Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет __Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти ____другий (магістерський)_____
(рівень вищої освіти)

Моделі даних автоматизованого тестування користувацького інтерфейсу
_____веб-сайтів_____.
(тема)

Виконав: студент 2 курсу, групи _СКСм-22-1

_Томачинська В. С._____
(прізвище, ініціали)

Спеціальність 123 – Комп'ютерна інженерія
(код і повна назва спеціальності)

Тип програми ___освітньо-професійна____
(освітньо-професійна або освітньо-наукова)

Освітня програма _____Спеціалізовані___
комп'ютерні системи_____
(повна назва освітньої програми)

Керівник _доцент Рожнова Т.Г._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ____ _____ ____Чумаченко С.В.__
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління

Кафедра Автоматизації проектування обчислювальної техніки

Рівень вищої освіти другий (магістерський)

Спеціальність 123 Комп'ютерна інженерія
(шифр і назва)

Тип програми Освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Спеціалізовані комп'ютерні системи
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. Кафедри АПОТ  (підпис)

«_____» _____ 2023 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Томачинській Валерії Сергіївні
(прізвище, ім'я, по батькові)

1. Тема роботи Моделі даних автоматизованого тестування користувацького інтерфейсу веб-сайтів

затверджена наказом по університету від 03 листопада 2023 р. № 1282 Ст.

2. Термін подання студентом роботи до екзаменаційної комісії 19 січня 2024 р.

3. Вихідні дані до роботи

Методики розробки: Behaviour-Driven Development (BDD), Об'єктно-орієнтоване програмування (ООП).

Програмні середовища та мови програмування: Java, Selenium, TestNG, Maven, TestNG.

4. Перелік питань, що потрібно опрацювати в роботі

Огляд та аналіз сучасних методів тестування програмного забезпечення

Розгляд застосування штучного інтелекту в процесі тестування

Розробка програмного інструменту для автоматизованого тестування

Інтеграція сучасних технологій та методів у процес тестування

Синтез та оптимізація методів тестування

Порівняльний аналіз ефективності традиційних та інноваційних методів тестування

Експериментальне дослідження та аналіз результатів

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) _____
слайди презентацій___14 слайдів.pptx_____
_____
_____
_____
_____
_____

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

7. Дата видачі завдання ___20 жовтня_2023р._____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Видача теми роботи, узгодження і затвердження теми	20.10.2023-25.10.2023	
2	Аналіз проблемної галузі, постановка задачі	25.10.2023-31.10.2023	
3	Вибір інструментальних засобів	01.11.2023-10.11.2023	
4	Аналіз сучасних методів тестування та застосування штучного інтелекту	11.11.2023-20.11.2023	
5	Розробка програмного інструменту для тестування	21.11.2023-30.11.2023	
6	Проведення експериментів та аналіз результатів	01.12.2023-05.12.2023	
7	Оформлення пояснювальної записки	06.12.2023-30.12.2023	
8	Перевірка виконаного проекту керівником, допуск до захисту	02.01.2024-26.01.2024	

Студент _____
(підпис)

Керівник роботи _____ доцент Рожнова Т.Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить 61 сторінку, 9 рисунків, 2 таблиці, 20 джерел за переліком посилань.

АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, GUI, МЕТОДИ ТЕСТУВАННЯ, ШТУЧНИЙ ІНТЕЛЕКТ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, ЕФЕКТИВНІСТЬ ТЕСТУВАННЯ, ДЕФЕКТ, АЛГОРИТМ, ІНТЕГРАЦІЯ

У цій кваліфікаційній роботі проведено дослідження методів автоматизованого тестування графічного користувацького інтерфейсу, з акцентом на ефективність, швидкість та точність виявлення дефектів у спеціалізованих комп'ютерних системах.

Об’єкт дослідження – процес тестування користувацького інтерфейсу веб-сайтів.

Мета роботи – дослідження та вдосконалення ефективних методів тестування користувацького інтерфейсу, з урахуванням їх застосування та можливостей оптимізації.

Наукова новизна: розробка нейронної мережі, інтегрованої у процес тестування, використовуючи глибоке навчання для класифікації макетів, що підвищує точність і зменшує потребу в ручному аналізі.

