МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПОЛІСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ, ОБЛІКУ ТА ФІНАНСІВ

Кафедра комп'ютерних технологій і моделювання систем

Кваліфікаційна робота
на правах рукопису

Остапенко Микола Олександрович

УДК 004.415.5:004.89

КВАЛІФІКАЦІЙНА РОБОТА

Розроблення алгоритму автоматизованого тестування програмного забезпечення на основі AI

122 «Комп’ютерні науки»

Подається на здобуття освітнього ступеня магістр

кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Остапенко Микола Олександрович

Керівник роботи
Ковальчук Майя Олегівна
к. п. н., доцент кафедри комп'ютерних технологій та моделювання систем

Житомир – 2025

Висновок кафедри комп'ютерних технологій і моделювання систем:
за результатами попереднього захисту:_____

Протокол засідання кафедри комп'ютерних технологій і моделювання систем
№ ____ від «_____» _____ 20__ р.

Завідувач кафедри комп'ютерних технологій і моделювання систем
к.п.н., доцент _____ М. О. Ковальчук
(науковий ступінь, вчене звання) (підпис) (прізвище, ім'я, по батькові)
«_____» _____ 20___ р.

Результати захисту кваліфікаційної роботи

Здобувач вищої освіти Остапенко Микола Олександрович захистив (ла) кваліфікаційну роботу з оцінкою:
сума балів за 100-бальною шкалою _____
за шкалою ECTS _____
за національною шкалою _____

Секретар ЕК
лаборант кафедри _____ В. В. Корольчук
(науковий ступінь, вчене звання) (підпис) (прізвище, ім'я, по батькові)

АНОТАЦІЯ

Остапенко М. О. Розробка алгоритму автоматизованого тестування програмного забезпечення на основі AI. – Кваліфікаційна робота на правах рукопису.
Кваліфікаційна робота на здобуття освітнього ступеня магістр за спеціальністю 122 – Комп'ютерні науки. – Поліський національний університет, Житомир, 2025.
Обсяг кваліфікаційної роботи: 35 сторінок (14 – рисунків, 3 – таблиці, 3 – додатки, 45 – використаних джерел)
Ключові слова: *автоматизоване тестування, візуальне тестування, інструменти автоматизації тестування, тест-дизайн, тестові техніки, штучний інтелект*
Робота присвячена дослідженню сучасних підходів та технологій автоматизованого тестування програмного забезпечення із застосуванням штучного інтелекту. Розглядаються теоретичні основи тестування, зокрема поняття та історія розвитку тестування програмного забезпечення, аналіз особливостей автоматизованих методів, а також різноманітні техніки тест-дизайну, що дозволяють підвищити ефективність процесу тестування. Значну увагу приділено моделям життєвого циклу тестування та класифікації видів тестування, включаючи функціональне і нефункціональне тестування, а також застосуванню методів штучного інтелекту у цій сфері.
Проведено аналіз сучасних технологій та інструментів автоматизованого тестування із застосуванням AI. Тут розглядаються можливості використання штучного інтелекту для автоматизації тестових сценаріїв, а також аналізуються існуючі інструменти, їх особливості та переваги. Вивчаються перспективи розвитку штучного інтелекту у сфері автоматизованого тестування, зокрема у контексті підвищення швидкості та точності тестових процесів.
Описується процес проектування та реалізації алгоритму автоматизованого тестування на основі AI. Детально розглядається технічний стек та архітектура системи, а також інтеграція популярних інструментів у тестовий процес. Наводяться практичні рекомендації та результати досліджень щодо застосування розробленого алгоритму.

SUMMARY

Ostapenko M. O. Development of an AI-based algorithm for automated software testing.
Qualification work for the degree of Master in specialty 122 – Computer Science. – Polesie National University, Zhytomyr, 2025.
Qualification work volume: 35 pages (14 – figures, 3 – tables, 3 – appendices, 45 - sources)
Key words: automated testing, visual testing, test automation tools, test design, test techniques, AI
This work is devoted to the study of modern approaches and technologies of automated software testing using artificial intelligence. The theoretical foundations of testing are considered, in particular, the concept and history of the development of software testing, analysis of the features of automated methods, as well as various test design techniques that allow increasing the efficiency of the testing process. Considerable attention is paid to testing life cycle models and classification of testing types, including functional and non-functional testing, as well as the application of artificial intelligence methods in this area.
An analysis of modern technologies and tools for automated testing using AI is carried out. The possibilities of using artificial intelligence to automate test scenarios are considered here, as well as existing tools, their features and advantages are analyzed. The prospects for the development of artificial intelligence in the field of automated testing are studied, in particular in the context of increasing the speed and accuracy of test processes.
The process of designing and implementing an automated testing algorithm based on AI is described. The technical stack and architecture of the system are considered in detail, as well as the

integration of popular tools into the test process. Practical recommendations and research results on the application of the developed algorithm are provided.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI – штучний інтелект
API – інтерфейс програмування додатків
BDD – поведінковий підхід (Behavior-Driven Development)
Cucumber – інструмент для підтримки BDD та автоматизації сценаріїв у форматі Gherkin
Explyt – платформа для автоматичного генерування тестових сценаріїв на базі штучного інтелекту
POM – шаблон Page Object Model для організації автоматизованих тестів інтерфейсу
Selenide – фреймворк для стабільного автоматичного тестування веб-інтерфейсів
UI – користувацький інтерфейс
Page Object – шаблон для організації автоматизованих тестів, що описує сторінки або компоненти UI
TestCase – тестовий сценарій або випадок
TestSuite – набір тестових сценаріїв або тестів, що виконуються разом
Automation Framework – структура та набір інструментів для автоматизації тестування
Visual Testing – візуальне тестування для перевірки зовнішнього вигляду UI
Test Plan – план тестування, документ з описом стратегії, обсягів, ресурсів і термінів

Вступ

У сучасних умовах цифрової трансформації надійність програмного забезпечення (ПЗ) стала критичним фактором стабільності економічних та соціальних процесів. Високі вимоги до функціональності та безпеки систем зумовлюють необхідність застосування сучасних методів тестування ще на ранніх етапах розробки. Проте, згідно зі світовими звітами World Quality Report, хоча витрати на забезпечення якості (QA) сьогодні становлять до 25-30% загального бюджету розробки, традиційні підходи до автоматизації вже не встигають за темпами Continuous Delivery (CD) та зростаючою складністю мікросервісних архітектур [1].

Актуальність дослідження зумовлена кількома чинниками. По-перше, це проблема низької адаптивності та «крихкості» тестів. За аналітичними даними Capgemini, лише 15-20% організацій досягли високого рівня зрілості в [2]. Основною перешкодою є використання класичних інструментів (на кшталт Selenium чи Selenide), які базуються на жорсткій прив’язці до DOM-структури сторінки. Будь-яка візуальна зміна інтерфейсу (GUI) спричиняє «падіння» таких тестів, що призводить до значних часових витрат на їх рефакторинг. Застосування методів штучного інтелекту (AI), зокрема машинного зору та інтелектуального аналізу об’єктів (Object Recognition), дозволяє створити механізми Self-healing (самовідновлення), що мінімізує втручання людини [3].

По-друге, це необхідність обробки великих масивів даних. Сучасне тестування генерує великі обсяги логів та результатів. Статистичні дані свідчать, що впровадження AI-алгоритмів для аналізу результатів дозволяє скоротити час на пошук першопричини помилки (Root Cause Analysis) на 50-70% [4]. Це критично важливо для великих проектів, де швидкість виправлення дефектів безпосередньо впливає на конкурентоспроможність продукту.

По-третє, це економічна доцільність та дефіцит ресурсів. В умовах зростаючої складності ПЗ та дефіциту кваліфікованих інженерів, розробка інтелектуальних

систем, здатних автоматично генерувати тестові сценарії на основі аналізу вимог, стає стратегічним завданням для IT-індустрії [5]. Це дозволяє не лише скоротити час тестування, а й підвищити його глибину та точність.

Незважаючи на активний інтерес до цієї сфери, існує гостра наукова потреба у створенні універсальних алгоритмів, які забезпечують високу точність виявлення дефектів при низькому рівні хибнопозитивних спрацювань (False Positives). Більшість існуючих рішень мають обмеження у здатності до самонавчання в динамічних системах.

Таким чином, актуальність теми дослідження обумовлена необхідністю розв'язання суперечності між зростаючими темпами розробки ПЗ та обмеженими можливостями традиційних методів автоматизації. Напрям роботи, пов'язаний із розробленням алгоритму на основі штучного інтелекту, є надзвичайно важливим як для науки, так і для практики.*Об'єктом* дослідження є процес автоматизованого тестування програмного забезпечення, зокрема, алгоритми, методи та технології його реалізації з використанням штучного інтелекту, здатні самостійно аналізувати та оцінювати якість програмних продуктів.

Мета роботи полягає у розробленні алгоритму автоматизованого тестування програмного забезпечення на базі штучного інтелекту та дослідженні його поведінки з використанням програмних засобів. Це дасть можливість не тільки краще зрозуміти внутрішні процеси системи, але й підвищити її ефективність та надійність.

Завдання роботи включає в себе аналіз сучасного стану предметної області, зокрема існуючих методів автоматизованого тестування та застосування AI у цій сфері. Вивчення існуючих алгоритмів та технологій тестування програмного забезпечення. Розробити алгоритм автоматизованого тестування на основі штучного інтелекту та експертних систем. Реалізувати прототип системи автоматизованого тестування з використанням сучасних інструментів і мов програмування. Визначити напрями подальшого удосконалення системи і можливості її інтеграції у реальні процеси тестування.

Методи дослідження включають аналіз літературних джерел, системний підхід до розробки алгоритмів, експериментальні дослідження з використанням програмних засобів машинного навчання та тестування.

Предметом дослідження є властивості і механізми застосування методів машинного навчання і експертних систем для автоматичного аналізу тестових сценаріїв, виявлення дефектів і генерації тестових випадків у процесі тестування програмних систем.

Новизна полягає у створенні алгоритму автоматизованого тестування, що інтегрує сучасні методи штучного інтелекту для підвищення адаптивності та точності виявлення дефектів. Це дозволить зробити внесок у теоретичне обґрунтування і практичне застосування інтелектуальних систем тестування.

Результати дослідження мають важливе *теоретичне значення* для розвитку використання штучного інтелекту в автоматизації. Вони сприятимуть формуванню нових підходів до створення інтелектуальних систем тестування, розширенню їх можливостей та підвищенню ефективності процесів забезпечення якості програмного забезпечення.

Практичне значення полягає у можливості застосування розробленого алгоритму у промислових умовах для автоматизації процесів тестування у великих та малих IT-компаніях, підвищенні швидкості випуску продуктів, зменшенні людських помилок і витрат. Прототип системи може бути інтегрований у сучасні CI/CD-конвеєри, що дозволить автоматизувати тестування на всіх етапах життєвого циклу розробки.

Розділ 1. Теоретичні основи автоматизованого тестування програмного забезпечення

1.1 Поняття тестування програмного забезпечення і його розвиток

У сучасному світі розробки програмного забезпечення якість продукту є одним із головних чинників його успіху та конкурентоспроможності. Вірність функціонування, стабільність, безпека та зручність у використанні - це ті характеристики, які визначають довіру користувачів і ефективність роботи системи. Щоб гарантувати відповідність програмного забезпечення цим вимогам, необхідно застосовувати системний та комплексний підхід до його перевірки.

Тестування програмного забезпечення - це невід'ємна частина процесу розробки, що спрямована на виявлення дефектів, недоліків і невідповідностей у програмі з метою їх усунення до випуску продукту на ринок. Воно дозволяє забезпечити якість, зменшити витрати на виправлення помилок на пізніх етапах і підвищити довіру кінцевих користувачів [6].

Залежно від цілей, обсягу і стадії розробки, тестування може здійснюватися різними методами та на різних рівнях - від ручного виконання тестів користувачами або тестувальниками до автоматизованих систем, що дозволяють швидко і ефективно виконувати повторювані сценарії. Важливим аспектом є правильний вибір підходів і інструментів для тестування, що забезпечить максимальну ефективність і якість процесу [6].

У цьому контексті особливе значення має автоматизоване тестування, яке дозволяє підвищити продуктивність і точність перевірок, зменшити людський фактор і забезпечити сталість результатів. Саме про поняття, особливості та застосування автоматизованого тестування й піде мова у наступному розділі.

Протягом десятиліть розвитку розробки ПЗ до питань тестування та забезпечення якості підходили дуже по-різному. Можна виділити кілька основних «епох тестування».

