - Application Programming Interface.
3. CI/CD - Continuous Integration (CI) and Continuous Delivery/Deployment.
4. CNN - Convolutional Neural Network.
5. CV - Computer Vision.
6. DevOps - Development Operations.
7. DL - Deep Learning.
8. IoT - Internet of Things.
9. IT - Information Technology.
10. KNN - K-Nearest Neighbors.
11. LLM - Large Language Model.
12. LSTM - Long Short-Term Memory.
13. NLP - Natural Language Processing.
14. NN - Neural Networks.
15. PCA - Principal Component Analysis.
16. QA - Quality Assurance.
17. RNN - Recurrent Neural Network.
18. ROI - Return on Investment.
19. SDLC - Software Development Life Cycle.
20. SPA - Share Purchase Agreement.
21. SVM - Support Vector Machine.
22. SVM - Support Vector Machine.
23. XGBoost - Extreme Gradient Boosting.
24. KPI - Key Performance Indicators.
25. НМ - Нейронні мережі.
26. ПЗ - Програмне Забезпечення.
27. ШІ - Штучний Інтелект.

ВСТУП

Швидкий розвиток інформаційних технологій у 21 столітті призвів до збільшення складності програмного забезпечення та підвищення вимог до якості. У сучасних умовах цифрової трансформації бізнесу та суспільства програмне забезпечення стало критично важливим компонентом практично всіх сфер людської діяльності, від охорони здоров'я та фінансових послуг до освіти та державного управління. Це вимагає забезпечення високої надійності, безпеки та функціональності програмних продуктів на всіх етапах їх життєвого циклу.

Традиційні підходи до забезпечення якості програмного забезпечення, що базуються переважно на ручному тестуванні та статичних сценаріях верифікації, все частіше виявляються недостатніми для задоволення сучасних вимог. Згідно зі Світовим звітом про якість 2023-24, 75% компаній активно інвестують у штучний інтелект з метою забезпечення якості, що свідчить про визнання галуззю трансформаційного потенціалу цієї технології.

Актуальність цієї теми зумовлена стрімким зростанням складності програмних систем, підвищенням вимог до швидкості випуску продуктів та необхідністю забезпечення високої якості в умовах конкурентного середовища. Впровадження штучного інтелекту в процеси забезпечення якості має вирішальне значення для організацій, які прагнуть зберегти конкурентоспроможність та відповідати очікуванням ринку щодо надійності та продуктивності програмного забезпечення.

1 AI В ПРОЦЕСАХ ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Актуальність впровадження Штучного інтелекту

Впровадження штучного інтелекту в процеси забезпечення якості приносить значні переваги. Дослідження показують, що автоматизація тестування на основі ШІ може скоротити час виконання тестів на 40-50%, підвищити точність виявлення дефектів на 15-20% та зменшити витрати на обслуговування тестових сценаріїв на 30-50%. Системи на основі ШІ можуть забезпечити ширше тестове покриття, виявляти помилки на ранніх етапах розробки та оптимізувати розподіл ресурсів для тестування. Завдяки можливостям прогнозної аналітики ШІ може ідентифікувати області коду з високим ризиком і рекомендувати пріоритетні області тестування, що призводить до більш ефективного використання часу і зусиль командами забезпечення якості.

Сучасний ринок програмного забезпечення швидко трансформується під впливом цифровізації, зростаючої складності IT-продуктів та інтеграції хмарних, мобільних і IoT-рішень. Якість програмного забезпечення стає ключовим фактором не тільки для успіху бізнесу, але й для безпеки користувачів, стабільності інфраструктури та захисту даних. Традиційні методи забезпечення якості (QA) та ручне тестування вже не здатні забезпечити необхідний рівень продуктивності та точності у великих і динамічних проєктах, особливо у фінансовому, медичному секторах та секторі критичних систем[1].

На цьому тлі штучний інтелект (ШІ) розглядається як рушійна сила розвитку галузі тестування програмного забезпечення. Завдяки автоматизації рутинних процесів, інтелектуальному аналізу даних та прогнозуванню дефектів інструменти AI можуть значно підвищити швидкість, глибину та якість тестування, відкриваючи новий рівень еволюції процесів забезпечення якості.

ШІ слід розуміти як систему, створену за допомогою інформаційних технологій, яка прагне моделювати низку аспектів процесів мислення та функціонування людини. Термін «ШІ» використовується для опису систем, здатних імітувати ключові функції сприйняття та розуміння, властиві людському розуму.

Машинне навчання – це галузь ШІ, яка застосовує алгоритми та дані для копіювання способу навчання людей, поступово вдосконалюючи точність.

Нейронні мережі є базовою технологією для ШІ. Нейронні мережі – це спроба змоделювати роботу мозку живої істоти, котрий складається з мільйонів нейронів, кожен з яких зв'язаний з декількома іншими. Кожен окремий нейрон дуже простий, але разом вони здатні опановувати виконання складних завдань. Штучні нейронні мережі наслідують ці процеси, спираючись на навчальні дані, з плином часу підвищуючи свою точність.

Глибинне навчання – галузь штучного інтелекту, яка дозволяє машинам виконувати ті завдання, які зазвичай роблять люди. Така методика машинного навчання дає змогу комп’ютерам набувати знання на людських прикладах, що у підсумку сприяє автоматизації різноманітних процесів.

Тестування займає важливе місце у процесі творення програмного забезпечення. Зазвичай виділяють такі етапи тестування програмного забезпечення:

1. Планування та управління, аналіз та проєктування;
2. Запровадження та виконання;
3. Оцінка критеріїв виходу та створення звітів;
4. Дії після завершення тестування.[4]

1.2 Сучасний стан AI в QA

Світовий ринок штучного інтелекту в тестуванні програмного забезпечення переживає стрімке зростання. За прогнозами аналітиків, обсяг цього ринку зросте з 1,01 млрд доларів у 2025 році до 3,82 млрд доларів у 2032 році, що відповідає середньому річному темпу зростання 20,9%. Це відображає

серйозні зміни в галузі забезпечення якості та визнання ШІ стратегічним активом для організацій, які прагнуть підвищити ефективність розробки та якість програмного забезпечення.

До 2025 року більшість провідних світових IT-компаній інтегрують інструменти ШІ у свої процеси забезпечення якості. Найбільший розвиток спостерігається в таких сферах:

- автоматична генерація тестових сценаріїв та даних,
- обробка великих масивів інформації для пошуку аномалій,
- прогнозування дефектів у проєктах,
- автоматична класифікація та визначення критичності багів,
- використання великих мовних моделей (LLM) для створення адаптивних тестів та аналізу бізнес-вимог.[42]

Провідні рішення серед інструментів – інтегровані AI-агенти, здатні виконувати генерацію тестових кейсів, адаптуватися до оновлень продукту і самостійно знаходити “слабкі місця” у коді. Водночас штучний інтелект активно впроваджується у сферу автоматизації DevOps, пришвидшуючи цикли релізу і мінімізуючи людський фактор у тестуванні.

- Інтеграція рішень на основі штучного інтелекту для тестування програмного забезпечення пропонує численні переваги, які спрощують процеси контролю якості та покращують якість програмного забезпечення:
- Підвищення ефективності тестування: інструменти на основі ШІ автоматизують повторювані завдання, такі як виконання тестів і формування звітів, що значно прискорює процес тестування та зменшує ручну працю. Це сприяє швидкому випуску продукту без шкоди для якості.
- Покращення охоплення тестуванням: ШІ допомагає командам з забезпечення якості досягти ширшого охоплення тестуванням, автоматично виявляючи крайні випадки та складні сценарії, які можуть бути пропущені традиційними методами тестування. Це забезпечує більш ретельну оцінку програмного забезпечення.

- Прогнозна аналітика для управління ризиками: аналізуючи історичні дані, моделі штучного інтелекту прогнозують потенційні дефекти та вразливості, що дозволяє командам вирішувати проблеми на ранній стадії. Це зменшує витрати та зусилля на виправлення дефектів після їх потрапляння у виробництво.
- Швидший зворотний зв'язок та постійне вдосконалення: інструменти ШІ надають зворотний зв'язок у режимі реального часу, що дозволяє розробникам виявляти дефекти на ранній стадії циклу розробки. Це прискорює процес розробки програмного забезпечення та забезпечує постійне вдосконалення завдяки ітеративному тестуванню.
- Зниження витрат: автоматизація тестування на основі штучного інтелекту знижує витрати на ручне тестування, покращує розподіл ресурсів та мінімізує необхідність постійного обслуговування тестів. Інструменти штучного інтелекту можуть автоматично адаптуватися до змін у програмі, усуваючи необхідність постійного оновлення скриптів.[4]

Аналіз даних, які демонструють значну різницю між традиційними методами забезпечення якості (QA) та методами на основі штучного інтелекту за останні 10 років (Рис. 1.1). З 2015 по 2025 рік виявив драматичне покращення всіх ключових метрик якості програмного забезпечення при використанні ШІ порівняно з традиційними методами.

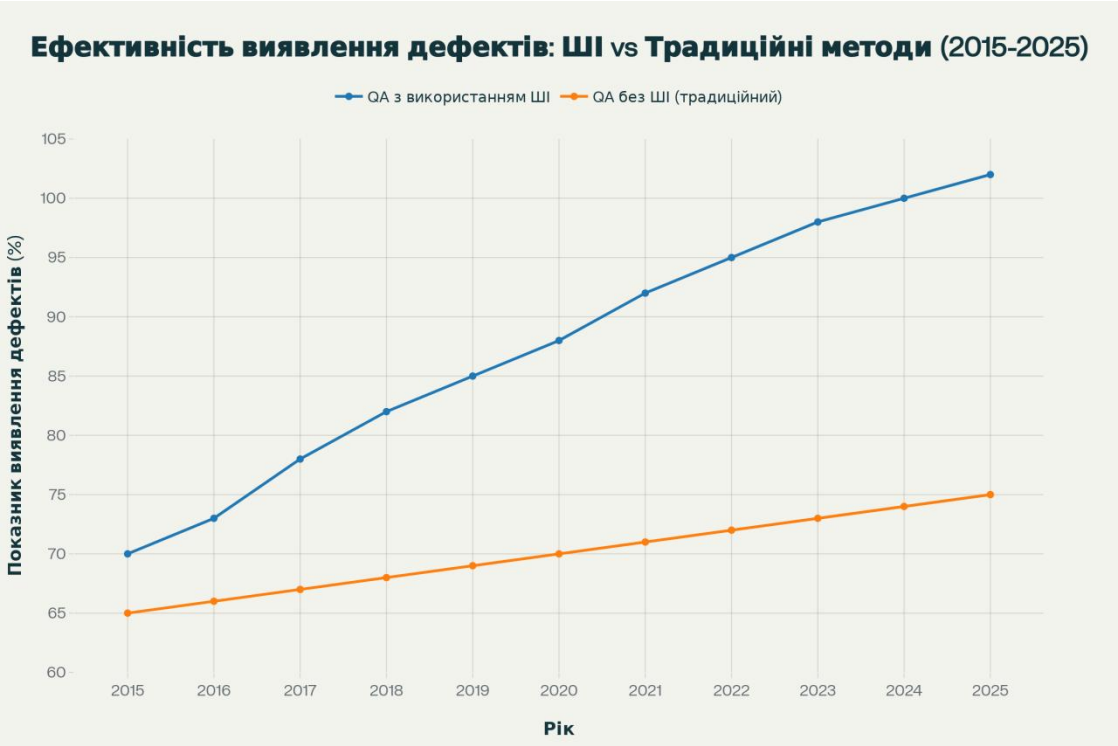


Рис. 1.1 - Ефективність виявлення дефектів: ШІ та Традиційні методи

Експерти прогнозують подалий розвиток:

- Повністю автономні системи QA з мінімальним втручанням людини
- ШІ-агенти, що пишуть та підтримують тести самостійно
- Предиктивне запобігання дефектів ще до написання коду
- Інтеграція з квантовими обчисленнями для тестування складних систем
- Повна інтеграція ШІ у SDLC від вимог до моніторингу продакшену

Зростаюча роль штучного інтелекту в тестуванні програмного забезпечення робить революцію в процесах забезпечення якості. Завдяки автоматизації повторюваних завдань, підвищенню точності тестування та наданню інформації в режимі реального часу, ШІ підвищує ефективність і результативність тестування програмного забезпечення. Його широке застосування закладає основу для більш інтелектуального, швидкого та надійного підходу до контролю якості, що забезпечує найвищу якість програмного забезпечення.

1.3 Проблеми застосування

Водночас, інтеграція штучного інтелекту у процеси QA супроводжується низкою викликів. Серед основних проблем — висока вартість початкового впровадження, складність налаштування та підтримки AI-систем, залежність від якості та обсягу навчальних даних, а також потреба у спеціалізованих знаннях у галузі машинного навчання. Крім того, відсутність стандартизації інструментів та методологій AI-тестування ускладнює вибір оптимальних рішень для конкретних організаційних потреб.

Попри очевидні переваги, впровадження ШІ у процеси забезпечення якості ПЗ пов’язане з низкою викликів:

- Кваліфікація персоналу. Дефіцит спеціалістів, які вміють коректно налаштовувати і використовувати AI-інструменти, знижує ефективність впровадження.
- Якість навчальних даних. Точність роботи AI-моделей сильно залежить від якості і повноти даних; недостатньо якісні дані ведуть до хибних висновків.
- Складність інтеграції. Впровадження AI-рішень у існуючі бізнес-процеси потребує глибоких технічних змін та додаткових фінансових витрат.
- Вартість рішень. Реалізація AI-інструментів на практиці часто потребує значних початкових інвестицій – не лише у самі технології, а й у навчання персоналу та оновлення інфраструктури.
- Довіра до результатів. Рішення, які приймає AI, потребують додаткової перевірки, і позбавити команду "чорної скриньки" (black box) – складне завдання для проектних менеджерів.

Вирішення цих проблем потребує подальших досліджень, розробки нових алгоритмів перевірки правильності роботи AI у тестуванні ПЗ і інтегрованих навчальних програм для фахівців у сфері QA.

2 ШТУЧНИЙ ІНТЕЛЕКТ У ТЕСТУВАННІ ТА ЗАБЕЗПЕЧЕННІ ЯКОСТІ ПЗ

2.1 Основні поняття та технології ШІ

Штучний інтелект (ШІ, англ. Artificial Intelligence, AI) – це галузь комп'ютерних наук та інженерії, що зосереджена на розробці інтелектуальних машин, здатних виконувати завдання, які зазвичай потребують людського інтелекту, такі як візуальне сприйняття, розпізнавання мови, прийняття рішень та переклад мов. ШІ являє собою технологію, яка дозволяє комп'ютерам та машинам імітувати процеси людського навчання, розуміння, вирішення проблем і прийняття рішень.

Системи ШІ розроблені для того, щоб навчатися на основі досвіду, адаптуватися до нових ситуацій і покращувати свою продуктивність з часом без явного програмування. Основна мета ШІ полягає у створенні машин, які можуть симулювати людський інтелект, включаючи здатність до міркування, вирішення проблем та креативності.