Практична значимість: завдяки аналізу різних підходів до тренування нейронних мереж та використанню навчальної бази даних, була створена оптимальна архітектура мережі для виявлення дефектів у користувацькому інтерфейсі веб-сайтів, забезпечуючи високу ефективність.

Результати дослідження спрямовані на підвищення якості та ефективності процесів тестування комп'ютерних систем. Робота містить аналіз, розробку програмного інструменту для тестування, інтеграцію сучасних технологій та методів.

ABSTRACT

The explanatory note contains 61 pages, 9 figures, 2 tables, and 20 references from the list of sources.

AUTOMATED TESTING, GUI, TESTING METHODS, ARTIFICIAL INTELLIGENCE, CONVOLUTIONAL NEURAL NETWORKS, TESTING EFFICIENCY, DEFECT, ALGORITHM, INTEGRATION.

This qualification work conducts a study of methods for automated testing of graphical user interfaces, with an emphasis on efficiency, speed, and accuracy in identifying defects in specialized computer systems.

Research Object – the process of testing the user interface of websites.

Objective of the Study – investigating and enhancing effective methods of user interface testing, considering their application and potential for optimization.

Scientific Novelty: development of a neural network integrated into the testing process, utilizing deep learning for the classification of layouts, which increases accuracy and reduces the need for manual analysis.

Practical Significance: through the analysis of various approaches to training neural networks and the use of a training database, an optimal network architecture has been developed for detecting defects in website user interfaces, ensuring high efficiency.

Research Outcomes – aimed at improving the quality and efficiency of computer system testing processes. The work includes analysis, development of a software tool for testing, and integration of modern technologies and methods.

The results of the study are aimed at improving the quality and efficiency of computer systems development processes. The work includes analysis, development of a software tool for testing, and integration of modern technologies and methods.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Програмний інтерфейс додатка (англ. Application Programming Interface);

BDD – Розробка з орієнтацією на поведінку (англ. Behavior Driven Development);

CI – Постійна інтеграція (англ. Continuous Integration);

DevOps – Розробка та експлуатація (англ. Development and Operations);

ECP – Поділ на класи еквівалентностей (англ. Equivalence Class Partitioning);

GUI – Графічний інтерфейс користувача (англ. Graphical User Interface);

IDE – Інтегроване середовище розробки (англ. Integrated Development Environment);

ML – машинне навчання (англ. Machine Learning);

NLP – обробка природної мови (англ. Natural Language Processing);

ООР – Об'єктно-орієнтоване програмування (англ. Object-Oriented Programming);

POM – Модель сторінки (англ. Page Object Model);

TDD – Розробка з тестуванням на основі тестів (англ. Test Driven Development):

ШІ – штучний інтелект.

ВСТУП

В сучасному інформаційному суспільстві важливим елементом успішної реалізації програмного забезпечення є якість та ефективність користувацького інтерфейсу (UI) веб-сайтів.

З ростом складності веб-сайтів з'являється потреба в тестуванні їх користувацького інтерфейсу для переконання в правильному функціонуванні та наданні безперебійного досвіду користувачам. Якщо веб-сайт має недоліки чи помилки, це може призвести до незадоволеності користувачів, втрати клієнтів та збитків для бізнесу. Тому забезпечення високої якості веб-сайтів є ключовим завданням для забезпечення задоволення та лояльності користувачів, а також для підтримання конкурентоспроможності в сучасному віртуальному просторі.

Метою даного наукового дослідження є розгляд питань та пропозиції щодо вдосконалення методів тестування та розробка ефективних та зручних методів тестування UI веб-сайтів, з урахуванням складності їхнього застосування та можливостей вдосконалення.

Перспективи розвитку включають аналіз інноваційних методів штучного інтелекту разом із традиційними техніками тестового дизайну з метою оптимізації якості та ефективності тестування UI. Експериментальний аналіз дозволить отримати об'єктивні дані щодо ефективності різних методів та визначити їхні переваги та обмеження в контексті тестування UI веб-сайтів. Такий підхід дозволить спростити та поліпшити процес тестування, що, в свою чергу, призведе до підвищення якості програмного забезпечення та задоволеності користувачів.

Об’єктом дослідження є процеси тестування користувацького інтерфейсу, виявлення та виправлення помилок, а також покращення загального користувацького досвіду.