У 50-60-х роках минулого століття процес тестування був гранично формалізований, відокремлений від процесу безпосередньої розробки ПЗ та «математизований». Фактично тестування було швидше налагодження програм. Існувала концепція так званого «вичерпного тестування», тобто перевірки всіх

можливих шляхів виконання коду з усіма можливими вхідними даними. Проте дуже швидко було з'ясовано, що вичерпне тестування неможливе, так як кількість можливих шляхів та вхідних даних дуже велике, а також за такого підходу складно знайти проблеми в документації [6].

У 70-х роках фактично народилися дві фундаментальні ідеї тестування: тестування спочатку розглядалося як доказ працездатності програми в деяких заданих умовах і мало назву позитивне тестування, а потім навпаки: як процес доказу непрацездатності програми у деяких заданих умовах, називалось негативним тестуванням. Це внутрішнє протиріччя не тільки не зникло з часом, але й у наші дні багатьма авторами зовсім справедливо відзначається як дві взаємодоповнюючі цілі тестування [6].

У 80-х роках відбулася ключова зміна місця тестування у розробці ПЗ, замість однієї з фінальних стадій створення проекту тестування стало застосовуватися протягом усього циклу розробки, що дозволило в багатьох випадках не тільки швидко виявляти та усувати проблеми, але навіть передбачати та запобігати їх появі. У цей же період відзначено бурхливий розвиток та формалізація методологій тестування та поява перших елементарних спроб автоматизувати тестування [6].

У 90-х роках відбувся перехід тестування до всеосяжного процесу, який називається «забезпечення якості» і охоплює весь цикл розробки ПЗ та зачіпає процеси планування, проектування, створення та виконання тест-кейсів, підтримку наявних тест-кейсів та тестових оточень. Тестування вийшло на якісно новий рівень, який природно призвів до подальшого розвитку методологій, появи досить потужних інструментів управління процесом тестування та інструментальних засобів автоматизації тестування, вже цілком схожих на своїх нинішніх нащадків [6].

У нульові роки нинішнього століття розвиток тестування продовжувався в контексті пошуку нових і нових шляхів, методологій, технік і підходів до забезпечення якості. Серйозний вплив на розуміння тестування справила поява гнучких методологій розробки та таких підходів, як «розробка під керуванням тестуванням» (TDD). Автоматизація тестування вже сприймалася як звичайна невід'ємна частина більшості проектів, а також стали популярні ідеї про те, що на головне місце тестування слід ставити не відповідність програми вимогам, а її

здатність надати кінцевому користувачеві можливість ефективно вирішувати свої завдання [6].

На сучасному етапі розаитку тестування ПЗ набуло значно більшої системності, автоматизації і технологічної складності. Тести виконуються у хмарних середовищах і на різних платформах одночасно, що зменшує час тестування і дозволяє перевіряти системи у реальних умовах. Інструменти використовують AI для автоматичного створення тестових сценаріїв, аналізу результатів, виявлення прихованих дефектів та оптимізації процесу тестування. Зростає увага до тестування на рівні API, мікросервісів і контейнеризованих систем, що забезпечує швидше і більш гнучке тестування складних архітектур [6].

Загалом, якщо у період минулих років тестування було здебільшого ручною і менш системною діяльністю, то сьогодні - це складний та високотехнологічний процес, що використовує інноваційні підходи, інструменти та технології. Сучасне тестування орієнтоване на швидкість, масштабність і автоматизацію, що дозволяє забезпечувати високий рівень якості програмних продуктів у швидкозмінних умовах сучасної IT-індустрії.

1.2 Аналіз особливостей автоматизованого тестування

Автоматизоване тестування програмного забезпечення є одним із ключових напрямків сучасної інженерії якості та забезпечення надійності програмних систем. Це процес застосування спеціалізованих інструментів і скриптів для виконання тестових сценаріїв без участі людини, що дозволяє підвищити швидкість, точність і повторюваність тестових процедур.

Автоматизоване тестування - це система методів і засобів, що дозволяють автоматично виконувати тестові сценарії з метою виявлення дефектів, перевірки відповідності системи технічним і функціональним вимогам. Від ручного тестування воно відрізняється тим, що більша частина процесу виконується автоматично за допомогою спеціальних інструментів, що імітують дії користувача або внутрішні функції системи [7].

Ключовими елементами автоматизованого тестування є тестові скрипти, тестові дані, тестові середовища та системи моніторингу результатів. Такі системи

здатні виконувати багаторазові повторювані операції, що особливо важливо при регресивному тестуванні та при роботі з великими обсягами даних.

Основні характеристики автоматизованого тестування включають:

- **Висока швидкість виконання**. Автоматизовані системи здатні значно швидше виконувати тестові сценарії порівняно з ручним тестуванням, що дозволяє швидко отримувати результати і вчасно реагувати на знайдені дефекти.
- **Повторюваність і стабільність**. Можливість багаторазового запуску однакових тестів без людських помилок забезпечує стабільність і надійність результатів.
- **Масштабованість**. Легко масштабувати тестові сценарії для перевірки великих та складних систем, у тому числі для тестування різних конфігурацій та середовищ.
- **Автоматична генерація і виконання**. За допомогою скриптів і тестових фреймворків можна автоматично створювати нові сценарії та запускати їх без втручання людини.
- **Об'єктивність і точність**. Відсутність людського фактора зменшує ймовірність пропуску дефектів і підвищує точність результатів [8].

Одним із важливих концептів у сучасному тестуванні є піраміда тестування, яка ілюструє оптимальну структуру ієрархії тестових сценаріїв для досягнення ефективного і економічного процесу тестування [9]. Піраміда передбачає три рівні (Додаток А, рис. А.1):

- **Юніт-тести (Unit tests)** - найнижчий рівень, що охоплює тести окремих модулів або функцій програми. Вони мають бути чисельними і швидкими, їхня кількість має переважати.
- **Інтеграційні тести (Integration tests)** - середній рівень, що перевіряє взаємодію між модулями або компонентами системи.
- **Системні тести (End-to-End tests, E2E)** - верхній рівень, що охоплює сценарії, близькі до реального користувача, але їх кількість має бути мінімальною через високу вартість і довгий час виконання [6].

Ця модель підкреслює важливість максимального автоматизаційного охоплення на нижчих рівнях юніт і інтеграційних тестах, що зменшує навантаження на ручне тестування і забезпечує швидке виявлення дефектів. Водночас, автоматизація тестів на рівні користувацького інтерфейсу є більш затратною і менш стабільною, тому вона застосовується для обмеженого числа тестових сценаріїв [6].

Такий вид тестування найбільш ефективний у випадках, коли потрібно регулярно виконувати однакові або схожі тестові сценарії, наприклад, у процесах регресивного тестування, тестування великих систем, або в рамках безперервної інтеграції (CI/CD). Також він корисний для тестування повторюваних функцій, що мають багато входів і сценаріїв, а також у випадках, коли ручне тестування є надто затратним або неможливим.

Водночас автоматизація має і свої обмеження. Вона вимагає розробки та підтримки тестових скриптів, що може бути трудомістким і вимагає спеціальних знань. Не всі типи тестів підходять для автоматизації - наприклад, тестування інтуїтивних та естетичних аспектів інтерфейсу або користувацького досвіду.

1.3 Процеси тестування і моделі розробки програмного забезпечення

Модель розробки ПЗ - це структурований підхід, що визначає послідовність етапів створення програмного продукту, а також методи і практики їх виконання. Вибір моделі розробки ПЗ серйозно впливає на процес тестування, визначаючи вибір стратегії, розклад, необхідні ресурси. Моделей розробки ПЗ багато, але в загальному випадку класичними можна вважати водоспадну, v-подібну, ітераційну, спіральну та гнучку [10].

Водоспадна модель (Waterfall model) зараз представляє скоріше історичний інтерес, так як у сучасних проектах практично не застосовується. Вона передбачає одноразове виконання кожної із фаз проекту які своєю чергою, суворо слідують друг за одним. Простими словами можна сказати, що в рамках цієї моделі будь-якої миті часу команда бачить тільки попередню та наступну фазу. У реальній розробці ПЗ доводиться бачити весь проект повністю і повертатися до попередніх фаз, щоб виправити недоробки чи щось уточнити (Додаток А, рис. А.2).

До недоліків водоспадної моделі прийнято відносити той факт, що участь користувачів ПЗ у ній або не передбачено взагалі, або передбачено лише побічно на

стадії одноразового збору вимог. З точки зору тестування ця модель погана тим, що тестування в явному вигляді з'являється тут лише з середини розвитку проекту, досягаючи свого максимуму наприкінці [6].

Проте водоспадна модель часто інтуїтивно застосовується під час виконання щодо простих завдань, та її недоліки послужили чудовим відправним пунктом до створення нових моделей. Також ця модель у дещо удосконаленому вигляді використовується на великих проектах, у яких вимоги дуже стабільні і можуть бути добре сформульовані на початку проекту таких як аерокосмічна область або медичне програмне забезпечення [6].

V-подібна модель (V-model) є логічним розвитком водоспаду. На відміну від водоспадної моделі, V-подібна модель життєвого циклу ПЗ може містити той самий набір стадій, але принципова відмінність у тому, як ця інформація використовується у процесі реалізації проекту (Додаток А, рис. А.3).

Дуже спрощено можна сказати, що при використанні v-подібної моделі на кожній стадії на спуску потрібно думати про те, що і як відбуватиметься на відповідної стадії на підйомі. Тестування тут з'являється вже на ранніх стадіях розвитку проекту, що дозволяє мінімізувати ризики, а також виявити та усунути безліч потенційних проблем до того, як вони стануть реальними проблемами [6].

Ітераційна модель (Iterative model) є фундаментальною основою сучасного підходу до розробки ПЗ. Ключовою особливістю даної моделі є розбиття проекту на відносно невеликі ітерації, кожена із яких може включати всі класичні стадії, притаманні водоспадній і v-подібної моделям. Підсумком ітерації є збільшення функціональності продукту, виражене у проміжному білді (build) (Додаток А, рис. А.4) [6].

Довжина ітерацій може змінюватись в залежності від безлічі факторів, проте сам принцип багаторазового повторення дозволяє гарантувати, що і тестування, і демонстрація продукту кінцевому замовнику буде активно застосовуватися з самого початку та протягом усього часу розроблення проекту.

Спіральна модель (Spiral model) є окремим випадком ітераційної моделі, в якій особлива увага приділяється управлінню ризиками, що особливо впливають на організацію процесу розробки проекту та контрольні точки. Вона була

запропонована Баррі Бахом у 1986 році і орієнтована на реалізацію великих, складних і ризикованих проектів [6].

Тут явно виділено чотири ключові фази: опрацювання задач, альтернатив та обмежень; оцінка ризиків та прототипування; розробка проміжної версії продукту; планування наступного циклу (Додаток А, рис. А.5).

Гнучка модель (Agile model) є сукупністю різних підходів до розробки ПЗ та базується на так званому agile-маніфесті. Люди та взаємодія важливіші за процеси та інструменти. Працюючий продукт важливіший за вичерпну документацію. Співпраця із замовником важливіша за погодження умов контракту. Готовність до змін важливіша за дотримання початкового плану.

Дуже спрощено можна сказати, що гнучка модель являє собою полегшену з погляду документації суміш ітераційної та спіральної моделей, при цьому слід пам'ятати про «agile-маніфест» і всі переваги, що з нього випливають та недоліки циклу (Додаток А, рис. А.6).

Головним недоліком гнучкої моделі вважається складність її застосування і навіть часте хибне використання її підходів, викликане нерозумінням фундаментальних принципів моделі. Проте можна стверджувати, що дедалі більше проектів починають використовувати гнучку модель розробки.

1.4 Методологічний апарат та процесні моделі забезпечення якості програмних систем

Техніки тест-дизайну - це методи і підходи, що використовуються для розробки тестових випадків і сценаріїв з метою ефективного виявлення дефектів у програмному забезпеченні. Вони допомагають систематизувати процес тестування, забезпечують охоплення різних аспектів системи і підвищують якість тестування. Основні техніки тест дизайну це еквівалентне розбиття, аналіз граничних значень, тестування переходів станів, попарне тестування і вгадування помилок [11].

Техніка еквівалентного розбиття передбачає розділення тестових даних на класи, де всі елементи певним чином схожі. Цей метод має сенс лише тоді, коли компоненти схожі та можуть вписатися в спільну групу. Вибір цього методу означає, що ми будемо тестувати лише кілька значень з кожної групи. Це не гарантує, що

решта значень, не охоплених тестами, будуть без помилок. Ми лише припускаємо, що використання кількох елементів з групи буде досить ілюстративним [11].