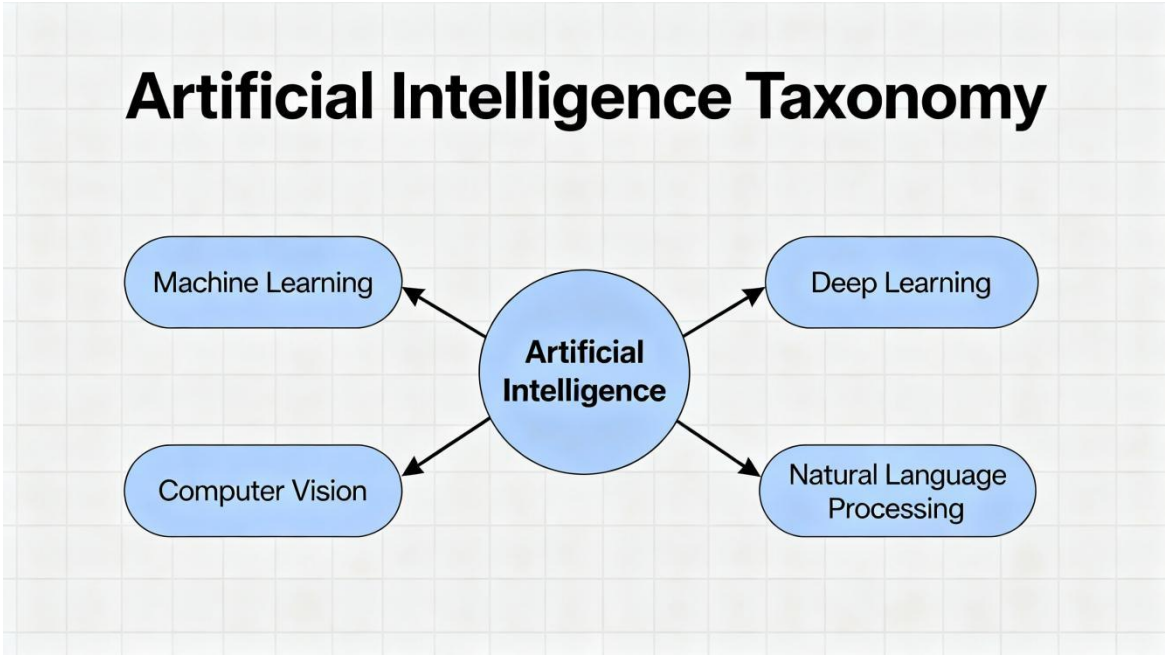


Рис. 2.1 - Ієрархія технологій штучного інтелекту та її основні галузі

2.1.1 Машинне навчання як основа сучасного ШІ

Machine Learning (ML, укр. МН - машинне навчання) є однією з найважливіших технологій ШІ і центральною підобластю штучного інтелекту. МН – це підхід, який дозволяє комп'ютерним системам навчатися на основі досвіду та даних без явного програмування кожного кроку. На відміну від традиційних програм, де розробники вказують усі можливі рішення, системи з ML навчаються на основі наданих даних та поступово покращують свою точність.

Машинне навчання базується на трьох основних компонентах:

1. Алгоритми – спеціальні програми, які «підказують» комп'ютеру, яким джерелом даних необхідно скористатися. Для кожної задачі підбираються окремі алгоритми, складені з розрахунком на прискорення обробки даних та отримання точного результату.
2. Набори даних (датасети) – інформація у вигляді текстових, графічних, відеофайлів, яку машина використовує для накопичення досвіду при навчанні. Для вирішення кожного конкретного типу завдань у систему повинні завантажуватися унікальні дані.
3. Обчислювальна потужність – апаратні ресурси, необхідні для обробки великих обсягів даних та виконання складних математичних операцій.

Існує чотири основних типи машинного навчання, кожен з яких має свої особливості та сфери застосування:

1. Навчання з учителем (Supervised Learning)

При навчанні з учителем алгоритм навчається на розмічених даних, де кожен приклад має відомі вхідні та вихідні значення. Система намагається побудувати модель, яка може передбачати цільову змінну на нових даних. Цей тип навчання використовується для двох основних категорій завдань:

Класифікація – передбачення дискретних категорій або міток (наприклад, виявлення спаму, медична діагностика, розпізнавання зображень).

Регресія – прогнозування неперервних значень (наприклад, прогнозування цін на нерухомість, фінансове прогнозування).

Типові алгоритми включають лінійну регресію, логістичну регресію, метод опорних векторів (SVM), дерева рішень та нейронні мережі.

2. Навчання без учителя (Unsupervised Learning)

У цьому випадку модель навчається на нерозмічених даних без відомих правильних відповідей. Метою є виявлення прихованих закономірностей або структур у даних. Основні підходи включають:

Кластеризація – групування схожих точок даних (наприклад, сегментація клієнтів, аналіз поведінки).

Зниження розмірності – спрощення складних даних зі збереженням важливих патернів (наприклад, метод головних компонент - PCA).

Виявлення аномалій – ідентифікація нетипових патернів у даних (наприклад, виявлення шахрайства).

3. Навчання з підкріпленням (Reinforcement Learning)

Цей тип навчання включає агента, який взаємодіє з середовищем, отримуючи винагороди або покарання за свої дії. Агент навчається методом спроб і помилок, прагнучи максимізувати довгострокову винагороду. Основна мета агента – накопичити якомога більше винагород, обираючи оптимальну стратегію. Типові застосування включають робототехніку, ігрові AI, автономні транспортні засоби та системи управління.

4. Напівконтрольоване навчання (Semi-supervised Learning)

При напівконтрольованому навчанні частина навчальних даних розмічена, а решта точок даних не розмічені. Модель використовує розмічені дані, щоб навчитися робити прогнози, а потім застосовує нерозмічені дані для виявлення закономірностей і зв'язків. Цей формат зазвичай застосовується у додатках класифікації тексту та сегментації зображень.

2.1.2 Нейронні мережі та глибоке навчання в ШІ

Нейронні мережі (НМ, англ. Neural Networks) - це обчислювальні моделі, що імітують структуру та функціонування людського мозку. Вони складаються

з великої кількості взаємопов'язаних вузлів (нейронів), організованих у шари, кожен з яких обробляє інформацію та передає результати до наступного шару.

Основна архітектура нейронної мережі включає три типи шарів:

- Вхідний шар – отримує вхідні дані, де кожен нейрон відповідає одній ознаці вхідних даних.
- Приховані шари – виконують більшу частину обчислювальної роботи, перетворюючи вхідні дані у форму, придатну для вихідного шару.
- Вихідний шар – генерує кінцевий результат моделі, формат якого залежить від конкретного завдання (класифікація, регресія тощо).

Процес роботи нейронної мережі включає два основних етапи:

1. Пряме поширення (Forward Propagation) – дані проходять через мережу від вхідного до вихідного шару. На кожному нейроні виконується лінійне перетворення (зважена сума входів плюс зміщення), після чого застосовується функція активації для введення нелінійності.
2. Зворотне поширення (Backpropagation) – мережа обчислює градієнти функції втрат відносно кожної ваги та зміщення, використовуючи правило ланцюга диференціювання. Це дозволяє визначити, наскільки кожен параметр впливає на помилку[36].

Глибоке навчання (англ. Deep Learning, DL) – це підмножина машинного навчання, яка використовує багатошарові нейронні мережі для автоматичного виявлення складних патернів у даних. На відміну від традиційного машинного навчання, яке значною мірою покладається на ручну розробку ознак, моделі глибокого навчання можуть автоматично виявляти представлення, необхідні для виявлення ознак або класифікації.

Існує два основних типи систем глибокого навчання з різною архітектурою:

1. Згорткові нейронні мережі (CNN) складаються з трьох груп шарів:
    1) Згорткові шари витягують інформацію з введених даних за допомогою попередньо налаштованих фільтрів

2) Шари пулінгу знижують розмірність даних

3) Повнозв'язані шари формують додаткові нейронні шляхи

2. CNN ефективні для розпізнавання зображень, комп'ютерного зору та виявлення просторових патернів.

Рекурентні нейронні мережі (RNN) мають архітектуру у вигляді послідовності взаємопов'язаних рекурентних блоків, що утворюють направлений цикл. На кожному часовому кроці блок приймає поточний вхід і об'єднує його з попереднім прихованим станом, формуючи новий прихований стан. Це дозволяє мережі уловлювати часові залежності та шаблони. RNN використовуються для аналізу часових рядів, обробки тексту та машинного перекладу.

2.1.3 Ключові технології ШІ для забезпечення якості

Для застосування у процесах забезпечення якості програмного забезпечення найбільш релевантними є наступні технології ШІ (рис. 2.2):

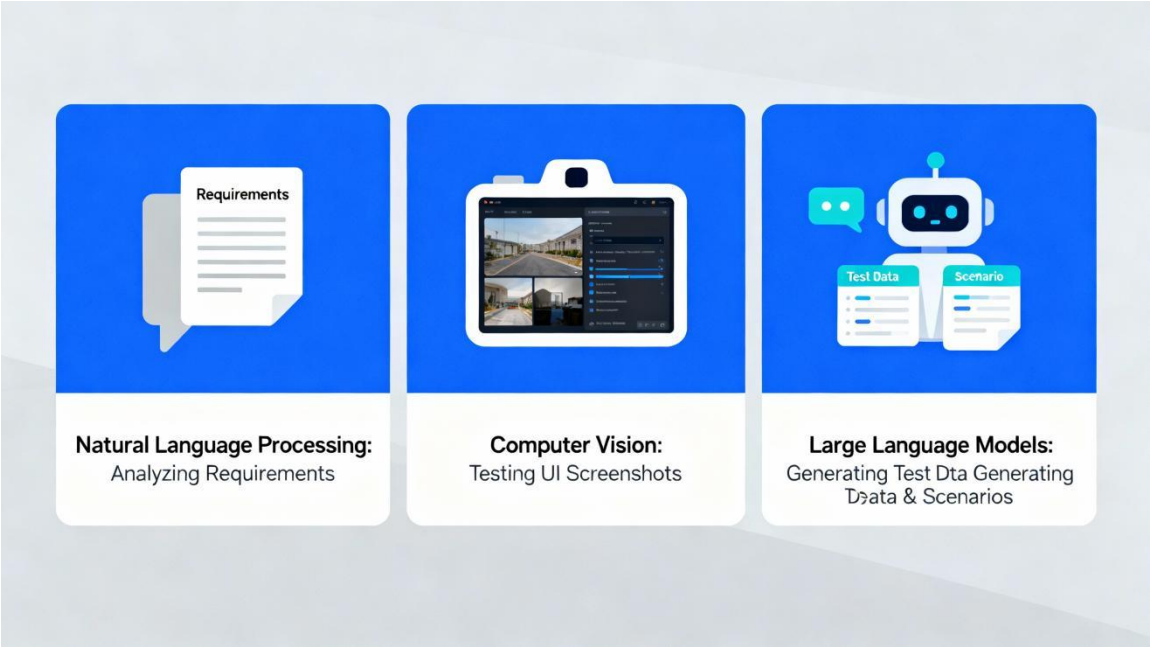


Рис. 2.2 - Ключові технології ШІ для забезпечення якості програмного забезпечення

Обробка природної мови (Natural Language Processing, NLP) – навчає комп'ютери розуміти написану та усну мову і генерувати відповіді, подібні до

людських. У контексті QA, NLP використовується для аналізу вимог, генерації тестової документації та обробки текстових звітів про помилки.

Комп'ютерний зір (Computer Vision, CV) – дозволяє системам ШІ бачити та ідентифікувати об'єкти. У тестуванні програмного забезпечення комп'ютерний зір застосовується для візуального тестування користувацького інтерфейсу, порівняння скріншотів та виявлення візуальних аномалій.

Великі мовні моделі (LLM) – це масштабні моделі, створені для генерації тексту. LLM побудовані на базових моделях (foundation models), які навчаються на величезних обсягах сирих, неструктурованих, нерозмічених даних. У QA-процесах LLM використовуються для автоматичної генерації тестових сценаріїв, створення тестових даних та аналізу бізнес-вимог.

Штучний інтелект та його ключові технології – машинне навчання, нейронні мережі та глибоке навчання – формують теоретичну основу для впровадження інтелектуальних систем у процеси забезпечення якості програмного забезпечення. Розуміння різних типів навчання (з учителем, без учителя, з підкріпленням) та архітектур нейронних мереж (CNN, RNN, трансформери) є критично важливим для ефективного застосування AI-технологій у QA-практиках. Спеціалізовані технології, такі як NLP, комп'ютерний зір та великі мовні моделі, відкривають нові можливості для автоматизації тестування, аналізу вимог та прогнозування дефектів, що докладно розглядатиметься у наступних підрозділах роботи.

2.2 Моделі і алгоритми ШІ, що застосовуються в QA

Впровадження моделей штучного інтелекту у процеси забезпечення якості програмного забезпечення вимагає глибокого розуміння різноманітних алгоритмів і методик машинного навчання, які можна застосовувати для конкретних задач тестування. Відбір і конфігурація правильної моделі є критично важливим для досягнення оптимальної ефективності, точності та масштабованості QA-процесів. Залежно від специфіки завдання — від

прогнозування дефектів до автоматичної класифікації багів — кожна модель пропонує унікальні переваги та обмеження.

2.2.1 Класичні алгоритми машинного навчання для QA

1. Decision Trees (Дерева рішень)

Дерева рішень — це непарамежрична модель, яка рекурсивно розділяє дані на основі значень ознак для прогнозування цільової змінної. У контексті тестування програмного забезпечення дерева рішень застосовуються для класифікації дефектів і визначення модулів з високим ризиком помилок. Алгоритм аналізує метрики коду (наприклад, кількість рядків коду, циклічну складність) і історичні дані про дефекти, щоб передбачити, які програмні компоненти найімовірніше матимуть проблеми. Перевага дерев рішень полягає в їхній інтерпретованості — результати можна легко пояснити менеджерам та розробникам, що критично важливо в корпоративних середовищах[12]. Проте дерева рішень схильні до перезапозичення (overfitting) на малих наборах даних.

2. Random Forest (Випадковий ліс)

Random Forest — це ансамблевий метод, який комбінує декілька дерев рішень для покращення точності та стійкості прогнозів. У задачах прогнозування дефектів Random Forest показує переважну точність (приблизно 84%) порівняно з поодинокими деревами рішень. Алгоритм автоматично визначає найбільш значущі ознаки коду (наприклад, частота змін, активність розробників, щільність дефектів) і використовує їх для зваженого голосування у передбаченні. Ця модель добре працює на великих і різнорідних наборах даних, що є типовим для QA-проектів у великих організаціях[7].

3. Support Vector Machine (SVM)

Support Vector Machine — це потужний алгоритм для класифікації, який находить оптимальну гіперплощину для розділення класів у багатовимірному просторі ознак. SVM особливо ефективна для класифікації типів дефектів (наприклад, функціональні, продуктивності, безпеки, інтерфейсу) та визначення рівня критичності багів на основі їхніх атрибутів. Модель досягає точності

близько 82%, що робить її надійним вибором для рутинних завдань класифікації у QA. Однак SVM може бути обчислювально інтенсивною на дуже великих наборах даних.

4. Naive Bayes

Naive Bayes — це ймовірнісний класифікатор, заснований на теоремі Байєса, який припускає незалежність ознак. У тестуванні програмного забезпечення Naive Bayes застосовується для швидкої класифікації звітів про дефекти на основі текстового опису та метаданих. Алгоритм обчислює ймовірність того, що баг належить до певного класу, використовуючи початкові ймовірності та умовні ймовірності ознак. Хоча точність Naive Bayes дещо нижча (близько 76%), він надзвичайно швидкий в обробці і добре масштабується для систем реального часу з великим обсягом багів.[32]

5. K-Nearest Neighbors (KNN)

K-Nearest Neighbors — це простий алгоритм, який класифікує новий об'єкт на основі його подібності до k найближчих об'єктів у навчальному наборі. У QA KNN застосовується для привілізації тестових випадків: тести, які математично подібні до тестів, що раніше знайшли дефекти, отримують вищий пріоритет. Це дозволяє фокусувати обмежені ресурси тестування на найбільш перспективних сценаріях. KNN також використовується для виявлення аномалій у поведінці системи.