Предметом дослідження є аналіз та порівняння різних технік та методів автоматизованого тестування UI, оцінка складності існуючих підходів та методів тестування користувацького інтерфейсу (UI) веб-сайтів. Особлива увага приділяється вдосконаленню існуючих методів тестування програмного забезпечення, а саме застосуванню штучного інтелекту в контексті тестування.

Основні виклики включають у себе аналіз розмірів та складності взаємозв'язків між елементами UI, а також відсутність оптимального балансу між абстракцією та реалізацією в тестових сценаріях. Додатковою проблемою є виразність підходів, включаючи їх легкість в розумінні, вивченні та ефективному використанні.

Кваліфікаційна робота буде зосереджена на інноваційних підходах до тестування UI, об'єднуючи в собі практики автоматизованого та штучного інтелектуального тестування. В результаті дослідження очікується формування рекомендацій та практичних висновків, спрямованих на підвищення ефективності та точності тестування UI веб-сайтів.

Так як від вибору виду та інструменту буде залежати ефективність тестування, то важливо визначити оптимальний підхід та інструменти для процесу тестування. У цій роботі буде вирішено проблему виявлення оптимального виду та інструменту для тестування інтерфейсу користувача на відповідність вимогам, враховуючи факт, що у кожного програмного забезпечення своя специфіка, не існує універсального виду тестування для кожного із рішень.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка задачі дослідження

В рамках кваліфікаційної роботи ставиться завдання розробки та аналізу методів автоматизованого тестування GUI, з особливим фокусом на ефективності, швидкості виконання та точності виявлення дефектів у спеціалізованих комп'ютерних системах.

Основні завдання дослідження:

– глибоке дослідження різноманітних підходів до автоматизованого тестування GUI, включаючи аналіз граничних значень, еквівалентного розбиття, причинно-наслідкових зв'язків та застосування технологій штучного інтелекту;

– порівняльний аналіз різних технік тестування з погляду їхньої здатності ефективно виявляти помилки, швидкості обробки та точності результатів;

– інструменту, який забезпечує автоматизацію тестування, який інтегрує вивчені методики, з метою поліпшення процесів виявлення та виправлення помилок в програмному забезпеченні.

Очікувальні результати:

– розробка ефективних методів тестування, які забезпечують високу точність виявлення дефектів, зменшують час тестування та оптимізують загальний процес розробки;

– поглиблення знань у сфері автоматизованого тестування та внесення важливого вкладу в наукові дослідження, пов'язані з розробкою спеціалізованих комп'ютерних систем;

– можливість використання розробленого програмного продукту як засобу для підвищення якості програмного забезпечення в реальних проектах, що включають складні системи з графічними інтерфейсами.

1.2 Визначення поняття тестування та основних його етапів

Тестування програмного забезпечення є ключовим процесом у забезпеченні його якості та надійності. Цей процес включає не тільки безпосереднє виконання тестів, але й ряд інших діяльностей, таких як планування, аналіз, проектування та створення звітів. Тестування поділяється на динамічне, яке передбачає виконання тестованого компонента або системи, і статичне, яке охоплює рецензування робочих продуктів, таких як вимоги та вихідний код.

Тестування не тільки перевіряє відповідність системи встановленим вимогам, але й забезпечує її відповідність очікуванням користувачів та інших зацікавлених сторін. Це включає оцінку якості робочих продуктів, перевірку виконання вимог, створення впевненості в якості об'єкта тестування, виявлення та запобігання дефектів, а також зниження ризиків.

Модель життєвого циклу проекту (Software life cycle model, SLCM) – це структура, що визначає послідовність виконання та взаємозв’язку процесів, дій та задач на протязі виконання проекту [1].

Життєвий цикл розробки програмного забезпечення визначає етапи та активності, які виконуються під час розробки ПО. Існують різні моделі життєвого циклу, включаючи послідовні, ітеративні, інкрементальні та інші. Кожна модель має свої особливості та підходи до тестування.

Послідовні моделі, такі як модель "Водоспаду" і V-модель, використовують строгий послідовний підхід, де кожен етап починається після завершення попереднього. Інкрементальні та ітеративні моделі, такі як RUP, Скрам та Канбан, дозволяють більш гнучке управління процесом розробки, з можливістю ранньої доставки продукту та постійним оновленням функціональності (рисунок 1.1).