Припустимо, є інтернет-магазин, який пропонує різні тарифи на доставку залежно від ціни кошика. Наприклад, ціна доставки для замовлень менше 100 доларів становить 15 доларів. Ціна доставки для замовлень понад 100 доларів становить 5 доларів. Безкоштовна доставка для замовлень понад 300 доларів. Тоді цінові діапазони для роботи будуть такі: від 1 до 100 доларів, від 100 до 300 доларів, від 300 доларів і вище (Додаток А, рис. А.7).

Отже, можна просто вибрати кілька чисел з кожного цінового діапазону та припустити, що решта подібних вхідних даних покажуть однакові результати.

Аналіз граничних значень подібний до попереднього методу. Можна навіть сказати, що він базується на еквівалентному розбитті класів. Проте різниця полягає у тому, що хоча все ще групуються дані в еквівалентні класи, але не тестуються значення лише з певного класу. Натомість ми перевіряються граничні значення, ті, що знаходяться на межах класів [11].

Наприклад, візьмемо попередній сценарій з різними тарифами на доставку. У нас ті самі дані, але інший підхід до їх використання (Додаток А, рис. А.8).

Припускаючи, що помилки найімовірніше виникатимуть на межах, ми тестуємо лише граничні числа.

Перехід станів візуалізує стани програмної системи в різні часові рамки та на етапах використання. Візуальну інформацію легше сприймати порівняно зі словесним описом. Тому перехід станів дозволяє швидше отримати максимальне тестове покриття. Цей метод ефективний для створення наборів тестів для систем, які мають багато варіацій станів. Буде корисно, якщо ви тестуватимете послідовність подій зі скінченною кількістю варіантів введення [11].

Найпростішим прикладом переходу станів є візуалізація входу в обліковий запис під час веб-тестування або тестування мобільного додатка. Припустимо, ми тестуємо систему, яка пропонує обмежену кількість спроб ввести правильний пароль. Якщо користувач не вводить правильний пароль три рази, тоді система блокує доступ (Додаток А, рис. А.9).

Попарне тестування вважається найскладнішим і найзаплутанішим з п'яти

методів розробки тестів. І для цього є вагома причина. Попарне тестування базується на математичних алгоритмах, а саме на комбінаториці. Воно дозволяє створювати унікальні пари та тестувати величезну кількість вхідних даних у різних комбінаціях, але розрахунки можуть ускладнитися. Щоб охопити максимум можливостей тестовими сценаріями, які вимагатимуть мінімального часу для тестування, потрібно правильно зіставити дані, комбінуючи пари певним чином на основі розрахунків [11].

Припустимо, існує мережа пекарень, які продають яблучні пироги та чізкейки онлайн. Кожна з них доступна у трьох розмірах – малий, середній та великий. Пекарня пропонує негайну та заплановану доставку за адресою, а також опцію самовивозу. Пекарня працює у трьох містах. Крім того, користувач може замовити до трьох товарів одночасно.

Якщо перевіряти всі можливі вхідні дані, це буде 2x3x3x3x2x2=216 дійсних комбінацій порядку. Однак перевіряти кожну з них було б недоцільно. Натомість можна розташувати змінні таким чином, щоб охопити максимальну кількість сценаріїв. Для цього потрібно згрупувати змінні або скористатися одним із інструментів, які можуть зробити це за вас, наприклад pairwiseTool [12]. У результаті ми отримали 17 сценаріїв, здатних охопити всі 216 комбінацій (Додаток А, рис. А.10).

При вгадуванні помилок інженер з контролю якості прогнозує, де можуть з'явитися помилки, спираючись на попередній досвід, знання системи та вимоги до продукту [13]. Таким чином, спеціаліст з контролю якості повинен виявляти місця, де накопичуються дефекти, та приділяти підвищену увагу цим областям [11]. Чим більше досвіду має QA спеціаліст, тим більше сценаріїв вгадування помилок він може швидко придумати.

Життєвий цикл тестування - це послідовність етапів, які проходять під час процесу тестування програмного забезпечення, від планування до завершення та підведення підсумків (Додаток А, рис. А.11). Ефективне управління цим циклом забезпечує систематичний підхід до виявлення, документування і виправлення помилок, а також підвищує якість кінцевого продукту [6].

Аналіз вимог. На цьому етапі вимоги отримуються від клієнта або

керівництва проекту та чітко визначаються, щоб визначити, що потрібно реалізувати в кінцевому продукті. Ці вимоги можуть стосуватися функціональності програмного забезпечення або інших аспектів [14].

Планування тестування. На цьому етапі визначаються цілі тестування, обсяги, ресурси, строки та відповідальні особи. Важливим є аналіз вимог до системи, що дозволяє сформулювати стратегію тестування, обрати відповідні техніки і інструменти. В результаті створюється детальний план тестування - документ, що слугує дорожньою картою для всієї команди. Планування також включає визначення критеріїв готовності до релізу, обсягів тестування, а також ризиків і обмежень [14].

Розробка тест-кейсів. Розробка тестових випадків починається після завершення планування тестування. На цьому етапі команда тестувальників створює тестові випадки та готує необхідні тестові дані [14].

Налаштування тестового середовища. Після підготовки тестових сценаріїв, команда переходить до проектування тестової інфраструктури. Це включає в себе налаштування необхідного обладнання, програмного забезпечення та автоматизаційних інструментів [14].

Виконання тестів. Наступний етап - це виконання тестування. Тут команда тестувальників запускає створені сценарії, фіксує результати і аналізує їх. В процесі виконання тестів фіксуються всі відхилення від очікуваних результатів, тобто дефекти, баги та недоліки системи [14].

Аналіз результатів тестування. Останній етап - це аналіз результатів і підсумкове звітування. Після завершення виконання тестів всі отримані дані аналізуються для визначення рівня покриття функціональності, кількості знайдених багів, їх серйозності та розподілу. Створюються звіти, що містять інформацію про стан системи, виявлені недоліки і рекомендації щодо виправлення.

Якість програмного забезпечення визначається не лише його надійністю та стабільністю, а й відповідністю визначеним вимогам і характеристикам. Тому у процесі тестування важливо виділяти дві основні групи - функціональні та нефункціональні види тестування. Кожен із них має свою ціль, методологію і особливості, що дозволяють всебічно оцінити якість продукту.

Функціональне тестування визначає, чи працює програма або мобільний

застосунок відповідно до технічних та бізнес-вимог.

Функціональні тести зазвичай виконуються перед нефункціональними тестами та виконуються вручну. Тестер надає певні вхідні дані програмі та порівнює результат з очікуваним. Функціональні тестери не переймаються вихідним кодом; вони зосереджуються на перевірці функціональності [15].

До функціонального типу тестування прийнято відносити [15]:

Димове тестування. Використовується перед фактичним тестуванням, щоб переконатися, що основна функція працює належним чином, перш ніж інші процедури тестування будуть налаштовані.

Модульне тестування. Виконується для перевірки окремих блоків або компонентів системи. Модульні тести зазвичай автоматизовані та зосереджені на перевірці того, що певні функції або сегменти коду працюють належним чином окремо.

Інтеграційне тестування. Виконується для перевірки зв'язку між різними компонентами системи. Ці тести необхідні для того, щоб різні частини системи працювали разом належним чином. Інтеграційні тести можна виконувати вручну або автоматично.

Тестування API. Тип тестування, який зосереджений на перевірці функціональності програмного інтерфейсу (API) програми. Тести API зазвичай виконуються за допомогою інструментів автоматизації, хоча в деяких випадках може використовуватися і ручне тестування.

Регресійне тестування. Тестування програмного забезпечення, яке перевіряє, чи програмна система все ще відповідає вимогам після внесення змін. Мета регресійного тестування – переконатися, що зміни не принесли жодних нових дефектів у систему.

Тестування сумісності. Перевіряє, чи може програма взаємодіяти з іншими програмними системами та компонентами без конфліктів.

Тестування інтерфейсу користувача. Необхідне для визначення того, чи будуть користувачі взаємодіяти з інтерфейсом, як заплановано. Це допомагає визначити частини програмного забезпечення, які користувач зазвичай використовує, і чи ці елементи поводяться відповідно до вимог.

Нефункціональне тестування досліджує всі аспекти, які не охоплені функціональним тестуванням. Воно охоплює продуктивність, доступність, масштабованість та надійність програмного забезпечення.

Цей тип тестування вимагає від тестувальника більшої креативності. Він не має нічого спільного з механічною роботою правого кліку. Фахівці з тестування повинні розробляти стратегії для збору очікувань клієнтів та надавати набір тестів для перевірки того, як ці очікування виконуються.

До нефункціонального типу тестування відносяться[15]:

Тестування локалізації. Гарантує, що продукт відповідає вимогам місцевої аудиторії. Наприклад, якщо ви хочете локалізувати мобільний додаток, створений для ринку США, та перекласти його китайською мовою, вам потрібно буде виконати цю форму тестування.

Тестування безпеки. Використовується для виявлення системних недоліків та встановлення того, наскільки добре захищені конфіденційні дані та внутрішні ресурси. Тестування безпеки має вирішальне значення для оцінки вразливостей та впровадження заходів захисту від потенційних загроз, що робить його невід'ємною частиною комплексної стратегії тестування програмного забезпечення.

Тестування на перенесення. Визначає, наскільки просто перенести програмний компонент або програму з одного обладнання або операційної системи на інше.

Тестування на аварійне відновлення. Використовується для оцінки того, скільки часу потрібно для відновлення та наскільки успішно програма відновлює дані після збоїв або перебоїв у роботі мережі.

Тестування продуктивності. Перевіряє продуктивність програмного застосунку. Мета полягає в тому, щоб переконатися, що система може відповідати необхідним рівням продуктивності. Цей вид тестування в свою чергу поділяється на:

Об'ємне тестування. Перевіряє наскільки ефективно система працює в умовах зростаючого обсягу оброблюваних даних.

Стрес-тестування. Тип тестування продуктивності який перевіряє, як працює програмне забезпечення під впливом різних навантажень та стресів на функціональність додатка. Стрес-тестування оцінює стабільність, стійкість та

продуктивність системи в екстремальних або несприятливих умовах.

Тестування надійності. Має на меті перевірку працездатності додатка під час тривалого тестування з середнім рівнем навантаження.

Тестування навантаження. Полягає в тому, щоб знати, як програма реагуватиме під певним навантаженням. Автоматизоване тестування, яке імітує роботу певної кількості бізнес-користувачів на загальному ресурсі. Це критично важливо, якщо ви хочете забезпечити споживачам сумісну продуктивність. Тестування навантаження дозволяє оцінити продуктивність вашої системи під високими піковими навантаженнями.

Тестування на витривалість. Використовується для оцінки ефективності програми під великими навантаженнями. Однак навантаження постійно зростає та триває протягом тривалого періоду.

Висновки до першого розділу

Було розглянуто фундаментальні поняття та історичний розвиток тестування програмних продуктів, що дозволило сформувати цілісне розуміння процесу та його важливості у сучасних умовах. Проведений аналіз особливостей автоматизованого тестування висвітив його переваги, зокрема підвищення ефективності, точності та швидкості виявлення дефектів у порівнянні з ручними методами. Розгляд процесів тестування і моделей розробки допоміг окреслити роль автоматизації в життєвому циклі створення програмних систем, а також визначити підходи до вибору технік тест-дизайну, що забезпечують систематичний і всебічний підхід до створення тестових сценаріїв. Вивчення основних принципів та видів тестування дозволило охарактеризувати різноманіття підходів та їхню роль у забезпеченні якості програмного забезпечення.

Особливий акцент було зроблено на аналіз застосування методів штучного інтелекту у тестуванні, що відкриває перспективи для підвищення автоматизації та ефективності процесів контролю якості. Загалом, розділ створює міцну теоретичну базу для подальших досліджень і практичних впроваджень у сфері автоматизованого тестування, підкреслюючи його важливість у сучасному розвитку програмної індустрії.

Розділ 2 Аналіз сучасних технологій автоматизованого тестування із застосуванням штучного інтелекту

2.1 Застосування штучного інтелекту в автоматизованому тестуванні програмного забезпечення

Коли мова йде про інструменти штучного інтелекту, ймовірно, спадають на думку такі інструменти штучного інтелекту, як ChatGPT або Google Gemini. Але технологія штучного інтелекту – це набагато більше, ніж просто ChatGPT, вона швидко розвивається і змінюється щодня.

В автоматизації тестування AI виходить за рамки простої автоматизації існуючих тестів. Він використовує алгоритми машинного навчання для навчання на тестових даних, виявлення закономірностей та прийняття інтелектуальних рішень. AI автоматизує виснажливі завдання, такі як генерація тестових даних та повторювані взаємодії з інтерфейсом користувача. Використовуючи прогнозну аналітику та моделі глибокого навчання, AI може передбачати високоризикові області програми та оптимізувати пріоритезацію тестів, звільняючи тестувальників від необхідності зосередитися на стратегії тестування високого рівня та дослідницькому тестуванні.