6. XGBoost (Extreme Gradient Boosting)

XGBoost — це продвинутий ансамблевий метод, заснований на градієнтному бустингу, який послідовно будує дерева для коригування помилок попередніх моделей. У контексті прогнозування дефектів XGBoost показує виключну точність близько 87%, що робить його одним з найефективніших алгоритмів для цієї задачі. Модель здатна обробляти нелінійні взаємозв'язки між ознаками та цільовою змінною, що критично важливо для складних систем тестування. XGBoost також надає важливість ознак, дозволяючи командам QA розуміти, які метрики коду найбільше впливають на вероятність дефектів.

Табл. 2.1 - Порівняння класичних алгоритмів ML та QA

Алгоритм	Категорія	Основне застосування в QA	Точність (%)	Складність
Decision Trees	Класичне ML	Класифікація дефектів, передбачення модулів з ошибками	79	Низька
Random Forest	Класичне ML	Прогнозування дефектів, привілізація тестів	84	Середня
Support Vector Machine (SVM)	Класичне ML	Класифікація багів, прогнозування критичності	82	Середня
Naive Bayes	Класичне ML	Класифікація типів дефектів, тріаж багів	76	Низька
K-Nearest Neighbors (KNN)	Класичне ML	Привілізація тестових випадків за подібністю	75	Низька
Gradient Boosting (XGBoost)	Ансамблеве навчання	Прогнозування дефектів, вибір ознак	87	Висока

Продовження табл. 2.1

Convolutional Neural Networks (CNN)	Глибоке навчання	Візуальне тестування UI, порівняння скріншотів	89	Висока
Recurrent Neural Networks (RNN)	Глибоке навчання	Аналіз послідовних даних, обробка логів	86	Висока
LSTM Networks	Глибоке навчання	Генерація тестових сценаріїв, аналіз часових рядів	88	Висока
Genetic Algorithms	Еволюційні алгоритми	Оптимізація порядку виконання тестів	80	Середня

2.2.2 Глибокі нейронні мережі для автоматизації тестування

1. Convolutional Neural Networks (CNN) для візуального тестування

Convolutional Neural Networks спеціалізуються на обробці растрових даних, що робить їх ідеальними для візуального тестування користувацьких інтерфейсів. CNN навчаються розпізнавати шаблони у піксельних даних, що дозволяє їм виявляти візуальні регресії, розбіжності макетів та неправильно відтворені графічні елементи. Типова архітектура CNN включає згорткові шари для виявлення локальних особливостей, шари пулінгу для зменшення

розмірності та повнозв'язні шари для остаточної класифікації. Модель досягає точності близько 89-91% при виявленні візуальних аномалій, суттєво перевищуючи ручну перевірку скріншотів.

3. Recurrent Neural Networks (RNN) та LSTM для аналізу послідовних даних

Recurrent Neural Networks спеціалізуються на обробці послідовностей даних, таких як логи тестування, історія змін коду та часові ряди метрик продуктивності. RNN мають рекурентні з'єднання, які дозволяють їм запам'ятовувати попередні стани, що критично важливо для виявлення залежностей у часових послідовностях. Однак базові RNN часто страждають від проблеми втрати градієнтів при обробці довгих послідовностей.

Long Short-Term Memory (LSTM) — це спеціалізована архітектура RNN, яка вирішує цю проблему через механізми клітини та вентилів. LSTM ефективно аналізують логи тестування для виявлення аномальних патернів поведінки системи, що может вказувати на потенційні дефекти. Модель особливо корисна для автоматичної генерації тестових сценаріїв на основі спостережуваних користувацьких потоків. Точність LSTM-моделей у прогнозуванні відмов системи досягає 88%.

Трансформери для обробки природної мови у аналізі вимог (на базі механізму Attention) стали революційними для обробки природної мови і знаходять все більше застосувань у QA. Трансформери здатні аналізувати текстові вимоги, виявляти неоднозначності та конфлікти у специфікаціях, а також генерувати тестові сценарії на основі описів бізнес-логіки[3]. Модель обробляє всю послідовність одночасно (на відміну від RNN, які обробляють послідовно), що забезпечує вищу точність та швидкість. Популярні моделі, такі як BERT та GPT, можуть аналізувати вимоги з точністю до 85-92%.

2.2.3 Спеціалізовані методи для конкретних задач QA

Прогнозування дефектів за допомогою гібридних моделей. Найбільш ефективний підхід до прогнозування дефектів часто комбінує декілька

моделей. Гібридні методи, які поєднують класичне ML з елементами глибокого навчання, досягають точності близько 92%[18].

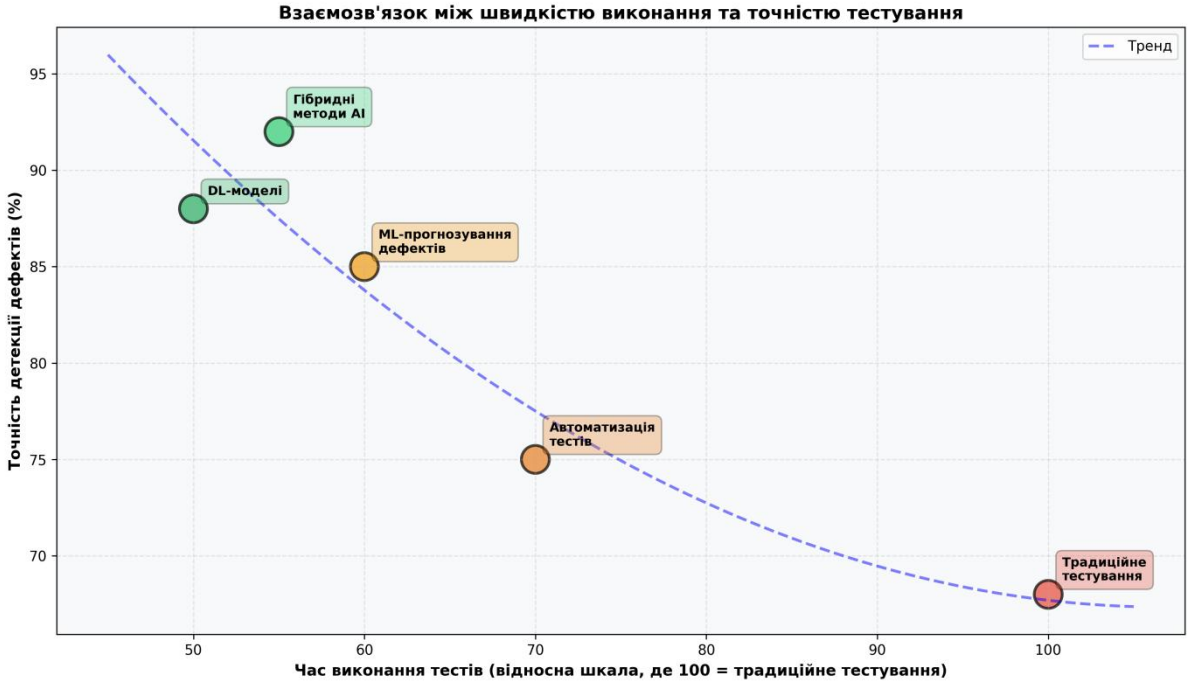


Рис. 2.3 - Взаємозв’язок між швидкістю виконання та точністю тестування

Рис. 2.3 демонструє, що гібридні методи забезпечують оптимальний баланс між точністю (92%) і швидкістю обробки.

Процес включає:

1. Екстракція ознак: Обчислення метрик коду (складність, покриття, щільність коментарів) та аналіз історії (частота змін, активність розробників)
2. Попередня обробка: Нормалізація даних та обробка пропусків
3. Обучення моделі: Тренування XGBoost та нейронних мереж паралельно
4. Ансамблювання: Комбінація прогнозів через зважене голосування
5. Постійне перенавантаження: Адаптація моделі до нових даних по мірі їх надходження

Для виявлення нетипових патернів поведінки у логах використовуються методи невиглядованого навчання:

Isolation Forest: Розщеплює дані випадковим чином для виявлення редких точок (аномалій). Точність — близько 82-87%.

One-Class SVM: Вивчає межі нормальної поведінки та флагує все, що виходить за межі. Точність — близько 80-85%.

Autoencoders (нейронні мережі): Навчаються відтворювати нормальні логи, виявляючи роздутості при кодуванні аномальних логів. Точність — близько 85-90%.

Natural Language Processing (NLP) моделі аналізують текстові описи багів, логи помилок та звіти користувачів для автоматичної класифікації та тріажу. Модель обчислює схожість звіту про баг з попередніми звітами, прогнозує тип дефекту (функціональний, продуктивності, безпеки) та рекомендує рівень критичності. Сучасні LLM-моделі, такі як GPT та BERT, досягають точності 80-92% при цьому завданні, суттєво прискорюючи процес обробки багів в великих проектах.[4]

Табл 2.2 - Порівняння методів прогнозування дефектів за точністю та швидкістю

Метод	Точність	Помилки першого роду	Час обробки	Обсяг даних
Статистичний аналіз	65%	35%	Швидко	Малий
Аналіз метрик коду	72%	28%	Швидко	Малий
Аналіз історії змін	74%	26%	Швидко	Середній
Машинне навчання	85%	15%	Середньо	Великий

Кінець табл. 2.2

Глибоке навчання	88%	12%	Повільно	Дуже великий
Гібридні методи	92%	8%	Середньо	Дуже великий

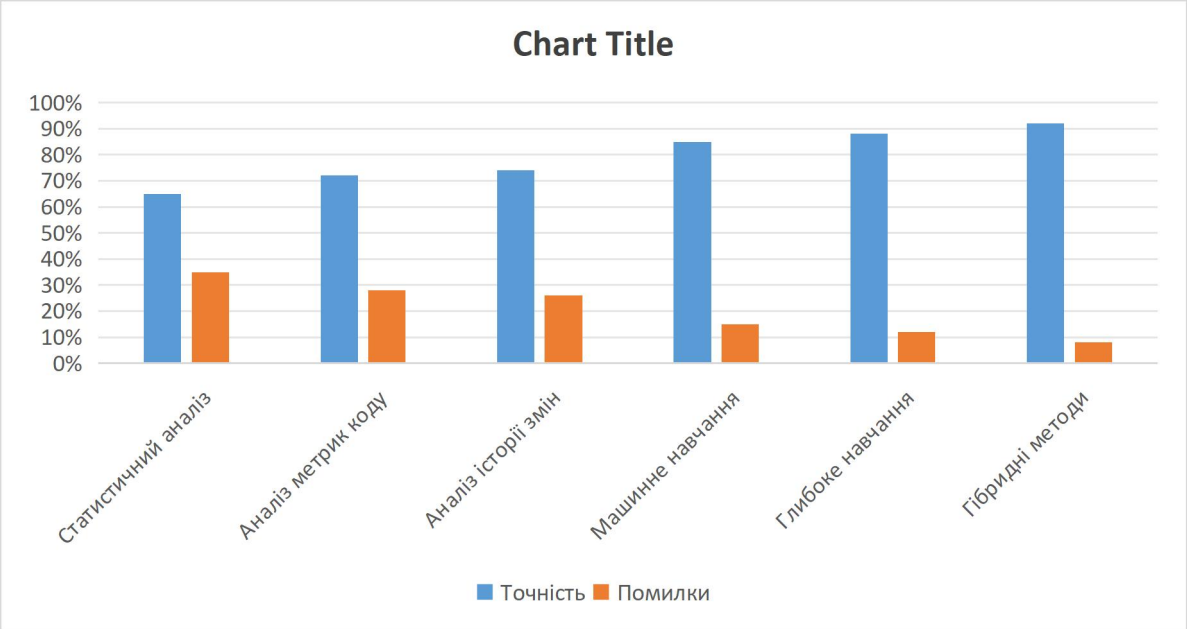


Рис. 2.4 - Графік порівняння методів

Як показано на рис. 2.4, використання AI-методів збільшує точність детекції дефектів з 68% до 92%.

2.2.4 Застосування AI-моделей для оптимізації тестування

Привілізація тестів на основі ризику. Машинні моделі можуть передбачати вероятність дефектів для кожного тесту на основі ознак змін коду, метрик якості та історичних даних про помилки. Це дозволяє командам вибирати найбільш релевантні тести першими, скорочуючи час виконання тестового набору на 30-50%, зберігаючи при цьому покриття ризику.

Рекурентні нейронні мережі та генеративні моделі (на базі трансформерів) навчаються на існуючих тестових сценаріях і користувацьких потоках, щоб автоматично генерувати нові тести. Такі моделі можуть:

- Визначати крайні випадки, які розробники часто пропускають
- Адаптуватися до змін в інтерфейсі без ручного оновлення

- Генерувати тести на основі звичайної мови опису функціональності

Точність автоматично генерованих тестів досягає 70-85% при мінімальному втручанні людини.

Управління флейкі-тестами. Flaky-тест, буквально «бавовняний», «розсипається на шматочки», в індустрії IT-тестування означає нестабільний, ненадійний тест, який іноді «pass», іноді «fail», і важко зрозуміти, за якою закономірністю. Вбивця часу тестувальника, джерело нервозності в команді. Флейкі-тести (intermittent failures) є надзвичайно дорогими для організацій, оскільки змушують команди витрачати час на налагодження їх замість концентрації на реальних дефектах. Машинні моделі аналізують історію виконання тестів, визначають несталі помилки та пропонують заходи для стабілізації (наприклад, збільшення тайм-аутів, удосконалення чекерів, виявлення расових умов).

Вибір правильної AI-моделі для конкретної задачі QA є критично важливим для досягнення оптимальних результатів. Класичні алгоритми ML (Decision Trees, Random Forest, SVM) пропонують хорошу інтерпретованість та швидкість для базових завдань класифікації. Ансамблеві методи, такі як XGBoost, досягають високої точності в прогнозуванні дефектів. Глибокі нейронні мережі (CNN, RNN, LSTM, Трансформери) відкривають нові можливості для автоматизації складних задач, таких як візуальне тестування, аналіз послідовних логів та обробка природної мови.

2.3 Переваги і виклики впровадження ШІ в QA

Впровадження штучного інтелекту у процеси забезпечення якості програмного забезпечення є складною і багатовимірною задачею, яка перед собою ставить одночасно чисельні перспективи та суттєві виклики. Розуміння цих факторів є критично важливим для прийняття обґрунтованих рішень щодо інвестування у AI-технології для QA-процесів[15]. З одного боку, організації отримують можливість суттєво підвищити ефективність тестування та прискорити цикли розробки. З іншого боку, вони стикаються з викликами,

пов'язаними з вартістю впровадження, дефіцитом спеціалістів та необхідністю адаптації організаційної культури.