Основна модель життєвого циклу розробки програмного забезпечення, яка часто використовується – це модель "Водоспад". Ця модель має чітку послідовність фаз, де кожна наступна фаза починається лише після

завершення попередньої. Основні фази моделі "Водоспад" включають:

– на стадії аналізу вимог визначаються вимоги до продукту, які повинні бути чітко задокументовані;

– розробляються архітектура та дизайн продукту, що відповідають визначеним вимогам;

– на етапі кодування відбувається безпосереднє програмування та створення програмного коду;

– після розробки продукту відбувається його тестування на виявлення та виправлення помилок;

– готовий продукт впроваджується в експлуатацію;

– відбувається підтримка продукту, його оновлення та виправлення помилок, які можуть виявитися під час експлуатації.

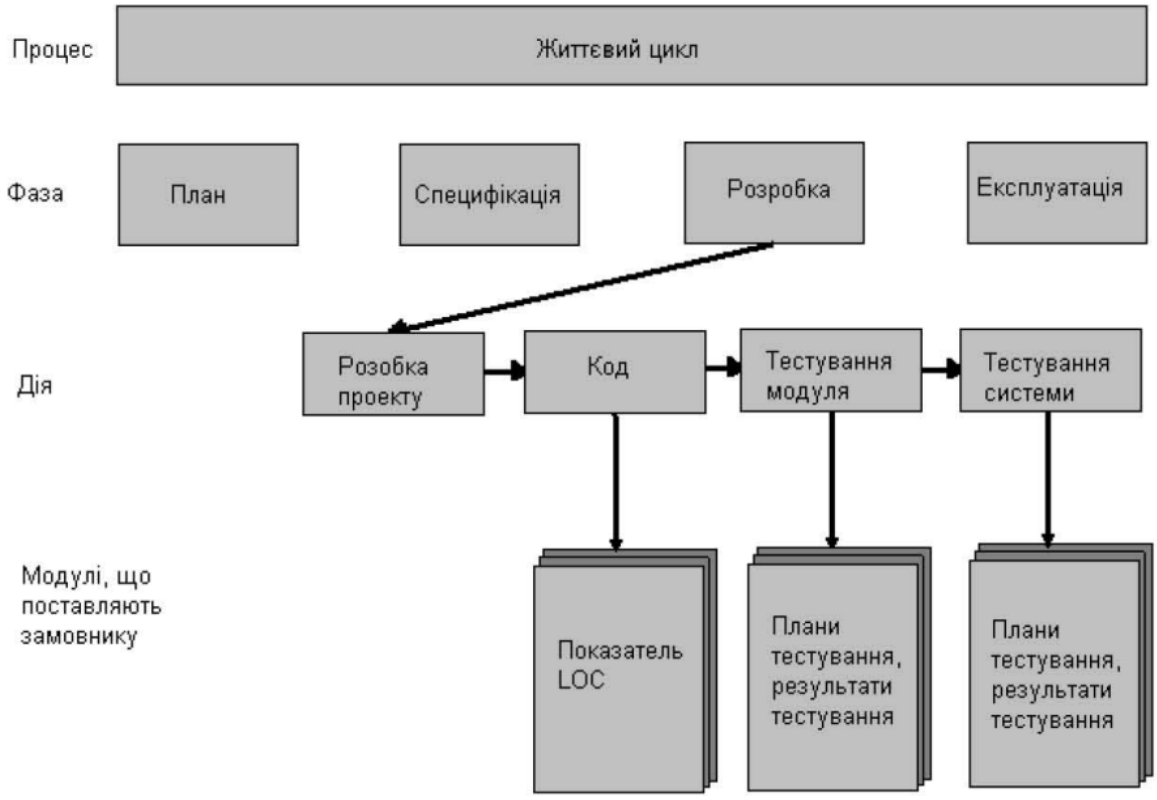

Рисунок 1.1 – Узагальнена модель життєвого циклу

Вибір моделі життєвого циклу залежить від особливостей проекту, включаючи його цілі, тип продукту, бізнес-пріоритети та ризики. Важливо

адаптувати модель життєвого циклу до конкретного проекту, враховуючи його унікальні характеристики і вимоги.