Також, використання цієї технології автоматизованому тестуванні може аналізувати поведінку програми та взаємодію з користувачами, щоб виявити області з низьким покриттям тестування. Потім він може рекомендувати нові сценарії тестування, забезпечуючи більш повне тестування. Може автоматично виявляти та адаптуватися до динамічних змін у тестованій програмі. Це зменшує кількість хибнопозитивних результатів та підтримує стабільність тестування, заощаджуючи цінний час для тестувальників. Автоматизуючи повторювані завдання та визначаючи найважливіші області для тестування, AI в автоматизованому тестуванні може значно скоротити цикли тестування.

Інструменти візуального тестування на базі AI можуть виявляти ледь помітні візуальні регресії, які можуть бути непомітними для традиційних тестів

на основі скриптів. Крім того, AI може аналізувати дані виконання тестів, щоб визначити потенційні закономірності дефектів та їх першопричини.

Найпопулярніші застосування AI в автоматизації тестування:

Тестування API. Автоматизація тестування ШІ може аналізувати поведінку API та автоматично генерувати тестові випадки, що охоплюють різні граничні випадки та сценарії помилок. Це забезпечує ретельне тестування API та зменшує ризик проблем інтеграції. Він також може імітувати кросбраузерні та сценарії, забезпечуючи сумісність програм у різних середовищах [16].

Тестування продуктивності. AI може аналізувати дані про продуктивність та прогнозувати потенційні вузькі місця в програмі. Цей проактивний підхід до тестування продуктивності дозволяє розробникам вирішувати проблеми продуктивності на ранніх етапах циклу розробки [16].

Візуальні локатори. У тестуванні користувацького інтерфейсу на основі AI візуальні локатори тепер можуть знаходити компоненти у веб-додатку за допомогою зору, навіть якщо їхні локатори були змінені. Це дозволяє уникнути необхідності жорстко кодувати ідентифікатори доступності або інші локатори. Крім того, інтелектуальні системи автоматизації тепер можуть використовувати OCR та інші алгоритми розпізнавання зображень для відображення програми, виявлення візуальних регресій та перевірки елементів [16].

Аналітика ШІ для даних автоматизації тестування. Тести генерують велику кількість даних, які необхідно проаналізувати для визначення сенсу. Додавання ШІ до цього процесу значно підвищує його ефективність. Алгоритми на базі ШІ можуть виявляти та класифікувати помилки. Потужніші системи ШІ можуть виявляти хибнонегативні та справді позитивні результати в тестових сценаріях [16].

2.2 Аналіз існуючих AI інструментів тестування

У 2025 році провідні команди контролю якості звертаються до передових інструментів для тестування на основі штучного інтелекту, починаючи від інструментів автоматизації тестування на основі штучного інтелекту і

закінчуючи інструментами тестування програмного забезпечення на основі штучного інтелекту, щоб усунути нестабільні тести, зменшити обсяг обслуговування та йти в ногу зі швидкозмінними кодовими базами.

Огляд найкращих інструментів для тестування на основі штучного інтелекту для оптимізації зусиль з автоматизації:

CoTester - це агент штучного інтелекту корпоративного рівня для тестування програмного забезпечення. Він вивчає контекст вашого продукту, адаптується до вашого робочого процесу та виконує тестування так само, як досвідчений користувач [17].

Цей інструмент щось на кшталт помічника для тестування програмного забезпечення на базі штучного інтелекту, який інтелектуально генерує та запускає тестові випадки з вашого схвалення, а потім адаптує їх у міру розвитку вашого застосунку, щоб забезпечити стабільність та стійкість автоматизації.

CoTester миттєво створює повні тестові випадки з історій JIRA або URL-адрес активних застосунків, самостійно відновлює скрипти під час виконання за допомогою AgentRx та запускає тести в реальних браузерах з журналами налагодження та виконання в режимі реального часу.

Testim.io - інструмент на базі штучного інтелекту допомагає пришвидшити процес випуску додатків завдяки швидшому та точнішому створенню тестів. Testim.io як найшвидший спосіб створення тестів та безперешкодного фіксування навіть складних дій. Більше того, функція автоматичного групування дозволяє швидко виявляти схожі кроки під час тестування та автоматично пропонувати спільні групи як заміну [18].

Functionize пропонує колекцію інструментів тестування GenAI, які можуть протестувати навіть найскладніші додатки. Як універсальна платформа для тестування, вона використовує тести на основі машинного навчання, які використовують великі дані для розуміння оновлень сайту та самовідновлення, щоб йти в ногу з розвитком додатка, уникаючи постійного обслуговування тестів [19]. За допомогою Functionize можна тестувати додатки, бази даних, API, файли .pdf, таблиці Excel та інші цифрові активи. Хмарна інфраструктура,

спеціально створена для автоматизації тестування на базі штучного інтелекту, спрощує масштабування.

Mabl пропонує хмарний підхід до якості програмного забезпечення на різних платформах. Він забезпечує безперебійну підтримку веб та мобільного тестування, а також тестування API, з можливістю імпорту тестів з Postman або створення власних низькокодових комплексних тестів API [20].

Testers.ai охоплює всі потреби в автономному тестуванні веб-додатків, від функціональності та продуктивності до API та доступності. Є можливість симулювати взаємодію з користувачами, генерувати відгуки користувачів, проводити конкурентний бенчмаркінг та тестувати конфіденційність і безпеку. Також можна отримати доступ до комплексного аналізу продуктивності, щоб виявити навіть найменші помилки [21].

Sauce Labs - універсальний центр для веб та мобільного тестування. Він пропонує широкий спектр можливостей low-code тестування, спрямованих на членів команди QA з мінімальним технічним досвідом або без нього [22]. Безпечна універсальна платформа забезпечує оптимізоване розповсюдження та управління додатками для Android та iOS, а також просте кросбраузерне тестування. Вона також пропонує підтримку кількох фреймворків автоматизації тестування, таких як Appium, Espresso, Selenium та Cypress.

Tricentis Tosca - комплексний інструмент тестування який включає Tosca Copilot, асистента з автоматизації на базі GenAI, який використовує інтерфейс чату, щоб допомогти знаходити, вивчати та оптимізувати тестові ресурси [23]. Як контекстно-залежний інструмент, Tricentis Tosca допомагає отримати безпрецедентний контроль над бібліотекою тестів, підсумовуючи складні тести простою мовою, підвищуючи продуктивність вашої команди. Він пропонує рішення для всіх цифрових проектів, від модернізації бізнес-додатків до переходу в хмару.

Testcraft - це інструмент із відкритим кодом на базі штучного інтелекту, який пропонує широкий спектр тестів без кодування, щоб допомогти навіть нетехнічним членам команди створювати та запускати тести. Використовуючи

можливості GPT-4, можна створювати різноманітні тести для різних мов програмування та фреймворків автоматизації на TestCraft [24].

Крім того, штучний інтелект платформи допомагає генерувати нові ідеї для тестування, щоб охопити кожен можливий сценарій. Також є можливість створювати налаштовувані набори тестів, щоб гарантувати, що кожен додаток відповідає найвищим стандартам якості.

Keysight Eggplant Test - це автоматизація на базі штучного інтелекту яка взаємодіє з додатком як з реальним користувачем, надаючи розумний та дієвий зворотний зв'язок без необхідності доступу до вихідного коду. Також можна виконувати тести для прогнозування поведінки під час виконання за різних умов, підвищуючи надійність та зручність використання додатка [25].

Perfecto - це високо якісна платформа автоматизації тестування для мобільних додатків. Її потужний GenAI допомагає створювати та виконувати тести простою мовою, а також генерувати та вставляти тестові зображення для безперебійного процесу тестування.

Інструмент для тестування на основі штучного інтелекту підтримує навіть найскладніші випадки використання, включаючи геолокацію, віртуалізацію мережі, біометрію тощо. Його хмара корпоративного рівня дозволяє проводити тестування віртуальних та реальних пристроїв з безпрецедентними можливостями безперервного тестування [26].

Уніфікована платформа **Testsigma** робить автоматизоване тестування простішим, ніж будь-коли. Вона дозволяє команді контролю якості автоматизувати тести простою англійською мовою або GenAI. Можна автоматизувати тести в різних браузерах без написання додаткових скриптів, додавати візуальні перевірки одним кліком, скорочувати час обслуговування тестів за допомогою самовідновлювальних тестів та багато іншого [27].

Платформа **aqua ALM** дозволяє керувати та запускати ручні та автоматизовані тести з одного спеціалізованого інструменту управління контролем якості. Його модель штучного інтелекту розуміє контекст і семантику потреб у тестуванні та може допомогти вам генерувати вимоги до тестів. Дає

можливість поєднувати кілька інструментів штучного інтелекту для автоматизованого тестування та перевіряти попередні тестові запуску на предмет покращень. А завдяки можливостям управління проектами команда контролю якості завжди може контролювати планування та визначення пріоритетів тестування [28].

AccelQ зосереджується на автоматизації бізнес-процесів. Його можливості тестування без коду допомагають інтуїтивно та масштабно охоплювати складні реальні сценарії. Він також підтримує ручнt тестування завдяки відстеженню, контролю та інтеграції [29].

TestComplete спрощує автоматизацію функціональних тестів інтерфейсу користувача для будь-якої програми. Незалежно від тестування на реальному чи віртуальному пристрої, тести ключових слів у кількох браузерах, ОС та комбінаціях пристроїв допоможуть виявити та виправити помилки [30].

Візуальне розпізнавання на основі штучного інтелекту допомагає ідентифікувати динамічні елементи інтерфейсу користувача, заощаджуючи час та допомагаючи керувати об'єктами в одному репозиторії.

Sealights - це розширена платформа аналізу якості програмного забезпечення, яка забезпечує повну видимість ризиків якості в усьому конвеєрі доставки. Застосовуючи штучний інтелект та машинне навчання, SeaLights надає прозорість та показники, необхідні для швидкої доставки програмного забезпечення без шкоди для якості. Більше того, можна впровадити розумнішу практику тестування, вибираючи та запускаючи лише найрелевантніші тести для кожної збірки [31].

Worksoft Certify – це безкодова платформа яка пропонує першокласне безперервне автоматизоване тестування для корпоративних пакетних додатків. IT-команда та нетехнічні спеціалісти можуть працювати паралельно, щоб тестувати ваші процеси в реальних сценаріях. Платформа автоматизації тестування Worksoft Certify дозволяє динамічно адаптуватися до змін за допомогою визначень смарт-об'єктів, які не вимагають постійної розробки окремих тестових сценаріїв [32].

Avo Automaton - це платформа на базі штучного інтелекту, яка забезпечує легке комплексне тестування ваших можливостей CI/CD. Наприклад, за допомогою Avo Genius можна автономно створювати тести за допомогою інтелектуального планувальника, інтегруватися з усіма інструментами ALM та використовувати попередньо вбудовану автоматизацію для таких середовищ, як Oracle та SAP [33]. Design Studio надає повний огляд усієї ієрархії тестування, допомагаючи краще планувати та розподіляти ресурси. Крім того, є доступ до розширеної аналітики впливу та ідентифікаторів об'єктів, які можуть самовідновлюватися, таким чином не відстаючи від розвитку додатка.

QA Wolf - це рішення для тестування програмного забезпечення з відкритим кодом, яке допомагає гнучким командам досягти високого рівня охоплення тестами. Розроблене для швидкості та масштабованості, воно піклується як про інфраструктуру, так і про фактичне написання ваших тестів.

Модель QA Wolf «людина в циклі» означає, що штучний інтелект не просто генерує та перевіряє тести окремо, а працює з досвідченими інженерами, тому вона не є неконтрольованою автоматизацією тестування, яка дає збій, коли це меньш потрібно [34].

ProdPerfect занурюється в дані про трафік у реальному часі, автоматично створюючи та підтримуючи тести додатків на рівні браузера. Замість того, щоб здогадуватися, якими шляхами найчастіше ходять користувачі, він визначає реальні робочі процеси та відтворює їх у тестових середовищах [35].

Але це не обмежується лише симуляцією - його механізм безперервного тестування означає, що пакет адаптується до змін поведінки користувачів, надаючи вам високоточну аналітику з мінімальним ручним втручанням.