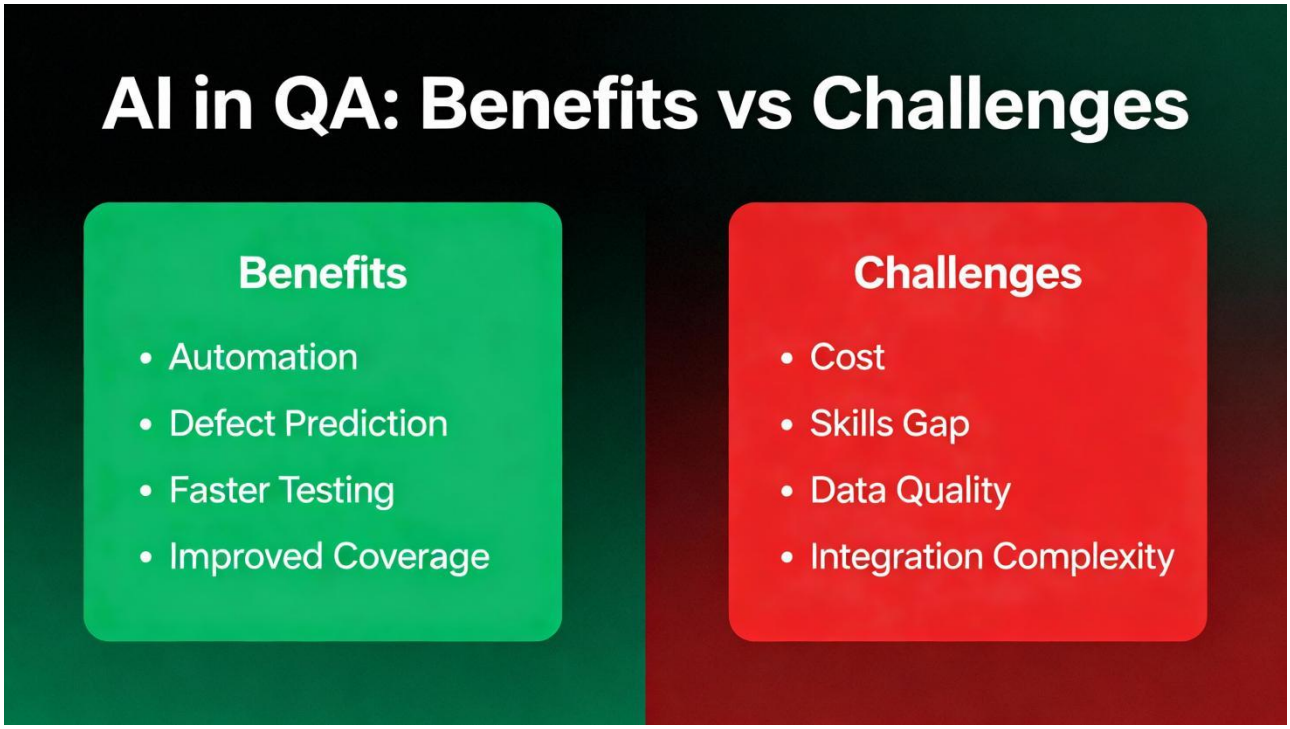


Рис. 2.5 - Баланс переваг та викликів при впровадженні AI у QA

2.3.1 Переваги впровадження штучного інтелекту в QA

Одна з найбільш очевидних переваг AI у тестуванні — це здатність до повної автоматизації рутинних завдань. На відміну від традиційних методів, які вимагають значних людських ресурсів для ручного виконання тестів, регресійного тестування та документування результатів, AI-системи можуть автоматично генерувати, виконувати та аналізувати тестові сценарії. Це призводить до 30-40% підвищення продуктивності та дозволяє командам QA зосередитися на більш складних та стратегічних завданнях, таких як дослідницьке тестування та оптимізація бізнес-логіки. Результатом є прискорення циклів розробки та скорочення часу виведення продукту на ринок на 40-50%[44]. Розглянемо переваги впровадження ШІ в QA-процеси:

1. Прогнозування дефектів та проактивна превенція.

AI-моделі, навчені на історичних даних про дефекти, здатні ідентифікувати потенційні проблеми ще до того, як вони проявляться у

виробничому середовищі. Через аналіз метрик коду, історії змін та попередніх дефектів, системи на базі машинного навчання можуть рекомендувати високоризикові області для фокусованого тестування. Це дозволяє організаціям економити значні кошти, адже вартість виправлення дефекту, виявленого під час розробки, у 5 разів менша від вартості виправлення дефекту, знайденого у виробництві. Проактивний підхід до управління якістю зменшує також кількість звернень від користувачів та підвищує довіру до продукту.

2. Розширене тестове покриття та точність

Традиційні методи тестування часто не змогу охопити всі можливі сценарії та крайні випадки через обмеженість людських ресурсів та часу. AI-системи здатні обробляти значно більші обсяги даних та виявляти складні взаємозв'язки між компонентами системи, які могли б залишитися непомічені при ручному тестуванні. Результатом є 25-35% розширення тестового покриття та поліпшення якості виявлення дефектів на рівні 35-45% раніше в процесі розробки. AI-моделі також мінімізують помилки першого роду (false positives), демонструючи 50-60% зменшення у порівнянні з традиційними методами.

3. Оптимізація ресурсів та скорочення витрат

Впровадження AI дозволяє компаніям оптимізувати розподіл ресурсів тестування. Замість утримання великих команд для виконання одноманітних тестів, організації можуть перерозподілити персонал на завдання з вищою доданою вартістю. Це призводить до зменшення витрат на людський капітал на 30-40%, оскільки менша кількість тестувальників може виконувати багато більший обсяг тестування. На додачу, AI-системи здатні масштабуватися без пропорційного збільшення витрат, дозволяючи компаніям запускати тисячі тестів без значного збільшення операційних витрат.

4. Безперервне тестування та прискорення релізів

Інтеграція AI з CI/CD-конвеєрами забезпечує безперервне тестування на всіх етапах розробки. Це дозволяє виявляти регресії та нові дефекти у режимі реального часу, що ускладнює потрапляння багів у виробництво. Результатом є 50-70% збільшення частоти релізів та прискорення часу між виявленням

проблеми та її розв'язанням. Така гнучкість критично важлива для організацій, що працюють за методологією Agile та DevOps.

Табл. 2.3 - Ключові переваги впровадження AI в QA процеси (ROI* - Return on Investment, укр. - Рентабельність інвестицій)

Переваги	Опис	Вплив на ROI*	Якісна метрика
Автоматизація тестування	Автоматичне виконання рутинних завдань без людської участі	Високий	+30-40% продуктивності
Підвищення покриття тестами	Виявлення крайніх випадків та складних сценаріїв	Дуже високий	+25-35% охоплення
Прогнозування дефектів	Аналіз історичних даних для запобігання дефектам	Дуже високий	+35-45% раннього виявлення
Скорочення часу тестування	Скорочення часу циклу тестування на 40-50%	Дуже високий	-40-50% часу
Оптимізація ресурсів	Можливість запустити більше тестів з меншими ресурсами	Високий	-30-40% вартості
Самовтілення тестів	Автоматична адаптація тестів до змін в коді	Середній	-20-25% обслуговування
Безперервне тестування	Інтеграція з CI/CD для постійного тестування	Дуже високий	+50-70% частоти релізів

Кінець табл. 2.3

Зменшення помилок людини	Послідовні та надійні результати тестування	Дуже високий	-50-60% помилок першого роду

2.3.2 Виклики впровадження штучного інтелекту в QA

Розглянемо виклики впровадження ШІ в QA-процеси:

1. Вартість впровадження та ROI

Одним з основних викликів впровадження AI у QA є високий рівень початкової інвестиції. Витрати на придбання AI-інструментів, налаштування інфраструктури, інтеграцію з існуючими системами та навчання персоналу коливаються від USD 50,000 до USD 100,000 для організацій середнього розміру, а для великих підприємств можуть сягати USD 200,000-500,000[. Крім того, слід враховувати приховані витрати на управління змінами організації (20-30% від загальної інвестиції), навчання та розвиток персоналу, та довгострокову підтримку систем. Хоча ROI показує позитивні результати (300-500% за 2-3 роки), період окупності становить 4-12 місяців залежно від розміру організації[29].

2. Дефіцит кваліфікованих спеціалістів

Впровадження AI-систем вимагає глибоких знань у галузях машинного навчання, обробки даних та інженерії програмного забезпечення. Однак ринок праці демонструє гострий дефіцит таких спеціалістів. Команди QA, як правило, не мають досвіду роботи з AI-моделями та потребують значного навчання для ефективного використання цих технологій. Витрати на залучення зовнішніх консультантів або навчання внутрішніх кадрів можуть бути суттєвими, а часовий лаг до отримання результатів може тривати кілька місяців.

3. Залежність від якості навчальних даних

Точність AI-моделей напряму залежить від якості та повноти навчальних даних. Якщо дані про минулі дефекти неповні, непослідовні або невиважено

розроблені, модель буде давати неточні або упереджені результати. Це вимагає значних інвестицій у підготовку, очищення та анотацію даних перед розпочатком тренування моделі. У багатьох організаціях дані про якість розпорошені по різних системах, що ускладнює їх централізацію та структурування.

4. Складність інтеграції з існуючими системами

Інтеграція AI-рішень в існуючу інфраструктуру забезпечення якості часто виявляється складною задачею. Потребується адаптація CI/CD-конвеєрів, налаштування інтеграції з системами управління тестами, та взаємодія з іншими інструментами розробки. Процес інтеграції може тривати 2-4 місяці і потребувати залучення IT-фахівців та консультантів. Крім того, вимагається переробка існуючих процесів та робочих потоків для максимізації вирішення від AI.

5. Культурна резистентність та питання довіри

Впровадження AI часто стикається з культурною резистентністю як з боку QA-команд, так і з боку управління. Тестувальники можуть розглядати AI як загрозу своєму робочому місцю, а менеджери можуть сумніватися в надійності результатів, отриманих від "чорної скриньки". Вирішення цієї проблеми вимагає прозорої комунікації щодо того, як AI змінить природу їх роботи (переведе акцент з рутинних завдань на більш творче та стратегічне тестування) та демонстрації конкретних успіхів впровадження.

Табл. 2.4 - Основні виклики впровадження AI в QA та рекомендовані рішення

Виклики	Критичність	Вихідні дані	Рішення
Вартість впровадження	Дуже висока	USD 50-100K початкові, USD 30-100K/рік	ROI 300-500% за 2-3 роки
Дефіцит спеціалістів	Висока	Потребується ШІ фахівець та ML інженер	Аутсорс або навчання внутрішніх кадрів

Продовження табл. 2.4

Якість навчальних даних	Дуже висока	Залежить 80% точність від якості даних	Інвестувати у якість датасету
Інтеграція з системами	Висока	Потребує 2-4 місяці, USD 15-25K	Використовувати готові рішення
Культурна резистентність	Середня	Людська природа, змінити складно	Навчання та демонстрація успіхів
Довіра до результатів	Висока	Вимагає верифікації результатів	Встановити чіткі критерії якості
Складність налаштування	Висока	Потребує експертної консультації	Стартувати з простих використовуватись
Потреба в постійному навчанні	Середня	10-20 годин/місяць на розвиток	Виділити час для професійного розвитку

Дослідження показують, що, незважаючи на високі початкові витрати, ROI впровадження AI у QA є дуже привабливим. Для компаній середнього розміру (50-200 розробників) ROI (ROI - Return on Investment, укр. - Рентабельність інвестицій) у першому році становить 200-250%, а у років 2-3 зростає до 400-500%. Для великих організацій ці показники ще вищі, досягаючи 300-400% у першому році. Період окупності коливається від 4 до 12 місяців залежно від розміру компанії та ступеня впровадження.

$$ROI = \left(\frac{Total\ benefits - Total\ costs}{Total\ costs}\right) \times 100\% \qquad (2.1)$$

Формула ROI (2.1) вказує як рахувати рентабельність.

Основні драйвери ROI включають:

1. Скорочення витрат на персонал: Зменшення кількості тестувальників або перерозподіл на вище вартісні завдання призводить до економії USD 360,000 річно при скороченні команди на 4 осіб.

2. Зменшення дефектів, які потрапляють у виробництво: Раннє виявлення дефектів економить у середньому USD 30,000-50,000 на 100 дефектів, які були б виявлені в продакшені.

3. Прискорення циклів розробки: Скорочення часу тестування на 40-50% дозволяє компаніям випускати нові версії частіше, генеруючи додаткові доходи.

4. Підвищення якості: Більш якісні продукти призводять до зменшення відтоку користувачів та збільшення лояльності, що трансформується у підвищення довгострокових доходів.

2.3.3 Рекомендації для успішного впровадження

На основі аналізу переваг та викликів, для організацій, що розглядають впровадження AI у QA-процеси, рекомендується:

1. Провести детальну оцінку ROI для конкретної організації, враховуючи розмір, поточні витрати на QA та плани розширення розробки.

2. Почати з pilot-проекту на невеликій частині тестового набору для демонстрації вартості та зменшення опору змінам.

3. Інвестувати у навчання персоналу для підготовки QA-команди до роботи з AI-інструментами та адаптації до нових робочих процесів.

4. Вибрати відповідний AI-інструмент, що відповідає специфіці організації, замість спроб розробляти власне рішення "з нуля".

5. Встановити чіткі метрики успіху для вимірювання ROI, якості тестування, часу циклів розробки та задоволення команди.

6. Забезпечити організаційну підтримку через комунікацію, демонстрацію успіхів та переведення персоналу на більш цінні завдання.

Впровадження штучного інтелекту у процеси забезпечення якості програмного забезпечення представляє одночасно значні можливості та суттєві

виклики. Переваги, такі як автоматизація, прогнозування дефектів, розширене покриття тестування та скорочення витрат, роблять AI привабливим вибором для організацій, що прагнуть підвищити ефективність розробки. Однак виклики, пов'язані з вартістю впровадження, дефіцитом спеціалістів, якістю даних та культурною адаптацією, вимагають уважного планування та відповідного управління проектом.

Організації, які успішно подолають ці виклики, отримують значний конкурентний перевага через прискорення виходу на ринок, покращення якості продуктів та зниження витрат на операції. ROI, який становить 300-500% за 2-3 роки, робить AI-інвестиції економічно обґрунтованими для організацій будь-якого розміру, від стартапів до великих підприємств[37].

3 ПРАКТИЧНЕ ВПРОВАДЖЕННЯ ШІ У ПРОЦЕСИ ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір методів та інструментів тестування на основі ШІ

Вибір методів та інструментів тестування на основі штучного інтелекту є критично важливим рішенням, яке прямо впливає на ефективність QA-процесів, витрати на впровадження та довгострокову окупність інвестицій. На відміну від традиційних інструментів тестування, AI-рішення розрізняються за архітектурою, рівнем автоматизації, потребою в навичках та моделлю розгортання. Організація повинна провести ґрунтовний аналіз своїх специфічних потреб перед вибором інструменту, оскільки неправильний вибір може привести до втрати часу та коштів на впровадження[35].

3.1.1 Критерії для вибору AI інструментів

Процес вибору AI-інструменту повинен ґрунтуватися на чітких критеріях, що відповідають унікальним вимогам організації. Дослідження показують, що 83% команд первинно вибирають неправильний інструмент, втрачаючи при цьому 6 місяців та близько 2,4 млн доларів. Для мінімізації ризику рекомендується керуватися такими ключовими факторами:

1. Тип та складність програми. Тип застосування (веб, мобільний, мікросервіси, API) істотно впливає на вибір інструменту. Простим веб-форм з обмеженою взаємодією достатньо базової автоматизації, тоді як мікросервісні архітектури та SPA-додатки вимагають більш потужних AI-рішень для обробки динамічного контенту та складних сценаріїв.

2. Рівень навичок команди. Розрізняють три категорії: новички (потребують No-Code рішень), проміжні користувачі (Hybrid платформи) та продвинуті розробники (Code-based + AI). Невідповідність інструменту навичкам команди призводить до повільного впровадження та незадовільних результатів.

3. Масштаб проекту та кількість тестів. Малі проекти (100-500 тестів) можуть обійтись простішими рішеннями, тоді як великі системи з 2000+ тестами потребують масштабованих платформ з хмарною інфраструктурою.

4. Інтеграція з CI/CD конвеєрами. AI-інструмент повинен безпроблемно інтегруватися з існуючими системами (Jenkins, GitHub Actions, Azure DevOps). Складність інтеграції може значно збільшити вартість впровадження.

5. Модель розгортання (Cloud vs On-Premise). Хмарні рішення зазвичай простіше впроваджувати та масштабувати, однак на-premise рішення можуть бути обов'язковими для організацій з суворими вимогами до безпеки та конфіденційності.