Підвищення якості програмного забезпечення є невід'ємною частиною розробки програмних продуктів, оскільки воно визначає їх ефективність, надійність та задоволення потреб користувачів. Протягом останніх років існує безліч методів, практик та підходів, які допомагають розробникам досягти високої якості програмного забезпечення. В цьому аналізі ми розглянемо деякі з них. Перш за все, одним з ключових факторів у підвищенні якості програмного забезпечення є використання методологій розробки програмного забезпечення. Agile-методології, такі як Scrum та Kanban, надають гнучкий підхід до розробки, де комунікація зі замовником та постійне вдосконалення продукту стають основою роботи. Це дозволяє швидко реагувати на зміни вимог та забезпечувати високу якість програмного забезпечення. Крім того, практики DevOps стають все більш поширеними для підвищення якості програмного забезпечення. Цей підхід поєднує розробку (Development) та операції (Operations) в один процес, щоб забезпечити спільну відповідальність за весь цикл життя програмного продукту, включаючи розробку, тестування, впровадження та підтримку. Основна мета DevOps в тестуванні полягає у поліпшенні співпраці та комунікації між розробниками програмного забезпечення, тестувальниками та операторами системи. Такий підхід до поєднання розробки програмного забезпечення та операцій може скоротити час виходу на ринок [2]. Одним з ключових аспектів DevOps в тестуванні є автоматизація процесів. Це означає, що тестування повинно бути інтегровано в процес розробки, а тести автоматизовані та виконуються на кожному етапі розробки. Це дозволяє швидко виявляти помилки та недоліки програмного забезпечення і вносити необхідні зміни.

Тестування є невід'ємною складовою підвищення якості програмного забезпечення. Функціональне тестування перевіряє, чи відповідає програмне забезпечення вимогам та специфікаціям. Це гарантує, що програма працює вірно та виконує очікувані функції. Навантажувальне тестування дозволяє

оцінити продуктивність програмного забезпечення під навантаженням, виявляючи проблеми з продуктивністю та масштабованістю. Автоматизоване тестування, в свою чергу, використовує автоматизовані інструменти для прискорення процесу тестування та зниження кількості помилок.

Існує багато інших методів підвищення якості програмного забезпечення, таких як контроль якості коду, використання шаблонів та бібліотек, валідація даних, безперервна інтеграція та безперервна доставка. Контроль якості коду включає в себе перевірку стандартів оформлення, виявлення потенційних помилок та забезпечення читабельності коду. Використання шаблонів та бібліотек дозволяє зменшити кількість дублювання коду та покращити його структуру.

Валідація даних забезпечує відповідність вхідних даних вимогам та запобігає помилковому введенню. Безперервна інтеграція та безперервна доставка (CI/CD) забезпечують автоматичну збірку, тестування та розгортання програмного забезпечення, що сприяє швидкій відгукності та зменшенню ризиків. CI/CD – надійний інструмент для більш надійного та частого розгортання [3]. Використання агрегаторів помилок та моніторингу продуктивності допомагає виявляти та вирішувати проблеми, що виникають у процесі роботи програмного забезпечення.

Узагальнюючи, підвищення якості програмного забезпечення вимагає комплексного підходу та використання різноманітних методів і практик. Agile та DevOps надають гнучкість та швидкість, тестування допомагає перевірити коректність та продуктивність, контроль якості коду та використання шаблонів поліпшують структуру та читабельність, а безперервна інтеграція та безперервна доставка забезпечують автоматизацію та швидкість постачання програмного забезпечення. Використання цих рішень може значно підвищити якість програмного забезпечення і задовольнення користувачів. Додатково, важливо також враховувати контекст проекту та його специфічні вимоги для вибору найбільш ефективних підходів у підвищенні якості програмного забезпечення.

1.3 Ефективність автоматизованого тестування

Підвищення якості програмного забезпечення є ключовим завданням для розробників, оскільки воно визначає ефективність та надійність програмних продуктів. Недоліки та помилки в роботі веб-сайтів можуть мати серйозні наслідки, включаючи втрату довіри користувачів, зменшення конкурентоспроможності та негативний вплив на репутацію компанії. Тому важливо розробляти та використовувати ефективні методи тестування, які допоможуть виявляти та усувати дефекти в користувацькому інтерфейсі веб-сайтів.