ReTest - це програмне забезпечення для автоматизації тестування графічного інтерфейсу, яке переосмислює методи обробки регресійних тестів. Воно займає унікальну позицію щодо тестування. Завдяки своєму диференціальному підходу до тестування, ReTest створює розумні базові лінії ваших застосунків та позначає будь-які ненавмисні візуальні чи функціональні зміни, якими б незначними вони не були. Вам не потрібно писати сценарії чи

детально визначати очікувані результати. Воно є важливим у гнучких середовищах, де оновлення інтерфейсу користувача є частими. Воно дозволяє вам зосередитися на інноваціях, не зациклюючись на повторюваному обслуговуванні тестів [36].

Applitools - це платформа автоматизованого тестування інтерфейсів користувача (UI), яка використовує технології штучного інтелекту для перевірки візуальної цілісності додатків. Вона дозволяє автоматично знаходити візуальні розбіжності між версіями веб- або мобільних додатків, що робить процес тестування більш точним і швидким. Applitools особливо корисний для тестування складних графічних елементів, адаптивних дизайнів та багатоплатформних застосунків [37].

Explyt - це сучасна платформа для автоматизації тестування веб-додатків, яка використовує штучний інтелект і машинне навчання для створення, виконання та аналізу тестових сценаріїв. Вона спрямована на підвищення ефективності процесу тестування, зменшення людських помилок і забезпечення високої якості програмного забезпечення [38].

Зараз доступно багато інструментів автоматизації тестування штучного інтелекту, але не кожне рішення забезпечує однаковий рівень надійності чи ефективності. Оцінюючи платформи для тестування штучного інтелекту, важливо зосередитися на ключових можливостях, які забезпечують розумніше створення тестів, швидше виконання та легше обслуговування. Ось деякі важливі характеристики, які слід враховувати, вибираючи правильний інструмент автоматизації контролю якості на основі штучного інтелекту для ваших потреб.

Масштабованість: незалежно від того, скільки коду генерується, інструмент для тестування штучного інтелекту повинен чудово виконувати тести паралельно в кількох виробничих середовищах без зниження продуктивності. Ця гнучкість також означає, що не потрібно буде перемикати інструменти в міру зростання вашого проекту, що заощадить час і ресурси.

Доступність: інструмент повинен інтегрувати доступність у функціональне та інтерфейсне тестування та оцінювати ключові аспекти доступності, такі як контрастність кольорів, сумісність з програмою зчитування з екрана та навігація за допомогою клавіатури. Рання інтеграція цих перевірок забезпечує відповідність стандартам доступності та більш плавний користувацький досвід для всіх.

Повне охоплення: інструмент для тестування повинен мати можливість тестувати програму в широкому діапазоні браузерів, пристроїв та комбінацій ОС, щоб охопити якомога більше реальних сценаріїв.

Інтеграція з CI/CD: інструмент для тестування повинен мати можливість інтегруватися з конвеєрами CI/CD для автоматичного створення та адаптації тестових випадків на основі результатів попереднього виконання.

Точність: інструменти для тестування AI корисні лише настільки, наскільки вони точні. Платформа має бути такою, яка чітко пояснює, чому пропонується певний тестовий випадок, виправлення або оптимізація.

Простота використання для непрограмістів: тестування можна пришвидшити, дозволивши навіть нетехнічним членам команди без навичок кодування створювати, запускати та підтримувати тести.

2.4 Перспективи розвитку штучного інтелекту в автоматизації тестування програмного забезпечення

Очікується, що штучний інтелект в автоматизації тестування змінить роль тестувальників програмного забезпечення для автоматизації, але навряд чи він повністю їх замінить. ШІ може автоматизувати деякі завдання тестування, такі як створення тестів з варіантів використання або шляхом спостереження за діями людини-тестувальника. Однак, люди-тестувальники все ще незамінні завдяки своїм когнітивним навичкам, креативності та здатності вирішувати проблеми. Тестувальники привносять критичне мислення та знання предметної області, що дозволяє їм виявляти граничні випадки та розробляти тести, які виходять за рамки сценарійних взаємодій.

Крім того, людська інтуїція відіграє вирішальну роль у тестуванні програмного забезпечення. ШІ може мати труднощі з виявленням неочікуваної поведінки користувача або тонких невідповідностей інтерфейсу, які досвідчений тестувальник може помітити. Майбутнє автоматизованого тестування полягає в спільному підході, де ШІ обробляє повторювані завдання та звільняє тестувальників, щоб вони могли зосередитися на стратегії тестування високого рівня, дослідницькому тестуванні та використанні свого емоційного інтелекту для розуміння потреб та розчарувань користувачів.

Штучний інтелект – це галузь, що швидко розвивається, і його застосування в автоматизації тестування пропонує безмежні можливості. З кожним роком алгоритми AI стають все більш складними, що призводить до передових інтелектуальних рішень для автоматизації тестування. Хоча багато технологій автоматизації тестування на основі штучного інтелекту все ще перебувають на ранніх стадіях розвитку, потенціал для трансформації незаперечний. Наприклад, інструменти на базі штучного інтелекту, які можуть не лише автоматизувати завдання, але й навчатися та адаптуватися до складної поведінки програмного забезпечення. Це може призвести до створення самовідновлювальних тестів, які автоматично підлаштовуються під зміни інтерфейсу користувача, або до пріоритезації тестів на основі оцінки ризиків та впливу на користувача.

Висновки до другого розділу

У розділі було проведено наліз існуючих AI-інструментів показав їхню різноманітність і здатність виконувати складні задачі, що раніше вимагали значних людських ресурсів. Особливості вибору автоматизованого AI-інструменту залежать від специфіки проекту, цілей тестування та технічних вимог, що підкреслює необхідність ретельного аналізу та підбору відповідних засобів. Крім того, дослідження перспектив розвитку штучного інтелекту у цій сфері свідчить про його швидке зростання та потенціал для впровадження інноваційних рішень, здатних значно покращити процеси тестування.

Розділ 3 Проектування та реалізація авторського модуля/алгоритму на основі ШІ

3.1 Технічний стек і архітектура системи тестування

В проекті застосовуються різноманітні інструменти та фреймворки для написання, виконання та аналізу тестів. Аналіз існуючої системи має на меті визначити її сильні і слабкі сторони, а також можливості для покращення і інтеграції нових рішень на базі штучного інтелекту.

Behaviour-Driven Development (BDD) – це процес розробки програмного забезпечення, який скорочує розрив між бізнесменами та технічними спеціалістами шляхом створення системної документації, яка автоматично перевіряється на відповідність поведінці системи. Ця методологія розробки програмного забезпечення розвинулася з розробки на основі тестування [39].

BDD заохочує співпрацю між розробниками, відділом контролю якості та нетехнічними зацікавленими сторонами за допомогою спільної мови, яку кожен може зрозуміти. Цей підхід усуває розрив між технічними та бізнес-командами, використовуючи описи вимог природною мовою, які також можуть слугувати тестами [39].

Cucumber - це популярний інструмент, який підтримує BDD. Він дозволяє писати тести функцій у синтаксисі Gherkin, а потім реалізовувати визначення кроків, які пов'язують ці фрази з фактичним кодом, що тестує програму [39].

Фреймворк Cucumber побудований на кількох ключових компонентах, які відіграють унікальну роль у поєднанні текстових описів функцій з виконуваними тестовими скриптами. Файли зі сценаріями написані в синтаксисі Gherkin та містять тестові сценарії у форматі, зрозумілому для людини [39].

Визначення кроків діють як місток між кроками у звичайному текстовому форматі у файлах функцій та базовим кодом автоматизації. Кожен крок у файлі з сценаріями відповідає відповідному методу у файлі визначення кроку [39].

Selenide значно спрощує роботу з браузером підвищуючи стабільність тестів за рахунок автоматичного очікування елементів. Тести націлені на

взаємодію з веб-інтерфейсом, натискання кнопок, заповнення форм, перевірку елементів [40].

Page Object Model – це шаблон проектування, який використовується в автоматизації тестування, зокрема з фреймворками автоматизації інтерфейсу користувача, такими як Selenium або Selenide. Його основна мета – покращити зручність обслуговування, можливість повторного використання та читабельність автоматизованих тестів шляхом відокремлення логіки тестування від коду, який взаємодіє з інтерфейсом користувача веб-застосунку [40].

3.2 Інтеграція Applitools Eyes та Explyt у проект

Значущим етапом підвищення якості автоматизованого тестування є впровадження системи візуального контролю - Applitools Eyes [41]. Вона дозволяє автоматично порівнювати знімки екранів, виявляти візуальні розбіжності і забезпечувати стабільність інтерфейсу користувача. Інтеграція Applitools у вже існуючу систему автоматизації вимагає кількох ключових кроків: встановлення SDK, налаштування API ключів, модифікація сценаріїв та оптимізація процесу запуску тестів [41].

Для зручності рекомендується створити базовий клас або метод, що інкапсулює відкриття, перевірку і закриття Eyes, щоб уникнути дублювання коду у кожному сценарії. Перший запуск потребує калібрування системи для визначення її чутливості [42]. Це включає порівняння з базовими знімками і аналіз відхилень. Важливо врахувати різні сценарії: з мінімальними змінами, з великими оновленнями UI і з динамічним контентом. З отриманих знімків слід проаналізувати відхилення - визначити, які з них є допустимими, а які - вказують на дефекти [42].

Інтеграція Explyt у проект відкриває широкі можливості для автоматизації пошуку і аналізу поведінкових сценаріїв користувачів. Модифікація існуючих тестів забезпечує безперервність і зручність роботи з новим інструментом [43]. Автоматизація генерації тестів за допомогою моделей штучного інтелекту дозволяє створювати більш повне і актуальне тестове покриття без значних

людських зусиль [44]. Згенеровані сценарії автоматично конвертуються у формат, придатний для запуску, наприклад, Gherkin для Cucumber або скрипти для Selenide [44]. Це дозволяє швидко оновлювати тестове покриття без ручного написання сценаріїв.

3.3 Опис алгоритму створення автоматизованих тестів на основі AI

У цьому підрозділі описуються етапи алгоритму автоматизованого тестування програмного забезпечення, що базується на застосуванні інструментів Applitools для візуального тестування та Explyt для автоматизації процесів (Додаток Б, рис. Б.1).

Процес починається із підготовки до тестування, яке включає визначення цілей і сценаріїв тестування, а також налаштування середовища виконання. На цьому етапі здійснюється формалізація плану тестування, визначаються ключові сценарії, цілі та критерії успішності.

Наступним кроком є визначення типу тесту: візуальний або функціональний. Перевірка цієї умови дозволяє обрати відповідний підхід до створення автоматизованих тестів. Для візуального тестування створюються знімки екранів та налаштовується порівняння із базовими зображеннями за допомогою Applitools. Для функціонального тестування генерується набір тестових випадків та сценаріїв на основі вихідних даних за допомогою Explyt.

Після створення тестів алгоритм переходить до процесу запуску тестів використовуючи Junit з подальшим аналізом результатів і обробкою помилок. Якщо тест проходить успішно або він не проходить з тієї причини що знайшов баг у системі, тоді тестова система фіксує ці результати і генерує звіт, після чого алгоритм закінчується [45].

Однак, якщо тест не проходить через якусь іншу проблему, наприклад, неправильно згенерований код, тоді він виправляється інженером з тестування і тест знову запускається. Цикл повторюється до тих пір поки тест або не пройде успішно, або не знайде баг в системі. Далі автоматично формуються звіти з результатами.

Висновки до третього розділу

У розділі було розглянуто застосування таких інструментів, як Cucumber, Selenide, Junit, Applitools і Explyt, що забезпечує високий рівень автоматизації та стабільності тестового процесу. Вони дозволяють писати читабельні сценарії, що легко підтримуються і доповнюються, а також забезпечують гнучкість у виконанні тестів у різних середовищах та браузерах. Особливо важливим є використання Applitools для візуального тестування, яке дозволяє автоматизувати перевірку зовнішнього вигляду UI, що раніше вимагало значних людських ресурсів і було схильним до людських помилок.

Також було описано алгоритм створення тестів з використанням AI інструментів для двох видів тестів: візуального і функціонального. Такий підхід дозволить зменшити час на розробку нових автоматизованих тестів та розподілити ресурси на інші процеси розробки програмного забезпечення.

Розділ 4 Дослідження і оцінка ефективності штучного інтелекту в автоматизації програмного забезпечення

4.1. Методика проведення експериментальних досліджень

Об’єктом дослідження обрано сторінку з вибору автошколи у веб-застосунку. Цей модуль є ключовим для користувачів при виборі навчальної заклади, тому його тестування має важливе значення для забезпечення якості сервісу.