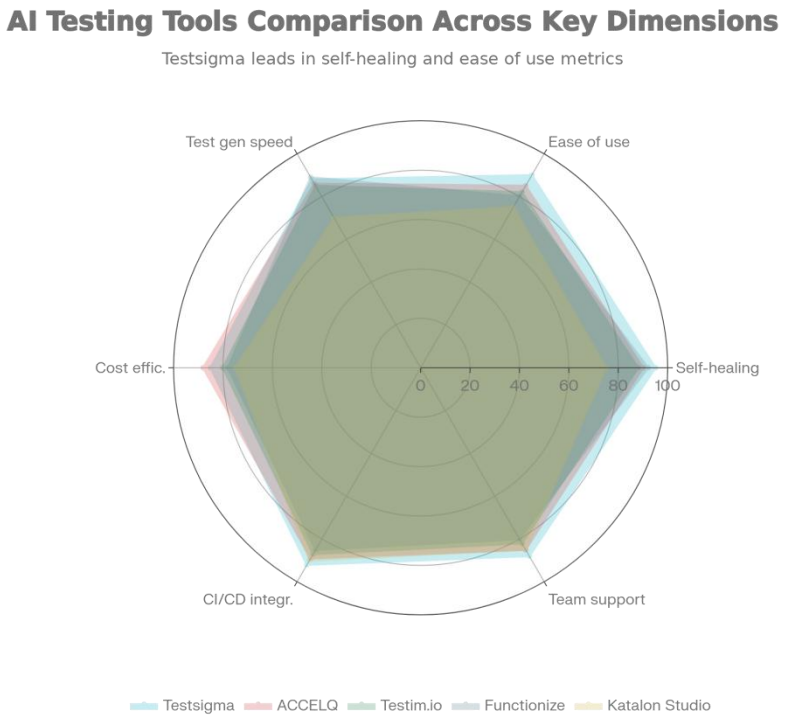


Рис. 3.1 - Радарна діаграма порівняння AI інструментів

3.1.2 Порівняння популярних AI інструментів тестування

На ринку існує декілька лідерів AI-рішень, кожне з яких має унікальні переваги та обмеження. Таблиця 3.1 надає детальне порівняння восьми найпопулярніших інструментів за ключовими характеристиками:

Testsigma – no-code платформа з штучним інтелектом, що дозволяє функціональним QA-спеціалістам писати тести на простій англійській мові без

необхідності програмування. Інструмент демонструє дуже низький поріг входу, високу точність самовідновлення та підтримує веб, мобільні та desktop-додатки. Вартість впровадження становить 30-60K USD на рік, що робить його привабливим для організацій з обмеженим бюджетом[13].

ACCELQ – хмарна платформа для безкодового тестування API та веб-інтерфейсів, спеціалізована на автоматизації критичних життєвих циклів тестування. Інструмент використовує натуральну англійську мову для створення тестів та пропонує предиктивний аналіз. Вартість складає 25-50K USD на рік, що робить його одним з найбільш економічних рішень[24].

Testim.io – ML-керована платформа, яка використовує машинне навчання для створення та підтримки автоматизованих тестів. Забезпечує самовідновлення тестів, інтелектуальне виявлення об'єктів та вбудовану відладку. Вартість 35-70K USD на рік[38].

Functionize – AI-native платформа, яка комбінує машинне навчання з людськими знаннями для створення і виконання тестів. Особливо ефективна для складних додатків. Вартість 40-80K USD на рік[11].

Katalon Studio – гібридна платформа, що поєднує код та no-code функції. Має чудову інтеграцію з CI/CD та підтримує веб, мобільні та desktop-тестування. Однак, самовідновлення тестів – тільки часткове. Вартість 50-100K USD на рік[6].

Табл. 3.1 - Порівняння AI інструментів тестування

Інструмент	Тип	Кривава навчання	Вартість рік (USD)	Самовсилення	Генерація тестів
Testsigma	No-Code AI	Дуже низька	30-60K	Так	AI
ACCELQ	Cloud-based AI	Низька	25-50K	Так	AI
Testim.io	ML-Powered	Середня	35-70K	Так	ML

Кінець табл. 3.1

Functionize	AI-Native	Низька	40-80K	Так	AI
Katalon Studio	Hybrid	Середня	50-100K	Частково	Manual
Selenium + ML	Code-based + AI	Висока	0-50K	Hi	Manual
TestCraft	No-Code AI	Дуже низька	25-50K	Так	GPT-4
Virtuoso QA	Cloud AI	Низька	50-100K	Так	AI

Організації часто займаються сумніви щодо обґрунтованості інвестування у AI-інструменти. Однак дослідження показують переконливий ROI для всіх розмірів проектів. Формула базового розрахунку ROI наведена вище (2.1).

Вигоди включають: зекономлені людино-години, менше дефектів у продакшені, прискорення циклу розробки та розширене тестове покриття[43].

Таблиця 3.2 демонструє очікувані параметри ROI в залежності від складності проекту:

Табл. 3.2 - Аналіз вартості та вигід за розміром проекту

Метрика / Розмір проекту	Вартість впровадження (місяці)	Вартість впровадження (USD)	ROI рік 1 (%)	Період окупності (місяці)	Продуктивність команди (+%)	Покриття тестами (+%)	Витрати на утримання рік
Простий	2	10-30K	150-200%	8	20-30	15-20	5-10K

Кінець табл. 3.2

Середній	4	50-100K	250-300%	12	30-50	25-35	20-30K
Складний	6	100-300K	300-400%	14	50-70	35-50	40-70K

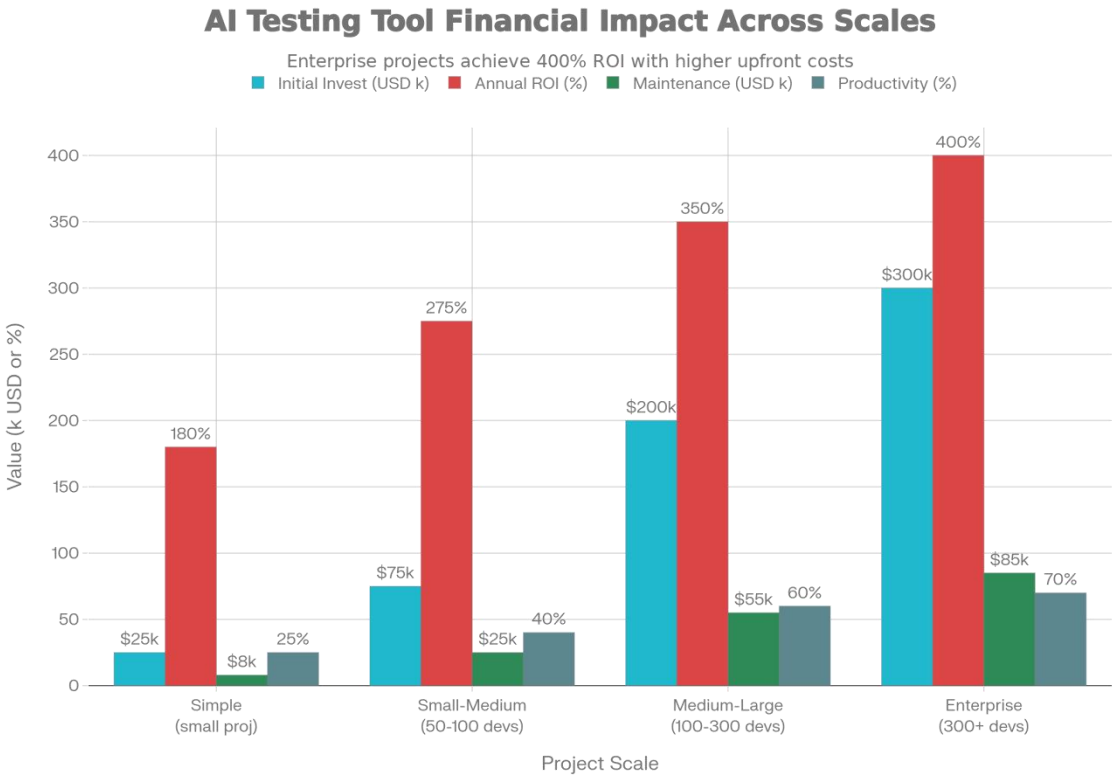


Рис. 3.2 - Аналіз вартості та ROI за розміром проекту

3.1.3 Процес прийняття рішення при виборі інструменту

Рекомендований процес вибору складається з п'яти етапів:

1. Визначення вимог (тиждень 1). Створіть детальний список функціональних та нефункціональних вимог на основі типу додатку, розміру команди та цілей бізнесу. Присвойте кожній вимозі вагу: «Обов'язкова» (1.0), «Важлива» (0.75), «Корисна» (0.5), «Не потребується» (0).

2. Складання списку потенційних інструментів (тиждень 2). На основі попередніх досліджень та аналізу попиту складіть список 5-7 потенційних інструментів, які задовольняють основні вимоги.

3. Первинна оцінка (тиждень 3). Проведіть оцінку за 100-бальною системою, розподіливши позиції наступним чином: Технічні можливості (30%), Функції AI (25%), Простота використання (20%), Вартість (15%), Підтримка (10%).

4. Pilot-тестування (2-4 тижні). Протестуйте 2-3 лідируючих кандидати на дійсних проектах, визначивши чіткі критерії успіху.

5. Фінальна оцінка та вибір (тиждень 5-6). Проаналізуйте результати pilot-тестування та виберіть інструмент, найкраще відповідаючий вимогам організації[20].

Вибір інструменту залежить від специфіки організації:

- Для стартапів та малих компаній: Testsigma, ACCELQ або TestCraft – простота впровадження та низька вартість є критичними.
- Для середніх компаній: Testim.io, Functionize – баланс між функціоналом та вартістю.
- Для великих підприємств: Virtuoso QA, Katalon Studio Enterprise – масштабованість, безпека та підтримка мають наивищий пріоритет.
- Для організацій з продвинутими розробниками: Selenium + ML-розширення, Robot Framework + AI-агенти – максимальна гнучкість і контроль[1].

Вибір правильного AI-інструменту для тестування є фундаментальним рішенням, яке визначає успіх впровадження. Процес вибору повинен базуватися на об'єктивній оцінці організаційних вимог, матеріальних обмежень та навичок команди. Дотримання структурованого підходу оцінки, проведення pilot-проектів та розрахунок очікуваного ROI допоможуть організаціям прийняти обґрунтоване рішення та мінімізувати ризики впровадження. Незалежно від обраного інструменту, успіх залежить від належного планування, навчання команди та поступового розширення покриття від пілотних проектів до повного впровадження на всій організації.

3.2 Реалізація і налаштування AI-систем у QA

Реалізація та налаштування штучного інтелекту у процеси забезпечення якості програмного забезпечення є складним і багатоетапним процесом, який

вимагає ґрунтовного планування, технічних знань та організаційної підготовки. На відміну від простої установки традиційних інструментів, впровадження AI-систем потребує інтеграції з існуючою інфраструктурою, навчання моделей на реальних даних та постійного моніторингу результатів. Успішна реалізація залежить від розуміння архітектури системи, дотримання кращих практик та адаптації до специфіки організації[22].

3.2.1 Архітектура AI-системи для тестування

Типова AI-система тестування складається з п'яти основних шарів, кожен з яких виконує критичну функцію[10]:

1. Шар розробки тестів (Test Development Layer)

- Аналіз вимог: Система розглядає специфікації продукту, користувацькі історії та попередні тести для розуміння функціональності
- Генерація тестових кейсів: AI автоматично генерує нові тестові сценарії на основі вимог, використовуючи LLM та машинне навчання
- Підготовка тестових даних: Система створює репрезентативні датасети, включаючи граничні випадки та аномальні стани

2. Шар виконання (Execution Layer)

- Оркестрація тестів: Управління порядком виконання, розпаралелюванням та синхронізацією тестів
- Механізм самовідновлення: AI розпізнає зміни у інтерфейсі та автоматично адаптує тести без ручного втручання
- Паралельне виконання: Розподіл тестів на кілька виконавців для прискорення циклу

3. Шар AI/ML (AI/ML Engine Layer)

- ML-моделі: Натреновані моделі для прогнозування дефектів, класифікації багів, визначення пріоритетів
- Механізм прогнозування: Система аналізує історичні дані та передбачає ризик-критичні області коду

- Knowledge Base: Накопичена інформація про архітектуру додатку, паттерни поведінки, типові помилки

4. Шар інтеграції (Integration Layer)

- CI/CD конвеєр: Автоматичне запускання тестів в процесі розробки
- Система відслідкування дефектів: Автоматичне створення багів в Jira, Azure DevOps або подібних системах
- Звітування: Генерація детальних звітів про результати тестування з метриками якості

5. Шар моніторингу (Monitoring Layer)

- Дашборд метрик: Візуалізація KPI у реальному часі
- Система сповіщень: Оповіщення про аномалії, невдалі тести, проблеми з інфраструктурою
- Відслідкування продуктивності: Аналіз трендів виконання системи

3.2.2 Етапи впровадження AI-систем

Структурований підхід до впровадження складається з 7 критичних етапів, загальною тривалістю 5-7 місяців[8]:

Етап 1: Оцінка поточного стану (1-2 тижні)

На цьому етапі команда аналізує існуючи QA-процеси, виявляє вузькі місця та можливості для вдосконалення. Ключові дії включають:

- Документація існуючих тестових наборів (кількість тестів, типи, тривалість виконання)
- Аналіз витрат на утримання тестів (скільки часу витрачається на оновлення, налагодження)
- Оцінка якості команди та наявних навичок у галузі автоматизації
- Визначення поточних болів: які типи помилок найчастіше пропускаються, скільки часу займає регресійне тестування

Результат: Детальна базова документація (baseline), що служить точкою порівняння для майбутніх вимірювань успіху.

Етап 2: Планування впровадження (2-3 тижні)

На основі оцінки складається детальний план впровадження:

- Визначення SMART цілей (наприклад: скоротити час регресійного тестування на 40%, покрити 80% функціоналу автоматизованими тестами)
- Вибір AI-інструменту (з урахуванням критеріїв з розділу 3.1)
- Складання бюджету та графіка роботи
- Визначення критичного шляху: які модулі/функціональності тестувати першими

Результат: Затверджений план впровадження, розподілені ролі та відповідальність.

Етап 3: Підготовка інфраструктури (2-4 тижні)

Технічна підготовка серверної інфраструктури:

- Налаштування тестового середовища (test, staging, UAT) з ізоляцією від продакшену
- Установка необхідного ПЗ: IDE, браузери, драйвери (ChromeDriver, GeckoDriver), залежності
- Налаштування CI/CD конвеєра (Jenkins, GitLab CI, Azure DevOps) з автоматичним запуском тестів
- Конфігурація системи логування та моніторингу (ELK Stack, Prometheus, New Relic)
- Впровадження Infrastructure as Code (IaC) з використанням Docker, Kubernetes для репродуцюючих середовищ

Результат: Повнофункціональна інфраструктура, готова до налаштування інструменту.

Етап 4: Налаштування інструменту (1-2 тижні)

Конфігурація обраного AI-інструменту:

- Створення об'єктів (object repository) для елементів UI з описовими назвами
- Налаштування параметрів самовідновлення (див. табл. 3.6)
- Інтеграція з системами управління тестами (TestRail, Zephyr) та дефектів (Jira, Azure DevOps)
- Налаштування оповіщень та дашбордів

- Тестування базової функціональності на простих прикладах

Результат: Готовий до використання AI-інструмент з базовою конфігурацією.

Етап 5: Пілот-проект (4-6 тижнів)

Критичний етап, де здобуваються реальні дані про ефективність:

- Вибір 2-3 критичних функцій/модулів (не занадто прості, не занадто складні)
- Автоматизація тестових сценаріїв для цих модулів з використанням AI-генерації
- Паралельне виконання AI та традиційних тестів для порівняння результатів
- Вимірювання KPI: час виконання, точність, покриття, витрати. Збір зворотного зв'язку від команди щодо usability та проблем

Результат: Данні про ROI, уроки вивчені, список рекомендацій для масштабування.