Проте, виконання ручних тестів користувацького інтерфейсу веб-сайту може бути витратним у часі та працездатності. Такі тести вимагають багато повторних дій та можуть бути підвержені людським помилкам. Крім того, зі зростанням розміру та складності веб-сайтів, масштабування ручного тестування стає викликом. Великі проекти можуть мати сотні або навіть тисячі сторінок та функціональних можливостей, і вручну перевірити кожну з них стає майже неможливою задачею.

Тому, для ефективного та швидкого тестування веб-сайтів виникає потреба у розробці фреймворку для автоматизованого тестування, який допоможе зменшити зусилля та покращити якість процесу.

Автоматизоване тестування дозволяє швидко виконувати велику кількість тестів, повторювати їх у будь-який момент часу та забезпечувати повну покриття функцій веб-сайту. Автоматизація необхідна для створення “страхування”, отримання суттєвого відгуку, зведення до мінімуму рівня технічного боргу та допомоги в управлінні розробки [4].

Автоматизоване тестування ПЗ – це процес верифікації програмного забезпечення, при якому основні функції та кроки тесту, такі як запуск, ініціалізація, виконання, аналіз та видача результату, виконуються автоматично за допомогою інструментів для автоматизованого тестування

Цей тип тестування допомагає автоматизувати завдання, що часто повторюються, але необхідні для максимізації тестового покриття.

Деякі завдання тестування, такі як низькорівневе регресійне тестування, можуть бути трудомісткими і вимагають багато часу, якщо виконувати їх вручну. Крім того, мануальне тестування може недостатньо ефективно знаходити деякі класи помилок. У таких випадках автоматизація може допомогти заощадити час та зусилля проектної команди.

Після створення автоматизованих тестів їх можна в будь-який момент запустити знову, причому запускаються і виконуються вони швидко і точно. Таким чином, якщо є необхідність частого повторного прогону тестів, значення автоматизації для спрощення супроводу проекту та зниження його вартості важко переоцінити. Адже, навіть, мінімальні патчі та зміни коду можуть стати причиною появи нових багів.

Тестувати програму можна на різних рівнях: код (юніт-тести та інтеграційне тестування), API та GUI. Різні види тестів краще підходять у різних ситуаціях. Розглянемо докладніше види автоматизованого тестування.

Юніт-тести – це тестування одного модуля коду (зазвичай це одна функція або один клас у разі ООП-коду) в ізольованому оточенні. Це означає, що якщо використовуються якісь сторонні класи, то замість них підставляються класи-заглушки (моки та стаби), також, код не повинен працювати з мережею (і зовнішніми серверами), файлами, базою даних (інакше ми тестуємо не саму функцію чи клас, а ще й диск, базу тощо).

API (інтерфейс прикладного програмування) – це тип програмного інтерфейсу, який дозволяє двом чи більше комп’ютерним програмам спілкуватися одна з одною. Він пропонує послуги іншим програмним забезпеченням. Іншими словами, API дає нам можливість використовувати чужі розробки для власних цілей. Наприклад, він може використовувати стандартні функції для малювання інтерфейсу. Сучасні API часто приймають форму веб-служб, які надають користувачам певну інформацію. Зазвичай

процедура обміну інформацією та формат передачі даних побудовані так, що обидві сторони знають, як взаємодіяти одна з одною.

Під час тестування API можна використовувати такі загальноприйняті прийоми, як аналіз граничних значень і розбиття на класи еквівалентності. У запитах API значення параметрів можна передавати явно. Це чудова можливість виділити межі вхідних і вихідних значень і перевірити їх. Також важливо оцінити API з точки зору зручності його використання іншими продуктами і з точки зору легкої інтеграції з ним.

Графічний інтерфейс користувача (GUI), комп'ютерна програма, яка дозволяє людині спілкуватися з комп'ютером за допомогою символів, візуальних метафор і вказівних пристроїв [5]. Це найскладніший для тестування вид. Якщо йдеться про перевірку роботи сайту, то ми повинні емулювати роботу браузера, який досить складно влаштований, аналізувати інформацію, що виводиться на сторінці. Але цей вид тестування дуже важливий, оскільки він взаємодіє з додатком так само, як і користувач. GUI тести ще називають End-to-End (E2E) або приймальні (aceptance) тести.