Для порівняльного аналізу використовувалась контрольна і експериментальна група. У контрольній групі автоматизація тестування здійснювалась з використанням стеку Java та бібліотеки Selenide. Це дозволило писати стабільні і гнучкі тести, але вони потребували багато часу на розробку та підтримку. Експериментальна група здійснювала автоматизацію тестування за допомогою інструментів Explyt та Applitools. Це дозволяє автоматично генерувати тестові сценарії та візуально перевіряти інтерфейс.

Для дослідження було сформовано еталонну вибірку з 17 типових тестових сценаріїв, що відображають основні функціональні можливості системи: перевірка назви сторінки, вибір автошколи за регіоном, перехід на сторінку вибраної автошколи, перевірка хедера та футера сторінки.

Для оцінки ефективності підходів була проведена оцінка декількох метрик. Час розробки тесту від початку створення тестового файлу до успішного проходження тесту у системі. Здатність тесту пройти успішно при внесенні незначних візуальних або технічних змін, наприклад, зміна кольору кнопки або її ID. Кількість знайдених реальних багів (функціональних та візуальних) у процесі тестування, у порівнянні з загальною кількістю існуючих багів, що були спеціально внесені для тестування.

Експеримент проводився у кілька етапів:

Ручне написання тестів. Створення 17 тестових сценаріїв вручну для модулю «Сторінка з вибору автошколи». Це включалє сценарії перевірки основних функцій, таких як вибір автошколи, натискання кнопок, перевірка правильності відображення інформації.

Автоматизація тестів за допомогою штучного інтелекту. За допомогою

платформи Explyt автоматично згенерувати тести на основі опису сценаріїв та зразків.

Порівняння показників. Вимірювання часу, необхідного для створення і запуску кожного тесту, а також фіксування результати проходження тестів, їх стійкість до змін інтерфейсу та кількість знайдених багів.

Аналіз результатів. Порівняння отриманих метрик для класичного та автоматизованого підходу з використанням штучного інтелекту.

4.2 Аналіз результатів впровадження AI-алгоритму

Процес автоматичної генерації тестів розпочинався з аналізу DOM-дерева сторінки з вибору автошколи. Explyt використовував сучасний інструмент для парсингу DOM і побудови дерева елементів, що дозволяло автоматично визначати інтерактивні компоненти сторінки. Однією з складностей було працювати з динамічними елементами, у деяких випадках елементи завантажувалися асинхронно, що ускладнювало їх автоматичну ідентифікацію.

Сторінки проходили візуальну перевірку за допомогою інструменту Applitools Eyes. Процес включав знімки екранів у різних станах та з різними даними. Процес включає в себе автоматичне захоплення скріншотів під час виконання тестів, виявлення відхилень і автоматичне позначення візуальних дефектів.

Під час виконання тесту було змінено селектор для назви автошколи «Житомирська область, Новоград-Волинська автошкола ВСА» h3 (Додаток В, рис. В.1) на h2 (Додаток В, рис. В.2). У класичному тесті,на базі фіксованого селектора тест автоматичного запуску зупинився з помилкою , оскільки селектор був змінений і тест не зміг знайти цільовий елемент. У випадку AI-алгоритму, система аналізує DOM-дерево та виявляє, що новий елемент має схожі ознаки, такі як, текст і схожу структуру. Вона автоматично ідентифікує елемент за допомогою альтернативних атрибутів і продовжує виконання тесту без зупинки.

4.3 Аналіз показників точності та статистичне обґрунтування результатів

Для оцінки ефективності автоматизованих тестів із використанням

штучного інтелекту було проведено аналіз за допомогою матриці помилок (Додаток В, табл. В.3).

True Positive випадки, коли AI правильно виявив реальний баг, були відсутні елементи для вибору автошколи, 4 тестові сценарії завершилися зі статусом Failed. False Positive випадки, коли ШІ позначив помилку там, де її фактично немає, через довге завантаження зображення, як дефект. False Negative випадків, коли AI не виявив існуючий баг, не зафіксовано. True Negative випадки, коли система правильно не позначала відхилення, тобто тестів які успішно пройшли було 12.

На основі цих даних обчислювалися метрики точності (precision): 0.8, повноти (recall): 1.0 та F1-міра: 0.89. Ці показники свідчать про досить високий рівень точності системи в автоматичному виявленні багів.

Зібрані дані для порівняння автоматизованого тестування з штучним підходом наведені у таблиці (Додаток В, таблиця В.4). Для оцінки ефективності застосування автоматизованого тестування використаємо просту формулу:

$$E = \frac{Тестове\ покриття * ількість\ дефектів}{Час\ на\ розробку\ тестів}$$

Тоді для оцінки ефективності тестування без AI і з його використанням:

$$E_{classic} = (0.5 * 3)/11 = 0.136$$

$$E_{AI} = (0.7 * 4)/3 = 0.933$$

Ефективність AI в 6.8 разів вища, ніж у традиційного імперативного підходу до автоматизації. Однак, на проектах з невеликою кількістю тестів різниця в часі може бути не суттєвою через час на налаштування AI-інструментів, але на проектах з більшою кількістю тестів економія стає стратегічною.

Для оцінки впливу масштабу проєкту було проведено лінійну екстраполяцію часу створення тестів для більшої кількості тестових випадків 17, 50, 100 і 200 тестів (Додаток В, рис. В.5). Екстраполяційний аналіз підтверджує, що точка рентабельності використання AI-алгоритму досягається вже при 50+ тестових сценаріях, що робить його впровадження доцільним для Agile-команд з частими релізами. Побудована діаграма демонструє, що зі збільшенням кількості тестів часовий розрив між ручним підходом та використанням AI зростає. Ручне

тестування характеризується лінійним зростанням часових витрат з високим коефіцієнтом, тоді як використання AI має значно нижчий темп зростання часу. Це підтверджує економічну доцільність використання AI на середніх і великих програмних проектах.

Висновки до четвертого розділу

У розділі обґрунтовано методику досліджень, що базується на порівняльному аналізі контрольної групи (класичний стек Java/Selenide) та експериментальної групи (інструменти Explyt та Applitools). Сформована еталонна вибірка з 17 тестових сценаріїв дозволила комплексно оцінити часові витрати, стійкість тестів та якість виявлення дефектів. Доведено високу адаптивність AI-алгоритму до динамічних змін графічного інтерфейсу (GUI). На відміну від класичного підходу, де зміна селектора (наприклад, з h3 на h2) призводить до зупинки тесту з помилкою, запропонована система на основі аналізу DOM-дерева та семантичних ознак елементів успішно ідентифікує об'єкти та продовжує виконання сценарію без втручання розробника.

Визначено показники точності та надійності системи за допомогою матриці помилок. Отримані метрики – точність (precision) 0.8, повнота (recall) 1.0 та F1-міра 0.89 – підтверджують здатність алгоритму безпомилково виявляти реальні функціональні баги за низького рівня хибнопозитивних спрацювань, спричинених затримками завантаження контенту.

Встановлено значну перевагу у продуктивності: порівняльний статистичний аналіз показав, що ефективність використання AI-інструментів у 6.8 разів вища порівняно з класичним написанням коду. Математична екстраполяція результатів підтвердила, що зі збільшенням масштабу проекту (до 200+ тестів) часовий розрив між ручним та автоматизованим підходами зростає нелінійно, що робить впровадження штучного інтелекту економічно доцільним для середніх та великих програмних систем.

ВИСНОВКИ

У роботі було досліджено можливості застосування інструментів автоматизованого тестування програмного забезпечення з використанням штучного інтелекту на прикладі інструменту Explyt та Applitools. Основною метою роботи було підвищення ефективності процесу тестування шляхом скорочення часових витрат, збільшення тестового покриття та підвищення якості виявлення дефектів.

Було отримано знання щодо процесів тестування, технік тест-дизайну і моделей розробки, підкреслюючи роль автоматизації у підвищенні ефективності, точності та швидкості виявлення дефектів. Аналіз сучасних методів і застосування ШІ у тестуванні дозволили окреслити перспективні напрями розвитку та розробити основу для подальших досліджень.

У ході виконання роботи було проаналізовано сучасні підходи до тестування програмного забезпечення, визначено їхні переваги та недоліки, а також обґрунтовано доцільність використання інструментів, що базуються на алгоритмах штучного інтелекту. Проведений огляд існуючих рішень показав, що автоматизоване тестування з використанням AI є перспективним напрямом розвитку інженерії програмного забезпечення.

У практичній частині роботи було проведено експериментальне порівняння ручного підходу до створення тестових випадків та автоматизованого підходу із застосуванням Explyt та Applitools. Отримані результати показали, що використання AI дозволяє суттєво зменшити час на створення тестів, підвищити тестове покриття та виявити більшу кількість дефектів у порівнянні з традиційним ручним тестуванням.

Проведено дослідження і оцінку ефективності розроблених систем. Проведені експерименти показали високий рівень точності та продуктивності автоматизованих тестів із застосуванням AI у порівнянні з традиційними підходами, що підтверджує перспективність і необхідність впровадження таких технологій у практику.

Запропонована у роботі розширена формула оцінки ефективності тестування, яка враховує тестове покриття, кількість виявлених дефектів та часові витрати, дозволила здійснити комплексну кількісну оцінку результатів експерименту. Застосування цієї формули підтвердило значну перевагу використання AI-інструментів, особливо у проектах із великою кількістю тестових сценаріїв.

Побудований аналіз масштабування показав, що зі зростанням кількості тестів часовий розрив між ручним тестуванням та автоматизованим тестуванням із використанням AI збільшується, що свідчить про економічну та практичну доцільність впровадження таких інструментів у середніх і великих програмних проєктах.

Таким чином, поставлені у дипломній роботі завдання було повністю виконано, а мету дослідження - досягнуто. Отримані результати можуть бути використані при виборі підходів до тестування програмного забезпечення, а також слугувати основою для подальших досліджень у сфері застосування штучного інтелекту в автоматизації процесів забезпечення якості програмних продуктів.

Загалом, результати дослідження демонструють, що застосування штучного інтелекту у автоматизованому тестуванні сприяє підвищенню якості програмного забезпечення, зменшенню витрат і часу на процес тестування, а також відкриває широкі можливості для інноваційних рішень у галузі забезпечення якості. Це створює наукову і практичну основу для подальшого розвитку інтелектуальних систем тестування і їхнього впровадження у сучасні розробки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. World Quality Report 2023-24. *Capgemini*. URL: https://www.capgemini.com/insights/research-library/world-quality-report-2023-24/ (дата звернення: 10.12.2025).

2. 17th State of Agile Report | Analyst Reports | Digital.ai. *Digital.ai*. URL: https://digital.ai/resource-center/analyst-reports/state-of-agile-report/ (дата звернення: 10.12.2025).

3. Amanneet Singh. Self-Healing in Software Test Automation: A Review of Techniques and Tools. International Journal of Computer Applications. 2022. Vol. 184, No. 12. P. 34–38.

4. The State of AI-augmented Software Testing Report / Gartner Inc. 2024. URL: https://www.gartner.com/en/documents/4015693 (дата звернення: 10.12.2025).

5. Meskini A., Nassif A. B. Artificial Intelligence in Software Testing: A Review. 2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC). 2023. P. 114–121.

6. Куликов С. Тестування програмного забезпечення. Базовий курс. 3-тє вид. 2025. 301 с.

7. Учасники проектів Вікімедіа. Автоматизоване тестування – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Автоматизоване_тестування (дата звернення: 03.10.2025).

8. Автоматизоване тестування - QALight. URL: https://qalight.ua/baza-znaniy/avtomatizovane-testuvannya/ (дата звернення: 09.10.2025).

9. The Way of the Web Tester: A Beginner's Guide to Automating Tests. Pragmatic Bookshelf, 2016. 258 p.

10. Учасники проектів Вікімедіа. Методологія розробки програмного забезпечення – Вікіпедія. *Вікіпедія*.

URL: https://uk.wikipedia.org/wiki/Методологія_розробки_програмного_забезпечення (дата звернення: 11.10.2025).

11. Badgett T., Myers G. J., Sandler C. Art of Software Testing. Wiley & Sons, Limited, John, 2015.

12. Pairwise Online Tool. *Pairwise Online Tool*. URL: https://pairwise.teremokgames.com/ (дата звернення-: 16.10.2025).

13. Тестування. Фундаментальна теорія. *dou.ua*. URL: https://dou.ua/forums/topic/13389/ (дата звернення: 18.10.2025).

14. Життєвий цикл тестування ПЗ (STLC). *it-notes.wiki*. URL: https://www.it-notes.wiki/software-testing/software-testing-life-cycle/ (дата звернення: 25.10.2025).