Етап 6: Масштабування (2-3 місяці)

- Розширення покриття на все більшу частину проекту:
- Постійне навчання моделей AI на нових даних (тести, дефекти, вимоги)
- Розширення тестового покриття від пілотних модулів на всю систему
- Масштабування обчислювальних ресурсів за потребою
- Впровадження автоматичного запуску тестів у CI/CD з блокуванням розгортання при критичних помилках
- Навчання всієї QA-команди роботі з AI-інструментом

Результат: Повне впровадження AI-тестування на всій системі.

Етап 7: Оптимізація (постійно)

Постійне вдосконалення та оптимізація системи:

- Моніторинг метрик і коригування конфігурацій на основі даних
- Поточне навчання ML-моделей на накопичених даних
- Аналіз найчастіших типів помилок та налаштування фокусу тестування
- Управління технічним боргом у тестовому коді
- Планування наступних версій AI-функцій
- Компоненти архітектури та їх налаштування

3.2.3 Моніторинг та оцінка результатів

Структурований моніторинг передбачає відслідкування набору KPI протягом впровадження.

Критичні KPI для відслідкування:

1. Успішність запуску тестів (Target: 94-96% до 6-го місяця)
2. Базис: 75-80% (традиційні тести часто бувають нестабільні)
3. Покращення: Механізм самовідновлення та інтелектуальні перевірки знижують хибні негативи
4. Покриття функціоналу (Target: 85-92% до 6-го місяця)
5. AI генерує тесту для необхідних область, що розширює покриття на 40-50%
6. Виявлення дефектів (Target: 88-92% до 6-го місяця)
7. Предиктивна аналітика передбачає і концентрує тестування на ризик-критичних областях
8. Витрати на утримання тестів (Target: 10-15% часу до 6-го місяця)
9. Самовідновлення знижує вручну роботу над оновленням тестів
10. Хибні позитивні (Target: 2-4% до 6-го місяця)
11. Інтелектуальні перевірки та аналіз контексту зменшують false positives
12. Часу на розробку одного тесту (Target: 45-60 хв до 6-го місяця)
13. AI-генерація та готові шаблони прискорюють розробку в 3-4 рази
14. Продуктивність команди (Target: 180-220% до 6-го місяця)
15. Команда може зосередитися на стратегічних завданнях замість утримання тестів

Табл. 3.3 - KPI моніторинг

KPI	Базис (до AI)	Місяць 1	Місяць 3	Місяць 6
Успішність запуску тестів	75-80%	82-85%	90-93%	94-96%
Час виконання набору тестів	45-60 хв	40-50 хв	30-40 хв	20-30 хв
Покриття функціоналу	45-55%	60-70%	75-85%	85-92%
Виявленні дефекти	68-72%	78-82%	85-88%	88-92%
Хибні позитивні (false positives)	15-20%	8-12%	4-6%	2-4%
Витрати на утримання тестів	35-40% часу	25-30% часу	15-20% часу	10-15% часу
Часто на розробку 1 тесту	2-3 години	1,5-2 години	1-1,5 години	45-60 хвилин
Продуктивність команди	100%	120-130%	150-170%	180-220%

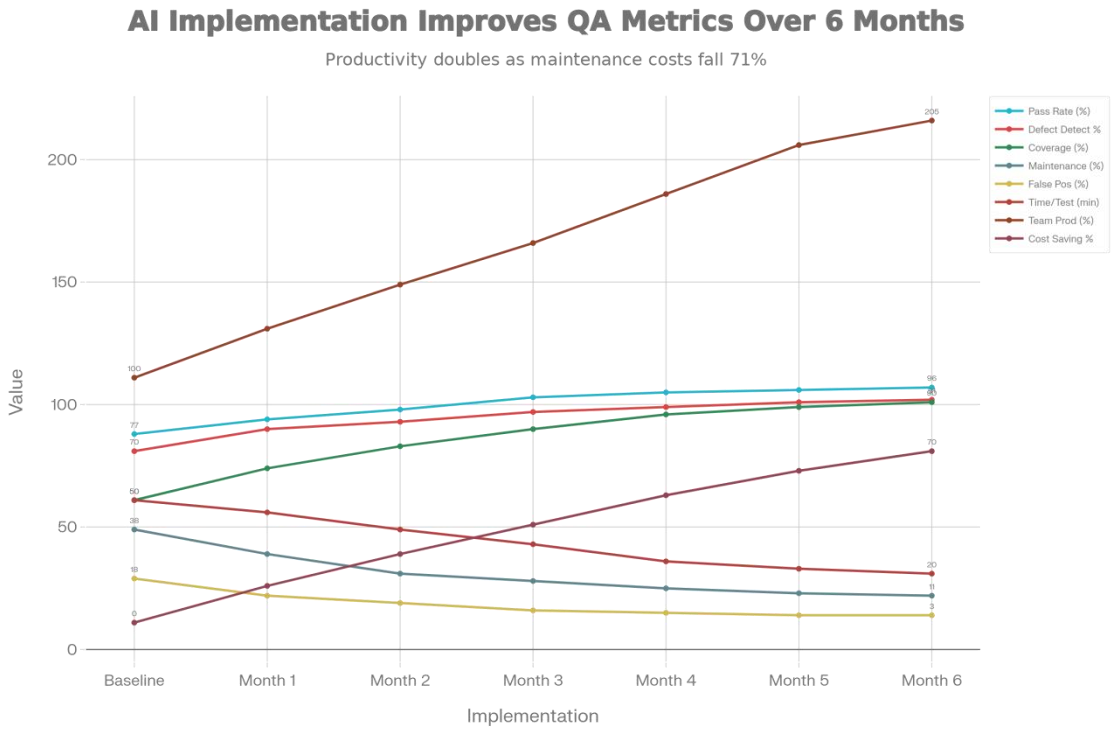


Рис. 3.3 - Динаміка метрик за 6 місяців

Впровадження AI-систем супроводжується низкою ризиків, які необхідно активно управляти:

Ризик 1: Початкова недовіра команди

- Рішення: Демонстрація успіхів на малих пілот-проектах, постійне навчання команди, прозорі дашборди результатів

Ризик 2: Низька якість тестових даних

- Рішення: Використання реальних даних, комбінування з синтетичними, регулярна валідація датасетів

Ризик 3: Надмірна автоматизація самовідновлення

- Рішення: Вибір консервативного чи збалансованого рівня автоматизації, людський контроль над критичними змінами

Ризик 4: Залежність від одного інструменту

- Рішення: Архітектурно відділити інструмент від логіки тестування, вибирати рішення з відкритими API

Ризик 5: Проблеми з інтеграцією CI/CD

- Рішення: Поступова інтеграція, починаючи з нічних запусків, моніторинг часу виконання

Реалізація AI-систем у тестування програмного забезпечення вимагає комплексного, структурованого підходу, що охоплює технічні, організаційні та людські аспекти. Успішне впровадження залежить від ясного розуміння архітектури системи, дотримання доведених етапів впровадження та постійного моніторингу метрик. Дотримання рекомендованих практик для налаштування самовідновлення, управління тестовими даними та інтеграції з CI/CD забезпечує мінімальні перебої та максимальну цінність від AI-інвестицій. При правильній реалізації організації досягають 40-70% прискорення тестування, 85-95% скорочення витрат на утримання та 200-500% розширення тестового покриття протягом 6 місяців, дозволяючи QA-командам перейти від реактивних до проактивних та стратегічних ролей у процесі розробки ПЗ[44].

3.3 Аналіз та інтерпретація результатів впровадження

Аналіз результатів впровадження AI-систем у QA є критично важливою фазою, яка трансформує збільшені дані у практичні рекомендації для подальшої оптимізації. Без структурованого аналізу організація не може визначити, чи досягла своїх цілей впровадження, де зосередити подальші зусилля та як обґрунтувати інвестицію перед керівництвом. Комплексний аналіз використовує як кількісні метрики (ROI, покриття тестами), так і якісне зворотне забезпечення від команди та користувачів[46].

Існує кілька наукових методів аналізу результатів впровадження AI-систем. Методи описані в табл. 3.4

Табл. 3.4 - Методи аналізу результатів впровадження AI-систем

Метод	Мета	Інструменти	Складність
Порівняльний аналіз (Before/After)	Порівняти базис з поточним станом	Excel, Tableau, Power BI	Низька

Продовження табл. 3.4

ROI аналіз	Розрахувати окупність інвестиції	Власні формули, Gartner калькулятор	Середня
Кореляційний аналіз	Виявити залежності між метриками	Python (Pandas, SciPy), SPSS	Середня
Тренд аналіз	Визначити тренди за часом	Matplotlib, Tableau, Google Sheets	Низька
Аналіз розподілу	Аналіз розподілу випадків/багів	Excel гістограми, Python seaborn	Низька
Якісний аналіз (опитування)	Зібрати відповідь команди та користувачів	Google Forms, SurveyMonkey	Низька
Регресійний аналіз	Прогнозувати вплив змін	Python (scikit-learn), R	Висока
Кластерний аналіз	Виявити групи подібних тестів	Python (scikit-learn), Orange	Висока

Аналіз результатів повинен зосереджуватися на наборі критичних KPI, які прямо пов'язані з цілями бізнесу:

1. Успішність тестів (Pass Rate) - Цільове: > 95%

Показує відсоток тестів, які успішно виконуються. Традиційно цей показник був 75-80% через нестабільні тести (flaky tests). При правильній конфігурації AI-систем, успішність повинна зросту до 94-96% за 6 місяців. Якщо цей показник залишається низьким, це сигнал про проблеми з якістю тестових даних або неправильну конфігурацію самовідновлення.

2. Покриття функціоналу (Test Coverage) - Цільове: > 85%

Відсоток функціоналу, охопленого автоматизованими тестами. Базис зазвичай 45-55%. AI-генерація тестів розширює покриття на 40-50%, досягаючи 85-92% за 6 місяців. Якщо покриття зростає повільно, розглянути активацію автоматичної генерації тестів для непокритих модулів.

3. Дефекти на 1000 ліній коду (Defect Density) - Цільове: < 20%

Метрика якості, що показує кількість дефектів на 1000 рядків коду. Низьке значення вказує на ефективне тестування. При правильній конфігурації AI-систем це значення скорочується з 4-5 до 1-2 дефектів на 1000 LOC за 6 місяців.

4. Витрати на утримання тестів (Maintenance Cost) - Цільове: < 15%

Відсоток часу команди QA, витраченого на оновлення та налагодження тестів замість їх розробки та виконання. Базис зазвичай 35-40%. Самовідновлення скорочує це до 10-15% за 6 місяців, звільняючи команду для стратегічної роботи.

5. Хибні позитивні (False Positives) - Цільове: < 5%

Відсоток тестів, які збивають помилку без реального дефекту. Це гасить довіру до автоматизації. Поступово скорочується з 15-20% до 2-4% за 6 місяців завдяки покращенню інтелектуальних перевірок та самовідновлення.

6. Продуктивність команди (Team Productivity Index) - Цільове: > 150%

Відносне покращення продуктивності команди QA порівняно з базисом (100%). За 6 місяців цей показник зростає до 200-220%, оскільки команда автоматизує більше тестів та скорочує утримання.

7. ROI - Цільове: > 300%

Окупність інвестицій. За 6 місяців добре впровадженої AI-системи ROI досягає 300-400%, що означає 3-4 кратний повернення інвестицій[21].

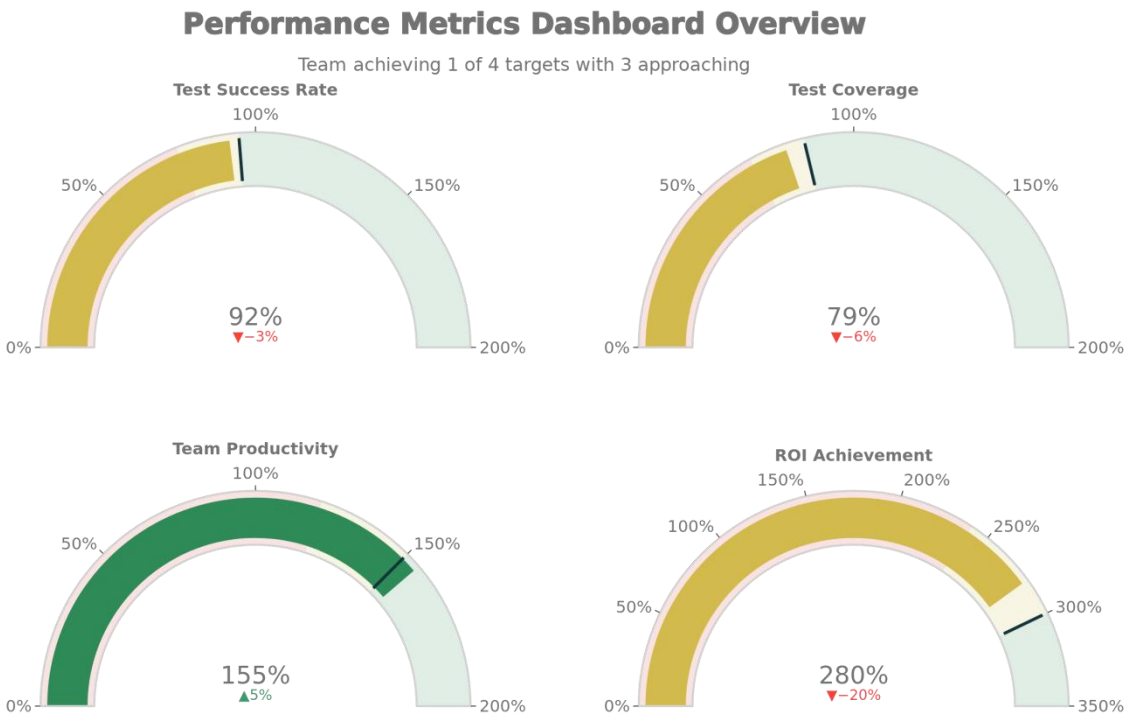
Ефективна комунікація результатів вимагає добре структурованих дашбордів та звітів:

Дашборд реального часу (Real-Time Dashboard) має включати:

- Граф успішності тестів за останні 7 днів
- Поточне покриття функціоналу (% выконанны до цілі)
- Кількість виявлених дефектів за останній день/тиждень
- Середній час виконання тесту (трендова лінія)
- Список найнестабільніших тестів (top 5 flaky)
- ROI прогрес (до цілі)
- Щомісячний звіт (Monthly Report)

Структура:

- Резюме (1 сторінка): 3-5 ключових досягнень, 2-3 проблемні області
- Метрики (2 сторінки): таблиця всіх KPI з тренду та інтерпретацією
- Розбір по модулям (1 сторінка): покриття та якість для кожного модуля
- Проблеми та рекомендації (1 сторінка): список виявлених проблем та рекомендацій
- ROI прогноз (1 сторінка): очікувані вигоди на основі поточного темпу



–

Рис. 3.4 - Дашборд KPI

При аналізі результатів важливо розуміти, як різні метрики впливають одна на одну:

Покриття vs Виявлення дефектів:

При розширенні покриття з 50% до 90%, виявлення дефектів зростає з 70% до 92%, але зростання замовляється (не лінійне). Це пояснюється тим, що перші 70% покриття охоплюють критичні функції з найбільшою кількістю дефектів, тоді як останні 20% покриття перевіряють менш критичні сценарії[17].

Витрати на утримання vs Продуктивність:

Скорочення витрат на утримання тестів з 40% до 15% звільняє часу, що дозволяє команді розробити більше нових тестів та сконцентруватися на дослідницькому тестуванні. Результат: продуктивність команди зростає на 80-120%.