Зробимо проміжний висновок про те, що потрібно автоматизувати:

– важкодоступні місця у системі (backend-процеси, логування файлів, запис у БД);

– автоматизувавши перевірку критичної функціональності, можна гарантувати швидке перебування помилок, отже, і швидке їх вирішення;

– рутинні операції, такі як перебори даних (форми з великим кількістю введених полів);

– автоматизувати заповнення полів некоректними даними та перевірку на появу тієї чи іншої валідації;

– довгі End-to-End сценарії;

– перевірка даних, які потребують точних математичних розрахунків;

– перевірка правильності пошуку даних.

Метрики в тестуванні можна розділити на дві групи: зовнішні та внутрішні. Зовнішні метрики – це ті, що використовуються для вимірювання

впливу на інші види діяльності (зокрема активності з тестування). Внутрішні метрики – це ті, що використовуються для вимірювання ефективності та дієвості у виконанні її завдань.

Вимірювані показники зазвичай включають такі:

– переваги автоматизації;

– витрати створення автоматизованих тестів;

– витрати аналіз інцидентів автоматизованого тестування;

– витрати на підтримку автоматизованих тестів;

– відношення фейлів до фактичних дефектів;

– час виконання автоматизованих тестів;

– кількість автоматизованих тесткейсів;

– кількість pass та fail;

– кількість хибнопозитивних та хибнонегативних результатів;

– покриття коду;

– метрики якості коду;

– щільність дефектів у коді автоматизації;

– швидкість та ефективність компонентів.

Особливо важливо вимірювати та повідомляти про переваги автоматизації. Наприклад, це може бути економія часу або сил, збільшення обсягу тестування (широта або глибина охоплення або частота виконання) або будь-яка інша перевага, таке як регулярне повторення тестів, краще використання ресурсів або менша кількість мануальних помилок.

Можливі метрики включають:

– кількість зекономлених годин ручного тестування;

– скорочення часу виконання регресійного тестування;

– досягнута кількість додаткових циклів тестування;

– кількість або відсоток виконаних додаткових тестів;

– збільшення покриття (вимоги, функціональність, структура);

– кількість дефектів, виявлених завдяки автоматизації, які не були виявлені при ручному тестуванні (наприклад, дефекти надійності).

Автоматизація тестування зазвичай заощаджує витрати на мануальне тестування. Заощаджені ресурси можна присвятити іншим видам тестування (наприклад, дослідницького тестування). Дефекти, виявлені цими додатковими тестами, можна розглядати як непрямі переваги, оскільки автоматизація тестування дозволила виконати їх.

Зусилля з автоматизації тестів – одна з ключових статей витрат, пов'язаних із автоматизацією тестування. Часто для цього потрібно більше ресурсів, ніж для виконання того ж таки тесту вручну, і це може бути перешкодою для використання автоматизації в проекті. Хоча вартість реалізації конкретного автоматизованого тесту буде значною мірою залежати від самого тесту, інші фактори, такі як підхід до написання сценаріїв, рівень володіння інструментом тестування, середовище і рівень навичок інженера з автоматизації тестування, також будуть мати значення. Так само надзвичайно важлива "зрілість" тестового фреймворку – трудовитрати на написання нових тестів (а тим більше подібних до існуючих) помітно знижується з розвитком. У деяких випадках автоматизувати тести набагато швидше, ніж проходити їх мануально, особливо у випадку параметризованих тестів. Наприклад, при реалізації data-driven тесту, сам тест достатньо реалізувати один раз і він буде виконуватися для кожного сету даних, коли при мануальному проходженні доведеться виконувати всі дії для кожного сету.

Поширена проблема автоматизованих тестів полягає в тому, що багато хто з них може впасти з однієї і тієї ж причини через один дефект у програмному забезпеченні. Хоча метою тестів є виявлення дефектів у програмному забезпеченні, проведення кількох тестів виявлення однієї й тієї ж дефекту марнотратно. Особливо актуально це для автоматизованого тестування, оскільки зусилля, необхідні аналізу кожного невдалого тесту, може бути значними. Вимірювання кількості автоматизованих тестів, які не пройдені для даного дефекту, допоможе зрозуміти, коли це вже стає проблемою.