15. The Difference Between Functional and Non-Functional Testing. *solutionshub.epam.com*. URL: https://solutionshub.epam.com/blog/post/the-difference-between-functional-and-non-functional-testing (дата звернення-: 28.10.2025).

16. AI in Test Automation: A Comprehensive Guide. *TestGrid | Blog*. URL: https://testgrid.io/blog/ai-in-test-automation/ (дата звернення-: 29.10.2025).

17. Meet CoTester by TestGrid: Your AI Software Testing Agent. *TestGrid | Blog*. URL: https://testgrid.io/blog/cotester-by-testgrid/ (дата звернення-: 31.10.2025).

18. Automated UI and Functional Testing - AI-Powered Stability - Testim.io. URL: https://www.testim.io/ (дата звернення-: 02.11.2025).

19. Functionize - Enterprise AI Test Automation Platform with QA Agents. *Functionize - Enterprise AI Test Automation Platform with QA Agents*. URL: https://www.functionize.com/ (дата звернення-: 04.11.2025).

20. AI-Powered Testing for the Next Generation of Software | mabl. *AI-Powered Testing for the Next Generation of Software | mabl*. URL: https://www.mabl.com/ (дата звернення-: 05.11.2025).

21. testers.ai - AI Testing Agents. *testers.ai - AI Testing Agents*. URL: https://testers.ai/ (дата звернення-: 07.11.2025).

22. Sauce Labs: Cross Browser Testing, Selenium Testing & Mobile Testing. *Sauce Labs: Cross Browser Testing, Selenium Testing & Mobile Testing*. URL: https://saucelabs.com/ (дата звернення-: 10.11.2025).

23. AI-powered automated continuous testing - Tricentis Tosca. *Tricentis*. URL: https://www.tricentis.com/products/automate-continuous-testing-tosca (дата звернення-: 12.11.2025).

24. TestCraft. *TestCraft*. URL: https://home.testcraft.app/ (дата звернення-: 15.11.2025).

25. Eggplant Test - Automated Software Testing Tool | Keysight. URL: https://www.keysight.com/us/en/products/software/software-testing/eggplant-test.html (дата звернення-: 17.11.2025).

26. Automated Web & Mobile Testing | Perfecto by Perforce. *Automated Web & Mobile Testing | Perfecto by Perforce*. URL: https://www.perfecto.io/ (дата звернення-: 20.11.2025).

27. Testsigma: #1 Unified & Agentic Test Automation Platform. *Testsigma Agentic Test Automation Tool*. URL: https://testsigma.com/ (дата звернення-: 23.11.2025).

28. aqua ALM. *aqua ALM*. URL: https://www.aqua-alm.ch/en (дата звернення-: 25.11.2025).

29. ACCELQ: #1 AI-Powered Codeless Test Automation QA Tool. *ACCELQ*. URL: https://www.accelq.com/ (дата звернення-: 26.11.2025).

30. Automated Testing Tools | Desktop, Web & Mobile Test Automation | TestComplete. *Software Testing, Monitoring, Developer Tools | SmartBear*. URL: https://smartbear.com/product/testcomplete/ (дата звернення-: 26.11.2025).

31. Welcome! | Knowledge Base. *Welcome! | Knowledge Base*. URL: https://docs.sealights.io/knowledgebase (дата звернення-: 26.11.2025).

32. Certify Codeless Test Automation Software - Worksoft. URL: https://www.worksoft.com/certify/ (дата звернення-: 27.11.2025)

33. AI-Powered Enterprise No-code Test Automation Platform- Avo Automation. *AI-Powered Enterprise No-code Test Automation Platform- Avo Automation*. URL: https://avoautomation.com/ (дата звернення-: 27.11.2025).

34. QA Wolf | 80% automated test coverage in 4 months. *QA Wolf | 80% automated test coverage in 4 months*. URL: https://www.qawolf.com/ (дата звернення-: 27.11.2025).

35. ProdPerfect: Data-Driven E2E Test Automation for CI/CD. *ProdPerfect*. URL: https://prodperfect.com/ (дата звернення-: 27.11.2025).

36. Retest - GUI Test Automation Software. *retest*. URL: https://retest.de/ (дата звернення-: 27.11.2025).

37. Home. *AI-Powered End-to-End Testing | Applitools*. URL: https://applitools.com/ (дата звернення-: 27.11.2025).

38. Explyt - The AI test code editor. *Explyt - The AI test code editor*. URL: https://explyt.ai/en (дата звернення-: 28.11.2025).

39. Cucumber. *Cucumber*. URL: https://cucumber.io/ (дата звернення-: 28.11.2025).

40. Selenide: concise UI tests in Java. *Selenide: concise UI tests in Java*. URL: https://selenide.org/ (дата звернення-: 28.11.2025).

41. A Comprehensive Guide to the Applitools Eyes for Automated Visual Testing - NashTech Blog. *NashTech Blog*. URL: https://blog.nashtechglobal.com/a-comprehensive-guide-to-the-applitools-eyes-for-automated-visual-testing/ (дата звернення-: 08.12.2025).

42. Applitools: My Deep Dive into AI-Powered Visual Testing. *Skypage*. URL: https://skywork.ai/skypage/en/Applitools-My-Deep-Dive-into-AI-Powered-Visual-Testing/1976162741829431296 (дата звернення: 09.12.2025).

43. Explyt - IntelliJ IDEs Plugin | Marketplace. *JetBrains Marketplace*. URL: https://plugins.jetbrains.com/plugin/27979-explyt (дата звернення-: 08.12.2025).

44. Explyt Spring Debugger. *Все публикации подряд на Хабре – LiveJournal*. URL: https://habr-all.livejournal.com/17516629.html (дата звернення: 09.12.2025).

45.Ручне та автоматизоване тестування - QALight. *QALight*. URL: https://qalight.ua/baza-znaniy/ruchne-ta-avtomatizovane-testuvannya/ (дата звернення: 10.12.2025).

ДОДАТКИ

Додаток А. Схеми моделей розробки програмного забезпечення, сценарії та піраміда тестування