ROI vs Тривалість впровадження:

ROI зростає кумулятивно: невеликі ROI в першому місяці (50-100%) накопичуються до 300-400% за 6 місяців. Це показує важливість довгострокового хеджування та терпіння під час впровадження.

Виявлення проблемних областей. Структурований аналіз допомагає виявити області, які потребують коригуючих дій:

Проблема 1: Низька успішність тестів (< 90%)

1. Можливі причини: неякісні тестові дані, неправильні пороги самовідновлення, проблеми з інфраструктурою.
2. Рішення: Аудит тестових даних, оптимізація порогів самовідновлення, перевірка стабільності інфраструктури.

Проблема 2: Мале покриття (< 70%)

1. Причини: недостатня генерація тестів, фокус на легких сценаріях, браком часу на розширення.
2. Рішення: Активація AI-генерації для непокритих модулів, розширення пілот-проекту, виділення ресурсів на покриття.

Проблема 3: Високе значення хибних позитивних (> 8%)

1. Причини: неточні перевірки, нестійкі селектори, прив'язаність до специфічних даних.
2. Рішення: Перегляд логіки перевірок, навчання команди кращим практикам, використання більш стійких селекторів.

Проблема 4: Невисокий ROI (< 200%)

1. Причини: висока вартість впровадження, повільне розширення покриття, неврахування всіх вигід.
2. Рішення: Розглянути альтернативні інструменти, прискорити масштабування, переоцінити вигоди (включити непрямі, як швидкість виходу на ринок)[31].

Важливо чітко та переконливо представити результати різним аудиторіям як це показано у табл. 3.5.

Табл. 3.5 - Шляхи представлення результатів різним аудиторіям

Для менеджменту	Для QA-команди	Для розробників
Зосередиться на ROI, скороченні вартості, прискоренню циклу розробки	Показати практичні покращення: скорочення часу на тести, менше утримання	Наголошувати на дефектах, виявлених раніше (економія часу на налагодження)
Порівняти ROI з цільовими показниками та конкурентами	Визначити нові можливості (більше часу для дослідницького тестування)	Показати покращення якості кода (менше дефектів у продакшені)
Показати стабільність результатів (тренди)	Обговорити “болі” та шляхи їх подолання	Демонструвати прискорення циклу розробки

Комплексний аналіз результатів впровадження AI-систем дозволяє організаціям переконатися, що впровадження досягає поставлених цілей, виявити області для подальшої оптимізації та обґрунтувати інвестицію перед заінтересованими сторонами. Використання комбінації кількісних методів (ROI-аналіз, тренд-аналіз) та якісних підходів (опитування команди) забезпечує всебічне розуміння впливу AI-систем на QA-процеси. Побудова інтерактивних дашбордів та регулярне звітування сприяють постійному вдосконаленню та утриманню мотивації команди на протягом впровадження[14].

3.4 Виявлення проблем та шляхів їх подолання

Впровадження AI-систем у QA неминуче супроводжується різними технічними, організаційними та людськими проблемами. Дослідження показують, що 85% команд стикаються з принаймні однією значною проблемою під час впровадження AI-тестування, тоді як 45% зустрічаються з

декількома критичними проблемами одночасно. Ключ до успішного впровадження полягає у своєчасному виявленні цих проблем, проведенні ґрунтовного аналізу та впровадженні обґрунтованих рішень.

Типові проблеми при впровадженні та кількість випадків у команд:

Табл. 3.6 - Проблеми при впровадженні, їх пріоритет та наслідок

Проблема	Пріоритет	Наслідок
Низька якість тестових даних	Критичний	Неточне прогнозування, хибні позитивні
Нестабільність тестів (flaky tests)	Критичний	Відсутність довіри, невеликі ROI
Висока вартість впровадження	Високий	Затримка впровадження, ROI негативний
Інтеграція з CI/CD	Високий	Затримки розгортання, manual запуски
Низька довіра команди	Високий	Опір змінам, низька адоптація
Недостатня навички персоналу	Високий	Неправильна конфігурація, помилки
Технічний борг у тестах	Середній	Складне утримання, замерзлі тести
Безпека та приватність даних	Критичичний	Витоки даних, недотримання compliance

1. Низька якість тестових даних (85% команд)

Це критична проблема, оскільки AI-моделі навчаються на даних, які надаються їм. Якщо дані некоректні, розпорошені або не репрезентативні, система буде генерувати неточні прогнози та хибні позитивні.

Рішення:

- Використання реальних даних з продакшену (з маскуванням PII)
- Регулярна валідація якості даних (щомісячно)

- Створення датасетів з граничними випадками та аномаліями
- Автоматизована перевірка повноти та консистентності даних

2. Нестабільність тестів - Flaky Tests (78% команд)

Тести, які іноді проходять, іноді не проходять без змін коду, підривають довіру до автоматизації. При впровадженні AI це может бути ще гіршим через помилки в генерованих тестах.

Рішення:

- Налаштування механізму самовідновлення на більш стійкі селектори
- Використання паралельного виконання з ретрою для нестійких тестів
- Регулярна ідентифікація та рефакторинг flaky тестів
- Налаштування більш довгих таймаутів та явних ожиданнь замість затримок

3. Інтеграція з CI/CD конвеєром (72% команд)

Інтеграція AI-тестування з існуючими CI/CD системами часто стикаються з проблемами несумісності та затримок розгортання.

Рішення:

- Контейнеризація (Docker) для узгодженості середовищ
- Infrastructure as Code (IaC) для reproducible конфіг
- Оптимізація часу виконання тестів через паралелізацію
- Встановлення чітких порогів для блокування/попередження розгортання

4. Висока вартість впровадження (58% команд)

Вартість AI-інструментів, навчання персоналу та інфраструктури часто перевищує очікування.

Рішення:

- Поступове впровадження з першого пілот-проекту
- Розрахунок ROI на базисі пілот-результатів
- Розгляд альтернативних, менш дорогих рішень (open-source + ML)
- Оцінка вигід на 12+ місяців, а не тільки перших кількох місяців

5. Опір команди до впровадження (65% команд)

Люди природно опираються змінам. QA-команда може бояти, що AI замінить їхні робочі місця.

Рішення:

- Демонстрація успіхів на малих пілот-проектах
- Чітке спілкування про те, що AI доповнює, а не замінює людей
- Залучення команди у вибір інструменту та процесу впровадження
- Прозорі дашборди та звіти, що демонструють користь

6. Безпека та приватність даних (75% команд, критичність 10/10)

Тестові дані часто містять конфіденційну інформацію. Передача їх на хмарні AI-сервіси може створити ризики.

Рішення:

- Маскування особистих даних в тестових наборах
- Шифрування даних при передачі та зберіганні
- Встановлення політик доступу та аудиту
- Вибір on-premise рішень для критичних систем
- DLP (Data Loss Prevention) інструменти для моніторингу

7. Дефіцит навичок персоналу (68% команд)

QA-команди часто не мають необхідних навичок для роботи з AI-інструментами та ML-концепціями.

Рішення:

- Регулярне навчання (щомісячні сесії)
- Сертифікації (ISTQB AI, курси по обраному інструменту)
- Менторинг від досвідчених фахівців
- Письмова документація best practices
- Knowledge sharing сесії у команді

8. Технічний борг у тестовому коді (55% команд)

AI-генеровані тести часто недостатньо модульні, мають дублювання та погану документацію.

Рішення:

- Рефакторинг тестів для модульності та переюзуємості
- Встановлення стандартів для AI-генерованих тестів
- Регулярний код review та аудити

- Документування логіки і призначення кожного тесту[20]

3.4.1 Процес виявлення та розв'язання проблем

Упорядкований метод для знаходження та усунення негараздів містить 5 етапів:

Етап 1: Знахідка (Detection) — того ж дня

1. Спостереження панелей KPI виявляє відхилення
2. Сповіщення системи CI/CD інформують про збої
3. Команда доповідає про труднощі в перевірці

Етап 2: З'ясування (Diagnosis) — 1-2 дні

1. Початкова оцінка наслідків негаразду
2. Збирання відомостей (журнали, логи тестів, показники)
3. Визначення важливості та черговості

Етап 3: Розбір першопричини (Root Cause Analysis) — 2-5 днів

1. Глибоке дослідження джерел проблеми
2. Ізолювання несправного елемента
3. Оцінка відбиття на інших частинах системи

Етап 4: Запровадження вирішення (Solution Implementation) — 3-10 днів

1. Створення та випробування вирішення
2. Впровадження з невпинним наглядом
3. Фіксація змін

Етап 5: Перевірка (Verification) — 2-5 днів

1. Підтвердження, що недолік усунуто
2. Спостереження показників для виявлення повторень
3. Оновлення матеріалів та процедур[16]

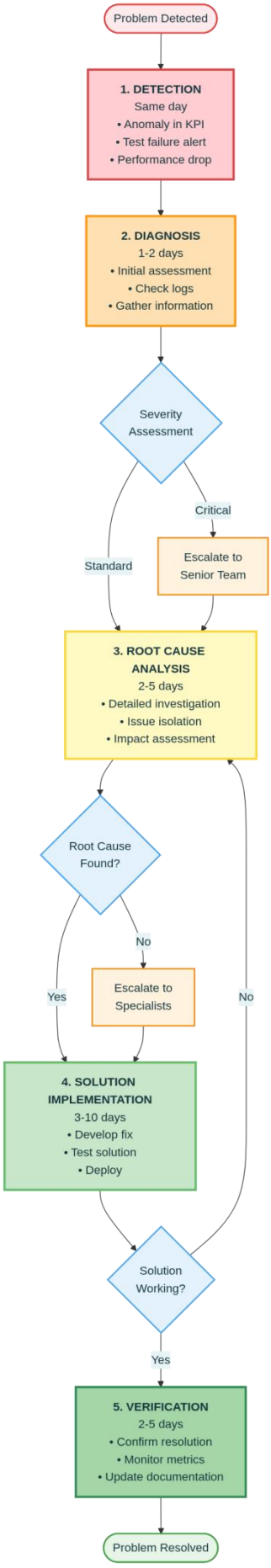


Рис. 3.5 - Алгоритм розв'язання проблем

Критичні проблеми та екстрені рішення

При виявленні критичних проблем необхідні миттєві дії. Рішення проблем запропоновано в табл. 3.7.

Табл. 3.7 - Екстрене рішення критичних проблем

Проблема	Симптом	Екстрене рішення	Час роботи
Системний відмова	Тести не запускаються	Rollback до попередньої версії, restart сервісів	< 30 хвилин
Масивні false positives	> 20% false positives	Тимчасово деактивувати самовідновлення, вручну перевірити	< 2 годин
Витік конфіденційних даних	Дані в логах/звітах	Негайне переривання систем, оповіщення security team	< 10 хвилин
CI/CD блокування	Тести блокують всі розгортання	Дозволити ручне обходження для критичних релізів	< 1 години
Втрата доступу до даних	Тести не можуть читати дані	Перевірити credential, відновити бекап	< 30 хвилин

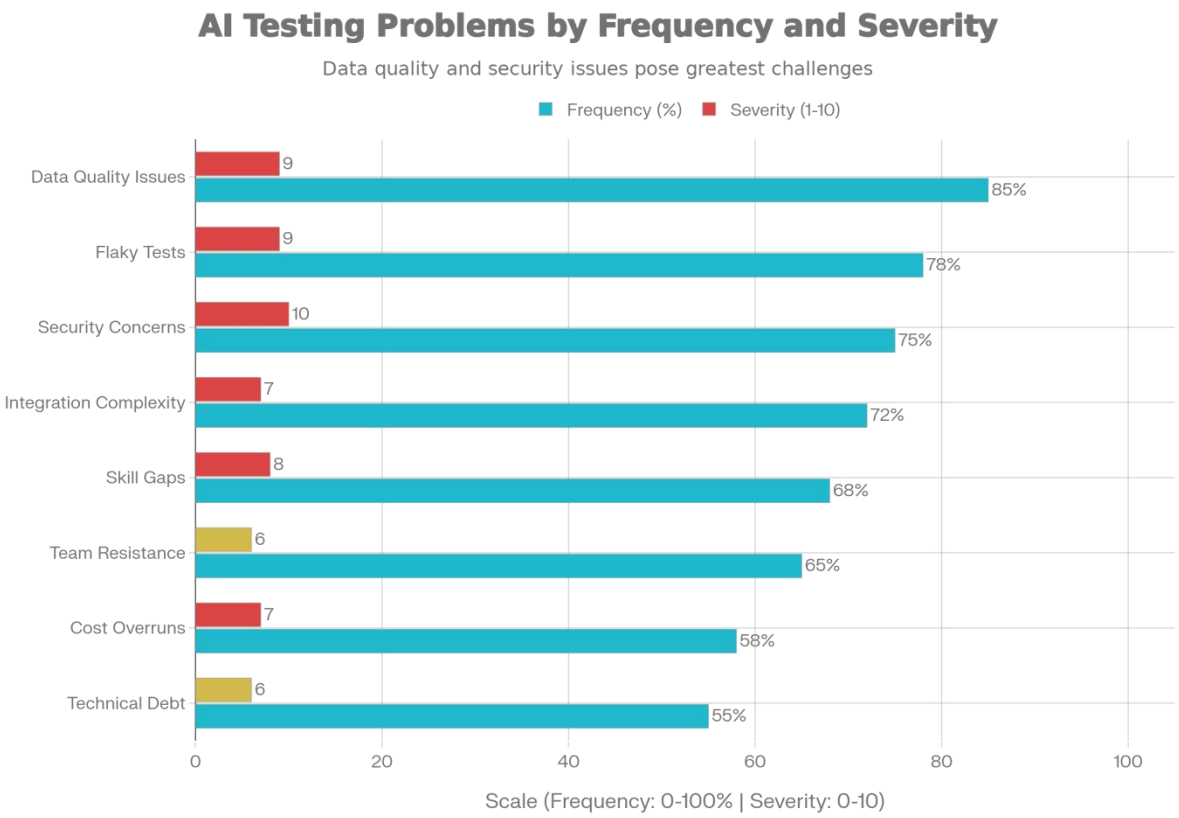


Рис. 3.6 - Частота та критичність проблем

Більш ефективно запобігти проблемам, ніж їх розв'язувати:

1. Регулярний моніторинг (щотижневий)

- Відслідкування KPI трендів
- Перевірка логів на аномалії
- Опитування команди про болі

2. Тестування системи (щомісячний)

- Нагрузкове тестування
- Тестування збоїв та відновлення
- Перевірка безпеки даних

3. Code reviews та аудити (щомісячний)

- Перегляд автоматично генерованих тестів
- Аудит якості кода та стилю
- Перевірка дотримання best practices

4. Навчання та розвиток (щомісячний)

- Knowledge sharing сесії
- Огляд нових функцій AI-інструменту
- Обговорення найбільш частих проблем

5. Планування на майбутнього (щокварталу)

- Огляд дорожної карти впровадження
- Планове оновлення інструменту
- Оцінка нових можливостей
- Висновки до підрозділу

Успішне впровадження AI-систем у QA вимагає активного управління проблемами та ризиками. Висока частота проблем (85% команд) демонструє, що це нормальна частина впровадження, а не ознака невдачі. Ключ до успіху полягає у своєчасному виявленні проблем, ґрунтовному аналізі та швидкому впровадженні рішень. Структурований процес розв'язання проблем (від виявлення до верифікації за 8-22 дні) дозволяє організаціям підтримувати імпорт впровадження навіть при стиканні з перешкодами. Активні превентивні заходи (регулярний моніторинг, тестування, навчання) значно скорочують кількість та серйозність проблем, що виникають у подальшому.