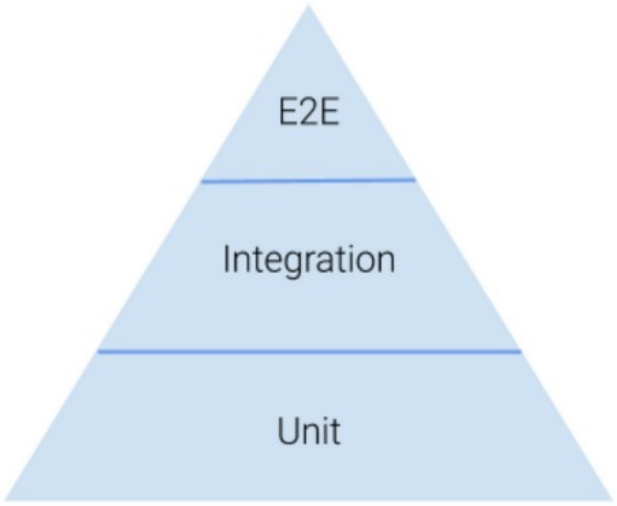

Рис. А.1 - Піраміда тестування

Рис. А.2 - Водоспадна модель розробки програмного забезпечення

Рис. А.3 - Водоспадна модель розробки програмного забезпечення

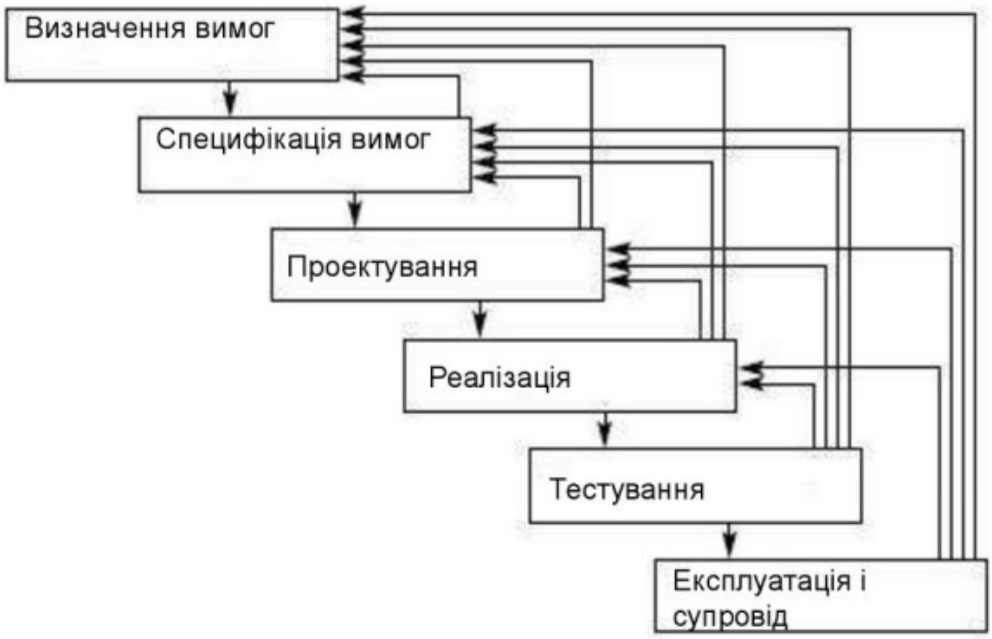

Рис. А.4 - Водоспадна модель розробки програмного забезпечення

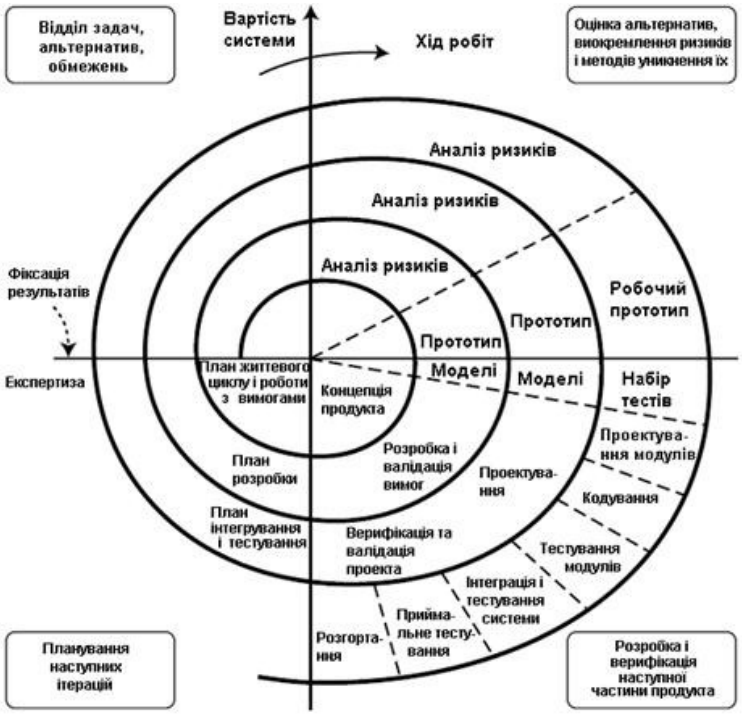

Рис. А.5 - Спіральна модель розробки програмного забезпечення

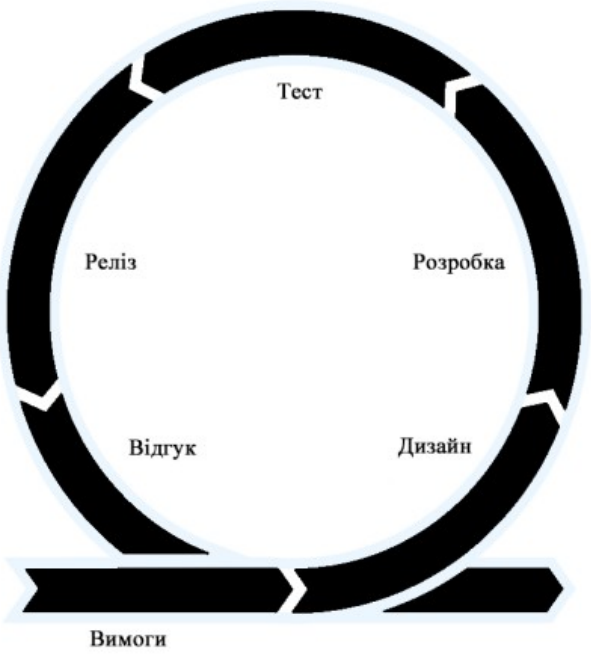

Рис. А.6 - Agile модель розробки програмного забезпечення

Рис. А.7 - Класи еквівалентності для значень тарифів на доставку

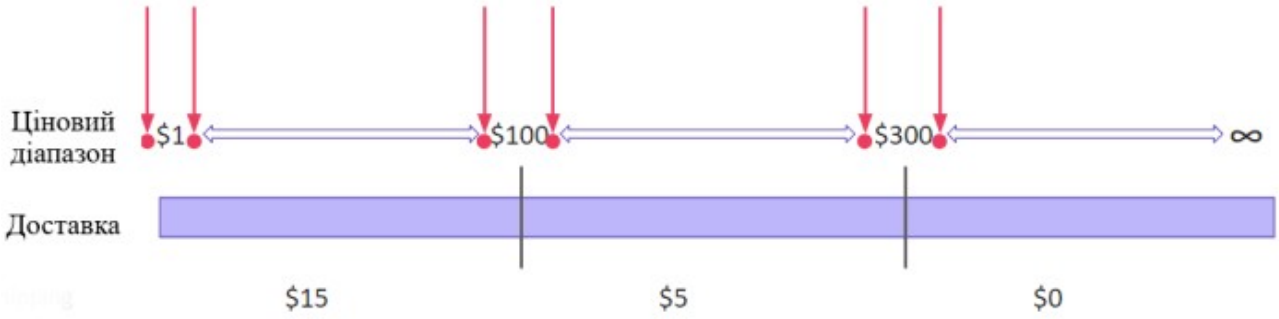

Рис. А.8 - Граничні значення для тарифів на доставку

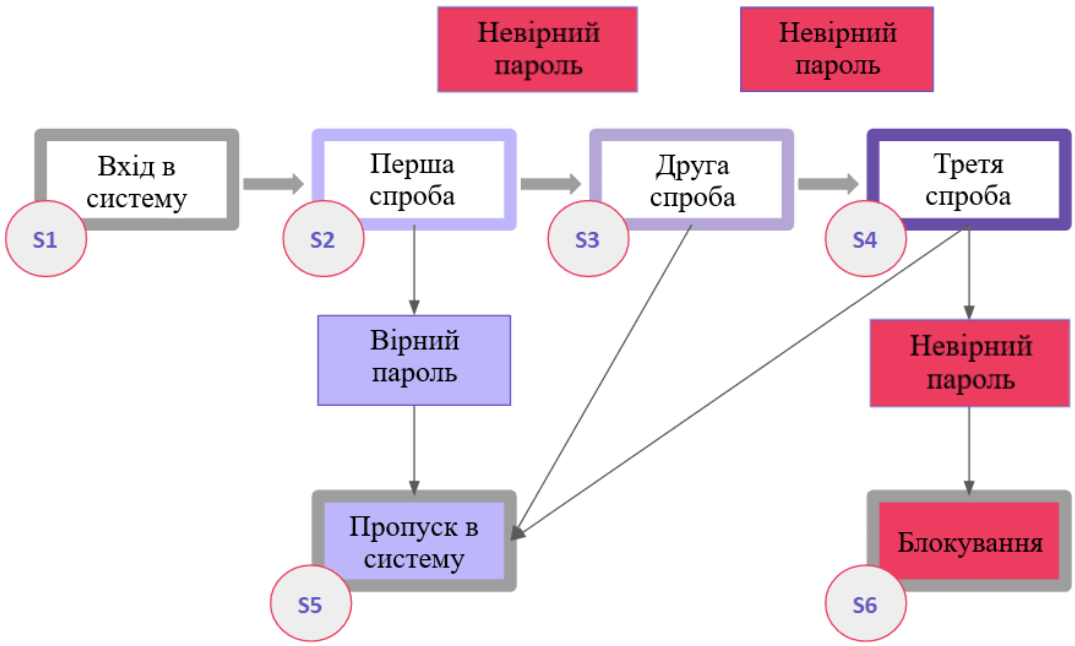

Рис. А.9 - Діаграма техніки переходу станів для входу в обліковий запис

Таблиця А.10 - Сценарії за технікою попарного тестування для продажу в пекарнях

Замовлення	**Розмір**	**Місто**	**Кількість**	**Доставка**	**Час**
Яблучний пиріг	Великий	Місто В	3	Адресна	Зараз
Чізкейк	Великий	Місто Б	1	Самовивіз	Зараз
Чізкейк	Середній	Місто А	2	Адресна	За графіком
Чізкейк	Великий	Місто Б	3	Самовивіз	Зараз
Яблучний пиріг	Великий	Місто А	3	Адресна	За графіком
Яблучний пиріг	Середній	Місто Б	1	Адресна	Зараз
Яблучний пиріг	Великий	Місто В	2	Самовивіз	За графіком
Яблучний пиріг	Середній	Місто А	1	Адресна	Зараз
Чізкейк	Великий	Місто Б	2	Адресна	Зараз
Чізкейк	Середній	Місто В	1	Самовивіз	За графіком
Чізкейк	Великий	Місто А	2	Адресна	Зараз
Чізкейк	Середній	Місто Б	3	Адресна	За графіком
Чізкейк	Великий	Місто А	1	Самовивіз	Зараз
Яблучний пиріг	Середній	Місто А	3	Самовивіз	Зараз
Яблучний пиріг	Великий	Місто Б	1	Адресна	За графіком
Яблучний пиріг	Середній	Місто А	2	Самовивіз	Зараз
Яблучний пиріг	Великий	Місто В	2	Адресна	Зараз

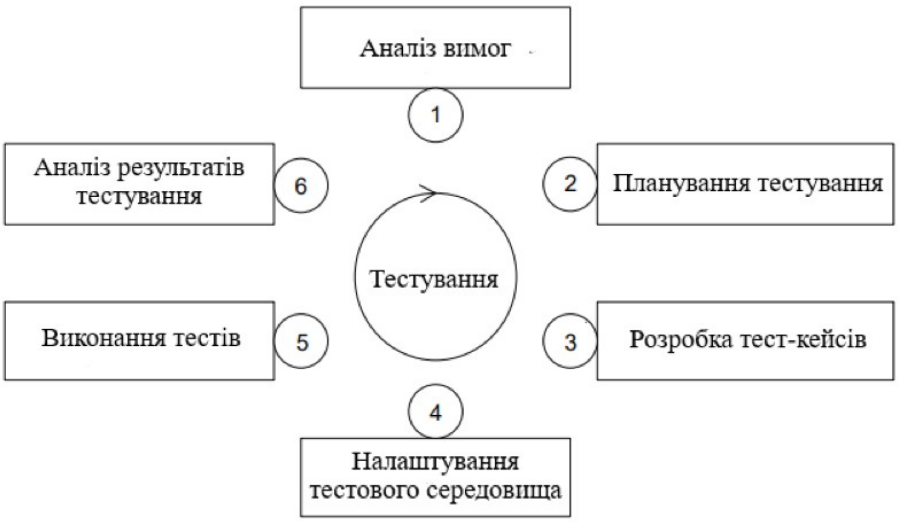

Рис. А.11 - Життєвий цикл тестування

Додаток Б. Блок-схема алгоритму

Рис. Б.1 - Блок-схема алгоритму створення автоматизованих тестів на основі AI

Додаток В. Селектори для веб-елементів, таблиці і графіки для порівняння підходів тестування

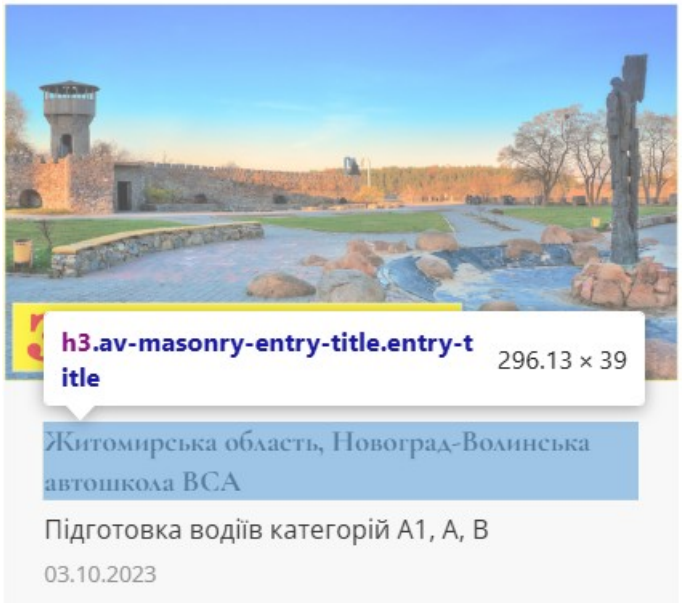

Рис. В.1 - Селектор для назви автошколи без змін

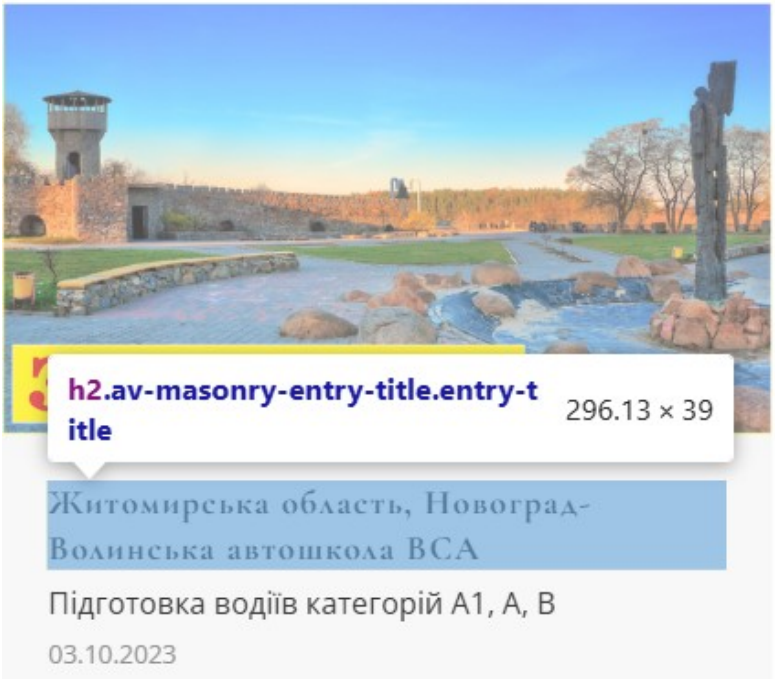

Рис. В.2 - Модифікований селектор для назви автошколи

Таблиця В.3 - Матриця помилок виявлених дефектів для сторінки вибору автошколи

	Баг існує	**Баг відсутній**
AI позначив баг	4	1
AI не позначив баг	0	12

Таблиця В.4 - Дані для порівняння тестування без AI і тестування з AI

	Тестові випадки без AI	**Тестові випадки з AI**
Час на створення 17 тестів для сторінки вибору автошколи	12 години	3 годин
Тестове покриття	50%	75%
Кількість невдалих тестів	1	5
Кількість виявлених дефектів	3	4

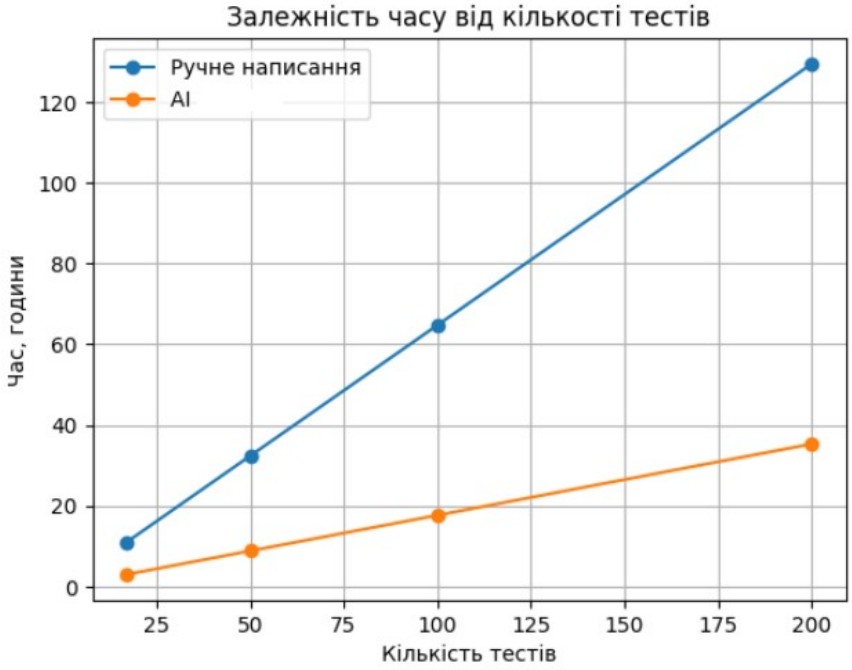

Рис. В.5 - Лінійна екстраполяція часу створення більшої кількості тестових випадків