ВИСНОВКИ

Вивчення теоретичних засад та практичних аспектів упровадження технологій штучного інтелекту у процеси забезпечення ґатунку програмного забезпечення засвідчило, що інтеграція ШІ у QA є логічним етапом еволюції сучасної індустрії розробки програмного забезпечення, а не короткочасним трендом. Зростаюча складність програмних систем, перехід до гнучких методик розробки та високі потреби до оперативності випуску роблять традиційні підходи до тестування недостатньо дієвими та фінансово невиправданими у тривалій перспективі.

У першому розділі обґрунтовано важливість теми та здійснено аналіз поточного стану використання штучного інтелекту в аспектах забезпечення ґатунку. Доведено, що світовий ринок рішень для тестування ШІ демонструє стійке зростання, а значна частина провідних фірм уже впроваджують інтелектуальні засоби у свої процеси забезпечення ґатунку. З’ясовано, що ШІ дає змогу зменшити тривалість виконання тестів, підвищити коректність виявлення недоліків, розширити охоплення тестів та суттєво урізати видатки на утримання тестової інфраструктури.

У другому розділі структуровано ключові концепції та технології штучного інтелекту, які мають пряме відношення до забезпечення ґатунку, охоплюючи машинне навчання, нейронні мережі, глибоке навчання, обробку природної мови та комп'ютерний зір.

Проаналізовано класичні алгоритми машинного навчання (Decision Trees, Random Forest, SVM, KNN, Naive Bayes, XGBoost) та продемонстровано їхню придатність для класифікації дефектів, передбачення ризиків та визначення пріоритетності тестів. Розглянуто глибокі нейронні мережі (CNN, RNN, LSTM) та трансформаторні моделі, які забезпечують вищу точність у тестуванні візуального інтерфейсу, аналізі логів та роботі з текстовими вимогами. Особлива увага приділена гібридним підходам, які комбінують класичне МН та глибоке навчання і демонструють оптимальний компроміс між точністю передбачення дефектів та обчислювальною продуктивністю.

Також у другому розділі розглянуто позитивні сторони та труднощі впровадження ШІ у практики забезпечення ґатунку на підставі порівняльних таблиць та аналітичних ґрафіків. Показано, що застосування моделей ШІ покращує точність виявлення дефектів порівняно з традиційними методиками, знижує кількість хибних спрацювань та дає змогу перейти від реактивного до превентивного керування якістю. Водночас аналізуються загрози, пов'язані з витратами на впровадження та складністю інтеграції в поточний ланцюжок CI/CD.

Третій розділ зосереджено на практичних аспектах впровадження рішень ШІ у процеси забезпечення ґатунку програмного забезпечення. На основі аналізу методів і засобів тестування на базі ШІ обґрунтовуються стратегії їхнього вибору залежно від типу системи, наявних даних та вимог до продуктивності. Розглядаються загальні засади інтеграції модулів ШІ в апаратуру забезпечення ґатунку, взаємодія з конвеєрами CI/CD та особливості калібрування моделей для конкретних завдань, як-от передбачення дефектів, автоматичне створення тестових сценаріїв, виявлення невідповідностей у логах та управління нестабільними тестами. Висвітлено підходи до аналізу та тлумачення результатів упровадження, включно з оцінкою показників точності, повноти, тривалості виконання тестів та віддачі від інвестицій (ROI).

Висновок результатів демонструє, що використання технологій штучного інтелекту у забезпеченні ґатунку дає змогу здійснити перехід від переважно ручного, обтяжливого тестування до інтелектуалізованих, гнучких і придатних до масштабування процесів контролю якості. Автоматизація аналізу великих обсягів відомостей, створення тестів на основі вимог, передбачення дефектів та розпізнавання аномалій робить процес забезпечення ґатунку більш гнучким, швидким та фінансово вигідним. Водночас, для досягнення очікуваного ефекту, установам потрібно інвестувати не лише у засоби, але й у підготовку даних, розвиток експертизи персоналу, пристосування процесів та формування культури довіри до рішень, базованих на штучному інтелекті.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. AI evaluation guidelines. GitLab. URL: https://www.uni-hildesheim.de/gitlab/help/development/ai_features/ai_evaluation_guidelines.md.

2. AI model evaluation guide | label studio. Label Studio. URL: https://labelstud.io/learningcenter/the-complete-guide-to-evaluations-in-ai/.

3. Andrades G. Supercharge automation with AI testing frameworks. ACCELQ. URL: https://www.accelq.com/blog/ai-testing-frameworks/.

4. Anglen J. Understanding artificial intelligence: key concepts and applications. Rapid Innovation | Next-Gen AI for Future-Ready Enterprises. URL: https://www.rapidinnovation.io/post/artificial-intelligence-the-key-concepts-of-ai.

5. Automated testing tools comparison: a data-driven guide for 2024. Home | Context Engineering: MCP for Cursor, Claude Code, VS Code. URL: https://contextengineering.ai/blog/automated-testing-tools-comparison/.

6. Bebe S. Machine learning algorithms for test automation. Sogeti Labs. URL: https://labs.sogeti.com/machine-learning-algorithms-for-test-automation/.

7. Bondar A. Top 8 AI based test automation tools in 2024. QA and Software Testing Company - Luxe Quality. URL: https://luxequality.com/blog/ai-based-test-automation-tools/.

8. CI/CD: що це, як пов’язано з DevOps, переваги, найкращі практики | NIX. NIX. URL: https://www.nixsolutions.com/ua/blog/for-developer/ci-cd-shho-cze-yak-povyazano-z-devops-perevagy-najkrashhi-praktyky/.

9. Cosstick J. Deep learning & neural networks: a beginner's guide. Tech Life Future. URL: https://www.techlifefuture.com/deep-learning-neural-networks/.

10. GDPR та захист персональних даних: чек-лист для українського бізнесу | Brandr. Brandr. URL: https://brandr.legal/gdpr-ta-zakhyst-personal′nykh-danykh-chek-lyst-dlya-ukrayins′koho-biznesu/.

11. Hahanov. Практичне машинне навчання лк 4 - нейронні мережі, 2025. YouTube. URL: https://www.youtube.com/watch?v=rZzpU1kLnEk.

12. How AI identifies and prioritizes defects in software testing. Best QA and Testing Blogs. URL: https://blog.qasource.com/software-development-and-qa-tips/how-ai-helps-for-defect-detection-and-prioritization.

13. How to select the right test automation framework for your project in 2025. AI Testing Tools - Find The Best AI-Powered Testing Tools. URL: https://www.testingtools.ai/blog/how-to-select-the-right-test-automation-framework-for-your-project-in-2025/.

14. Jha B. Maximizing test automation ROI: strategies, metrics, and tools. ACCELQ. URL: https://www.accelq.com/blog/test-automation-roi/.

15. Karimov A. AI and ML in QA: real tools, real impact. Bugsee. URL: https://bugsee.com/blog/ai-and-ml-in-qa/.

16. Karl T. Understanding AI: key concepts and technologies explained. New Horizons. URL: https://www.newhorizons.com/resources/blog/understanding-ai-technology.

17. Klyuchnik, O. Degtiarov et al., "Identification of parameters of measuring modules based on pyroelectric materials," Ukrainian Metrological Journal, no. 2, pp. 16–23, 2025, doi: https://doi.org/10.24027/2306-7039.2.2025.333812 (WoS).

18. Kumar R. Test automation tool selection guide. Intelligent AI Testing Tool For Enterprises | Virtuoso QA. URL: https://www.virtuosoqa.com/post/test-automation-tool-selection-guide.

19. Langenhoven (Hon Cau) V. The impact of AI on software quality assurance and testing: advantages, disadvantages, and available tools. linkedin.com. URL: https://www.linkedin.com/pulse/impact-ai-software-quality-assurance-testing-tools-dr-valdon-zgoyf.

20. Mostögl T. Automated testing - strategy and ROI analysis. Intelligent AI Testing Tool For Enterprises | Virtuoso QA. URL: https://www.virtuosoqa.com/post/automated-testing-strategy-roi-enterprises.

21. Muhammadi E. AI-Driven software testing: benefits, challenges, and future trends | emin muhammadi. eminmuhammadi.com.

URL: https://eminmuhammadi.com/articles/ai-driven-software-testing-benefits-challenges-and-future-trends.

22. News P. Машинне навчання у прогнозному тестуванні для середовищ DevOps - ProIT. ProIT: медіа для профі в IT. URL: https://proit.ua/mashinnie-navchannia-v-proghnoznomu-tiestuvanni-dlia-sieriedovishch-devops/.

23. O. Degtiarov et al., "Development of a method and a measuring instrument in the area of studying the parameters of the low-frequency magnetic field," Ukrainian Metrological Journal, no. 1, pp. 17–25, 2025, doi: 10.24027/2306-7039.1.2025.325873 (WoS).

24. Onyshchenko R., Kotenko N., Zhyrova T. The role and effectiveness of artificial intelligence tools in software testing. Information technology and society. 2024. No. 2 (13). P. 66–70. URL: https://doi.org/10.32689/maup.it.2024.2.10.

25. Pangan K., Nicholas S. The ultimate guide to an analytics dashboard template for AI performance in support. eesel AI.
URL: https://www.eesel.ai/blog/analytics-dashboard-template-for-ai-performance-in-support.

26. Popovska M. 10 best tools for automated testing in 2025. TestDevLab Blog. URL: https://www.testdevlab.com/blog/top-10-test-automation-tools-2025.

27. Reed J. The ultimate guide to choosing AI testing tools for your team. DEV Community. URL: https://dev.to/testjace/the-ultimate-guide-to-choosing-ai-testing-tools-for-your-team-2773.

28. Reznik D. How to calculate ROI for AI in testing? A complete guide - OwlityAI. OwlityAI. URL: https://owlity.ai/articles/how-to-optimize-qa-costs-roi-of-autonomous-testing.

29. Rose-Collins F. Ranktracker:The all-in-one platform for effective SEO. URL: https://www.ranktracker.com/blog/revolutionizing-quality-assurance-role-of-ai-in-software-testing/.

30. Simonova M. How artificial intelligence is impacting quality assurance roles. Forbes.

URL: https://www.forbes.com/councils/forbestechcouncil/2023/08/21/how-artificial-intelligence-is-impacting-quality-assurance-roles/.

31. Stryker C., Kavlakoglu E. What is artificial intelligence (AI)? | IBM. IBM. URL: https://www.ibm.com/think/topics/artificial-intelligence.

32. Test automation ROI - how to calculate it?. Global App Testing | Improve User Experience with Global Testing Solutions | Global App Testing. URL: https://www.globalapptesting.com/blog/test-automation-roi.

33. Test automation ROI: a step-by-step guide in 2025 - qasource. Best QA and Testing Blogs. URL: https://blog.qasource.com/resources/improve-your-test-automation-roi.

34. Tsymbal T. AI evaluation metrics 2025: tested by conversation experts. Master of Code Global. URL: https://masterofcode.com/blog/ai-agent-evaluation.

35. Types of machine learning: supervised, unsupervised and more | digitalocean. DigitalOcean | The Unified Agentic Cloud. URL: https://www.digitalocean.com/resources/articles/types-of-machine-learning.

36. Ukkuru G. Test automation framework: 30 top criteria for evaluation. TestMetry. URL: https://testmetry.com/test-automation-framework/.

37. Watson M. Software quality assurance framework: our best practices from 500+ projects. Full Scale. URL: https://fullscale.io/blog/software-quality-assurance-framework/.

38. Weingartz R., Suleymanov N. Calculating test automation ROI: best practices and examples. aqua cloud - best software for testing. URL: https://aqua-cloud.io/test-automation-roi/.

39. Williams B. 5 metrics that help prove the ROI of QA automation - insight7 - call analytics & AI coaching for customer teams. Insight7 - Call Analytics & AI Coaching for Customer Teams. URL: https://insight7.io/5-metrics-that-help-prove-the-roi-of-qa-automation/.

40. Williams B. How to build a benchmarking dashboard using AI evaluation scores - insight7 - call analytics & AI coaching for customer teams. Insight7 - Call

Analytics & AI Coaching for Customer Teams. URL: https://insight7.io/how-to-build-a-benchmarking-dashboard-using-ai-evaluation-scores/.

41. Автоматизація від коміту до продакшену. FoxmindEd. URL: https://foxminded.ua/shho-take-ci-cd-prostymy-slovamy-devops-dlya-vsih/.

42. Білоус Д. Ю., Дегтярьов О. В. Автоматизоване тестування вебзастосунків на основі підходу Behavior-Driven Development // Розвиток сучасної науки: актуальні питання теорії та практики : матеріали IX Всеукр. студент. наук. конф. (Київ, 21 листоп. 2025 р.). – Київ : Ін-т науково-технічної інтеграції та співпраці, 2025. – С. 479–480. – DOI: 10.62732/liga-ukr-21.11.2025. – ISBN 978-617-8582-04-3.

43. Використання штучного інтелекту для вдосконалення тестування софту. IT-компания полного цикла разработки программных продуктов WEZOM - Киев, Украина. URL: https://wezom.com.ua/ua/blog/yak-mi-vikoristovujemo-shi-dlya-revolyutsiynih-zmin-u-testuvanni-softu.

44. Дегтярьов О. В. та ін. Оцінка невизначеностей вимірювань просторових гармонік зовнішнього магнітного поля // Метрологія та прилади. – 2025. – № 1. – С. 4. https://doi.org/10.30837/2663-9564.2025.1.06

45. Інструменти автоматизації QA 2025: AI, LLM, генерація автотестів | Wezom. IT-компанія повного циклу розробки программного забезпечення WEZOM - Київ, Україна. URL: https://wezom.com.ua/ua/blog/naykraschi-instrumenti-dlya-avtomatizatsiyi-qa-u-2025-rotsi.

46. Козлов Ю. В., Дегтярьов О. В. та ін. Метод оцінювання рівня навченості суб’єкта навчання // Метрологія та прилади. – 2024. – № 1. – С. 59–64. – DOI: https://doi.org/10.30837/2663-9564.2024.1.11

47. Методологія CI/CD: автоматизація, тестування і швидкі релізи. IT Education Center Blog. URL: https://itedu.center/ua/blog/review/what-is-ci-cd/.

48. Мещерякова К. Що таке нейромережа? | нейронні мережі: приклади. High Bar Journal. URL: https://journal.gen.tech/post/sho-take-neiromerezhi.

49. Нейронні мережі | machine learning | google for developers. Google for Developers. URL: https://developers.google.com/machine-learning/crash-course/neural-networks?hl=uk.

50. Учасники проектів Вікімедіа. Штучна нейронна мережа – Вікіпедія. Вікіпедія.
URL: https://uk.wikipedia.org/wiki/Штучна_нейронна_мережа.

51. Хорошайло, Дегтярьов О. В. та ін. Дослідження можливостей використання колориметрії в медицині // Метрологія та прилади. – 2024. – № 2. – С. 32–38. – DOI: https://doi.org/10.30837/2663-9564.2024.2.07

52. ШІ в тестуванні програмного забезпечення. Visure Solutions.
URL: https://visuresolutions.com/uk/alm-guide/штучний-інтелект-у-тестуванні-програмного-забезпечення/.

53. ШІ у QA: швидша й розумніша автоматизація | Agiliway. Agiliway | Custom Software Development and AI Solutions for Global Businesses.
URL: https://www.agiliway.com/uk-ua/avtomatizacziya-qa-na-osnovi-shi-yak-pidvishhiti-efektivnist-testuvannya-programnogo-zabezpechennya/.

54. Шпаргалка з тестування - Q & A. Q & A - Навчальний ресурс з тестування програмного забезпечення.
URL: https://qalearning.com.ua/theory/about_qa/shpargalka-z-testuvannya/.

55. Що таке QA тестування та яка його роль процесі розробки. Sigma Software University. URL: https://university.sigma.software/what-is-qa-testing/.