Однією з найпростіших метрик для збору є час, необхідний виконання автоматизованих тестів. На початку реалізації це може бути не важливо, але зі збільшенням кількості автоматизованих тестів цей показник може стати дуже важливим.

1.4 Види тестування програмних продуктів

Тестування програмного забезпечення – це комплексний процес, який включає в себе різні типи тестувань, кожен з яких спрямований на оцінку певних характеристик системи або її частини.

Функціональне тестування – це тип тестування перевіряє, чи виконує система визначені функції відповідно до її вимог і специфікацій. Це може включати перевірку коректності виконання команд, обробку даних, інтеграцію з іншими частинами системи та взаємодію з інтерфейсами. Функціональне тестування зазвичай ґрунтується на зовнішньому описі роботи програми.

В нефункціональному тестуванні фокус зміщується з того, що робить система, на те, як вона це робить. Цей тип тестування оцінює якість програмного продукту, включаючи аспекти, такі як надійність, продуктивність, безпека, сумісність та зручність користування.

Нефункціональне тестування важливе для визначення того, наскільки добре програмне забезпечення впорається з різними навантаженнями, а також для оцінки його стійкості до помилок і зовнішніх загроз.

Тестування білим ящиком (структурне) тестування ґрунтується на внутрішній логіці та структурі програмного коду. Це включає аналіз потоків даних, алгоритмів, внутрішніх даних, зовнішніх інтерфейсів та іншої внутрішньої роботи програми. Мета тестування білим ящиком - забезпечити, що внутрішні операції виконуються відповідно до визначених специфікацій і що вони правильно інтегровані.

Тестування, пов'язане зі змінами, це включає в себе підтверджуюче тестування та регресивне тестування. Підтверджуюче тестування проводиться

для перевірки того, що виправлені помилки більше не виникають. Регресивне тестування забезпечує, що нові зміни, внесені в систему, не порушили існуючу функціональність та не внесли нових помилок [6].

Кожен із цих типів тестування може бути застосований на різних рівнях тестування – від компонентного (тестування окремих модулів) до системного (тестування всієї інтегрованої системи) та приймального (перевірка готовності продукту для використання кінцевими користувачами).

Ключовим аспектом ефективного процесу тестування є розуміння того, коли і які типи тестів найбільш доречні, а також забезпечення адекватного покриття тестами на всіх рівнях та для всіх типів тестування. Це дозволяє мінімізувати ризики і забезпечити високу якість кінцевого продукту.

1.5 Використання методів тестування користувального інтерфейсу

Тестові сценарії в атоматизованому тестуванні є документованою послідовністю кроків, які виконуються для перевірки певної функціональності або властивостей програмного забезпечення. Вони є основою для розробки та виконання автоматизованих тестів. Тестові сценарії описують необхідні дії, вхідні дані та очікувані результати, які тест повинен перевірити. Створення ефективних тестових сценаріїв вимагає докладної розробки, чіткого розуміння функціональних вимог та врахування потенційних сценаріїв використання програмного забезпечення. Це допомагає забезпечити якість тестування та ефективне виявлення помилок.

В ході роботи створення тестових сценаріїв, відбувається за допомогою концепції BDD (Behavior-Driven Development) – методології розробки програмного забезпечення, яка спрямована на співпрацю між розробниками, тестувальниками та бізнес-аналітиками. BDD дозволяє створювати зрозумілі та легко зчитувані тестові сценарії, які фокусуються на поведінці системи з точки зору бізнес-вимог.

У нашому випадку, тестовий сценарій описує дії, очікувані результати та перевірки для перевірки правильності роботи системи з точки зору вимог бізнесу. Застосування BDD дозволяє команді розробників та тестувальників зосередитися на функціональних можливостях системи та впевнитися, що вони виконують вимоги бізнесу.

Порівняно з BDD, TDD (Test-Driven Development) – це методологія розробки програмного забезпечення, яка базується на написанні тестів перед написанням функціональності. У TDD розробник спочатку пише тест, який описує очікувану поведінку коду, а потім реалізує функціонал для його проходження. TDD акцентується на покритті коду тестами та забезпеченні правильності роботи окремих функцій.

BDD та TDD мають багато спільного, але основна відмінність полягає в тому, що BDD зорієнтована на співпрацю між всіма сторонами проекту (розробниками, тестувальниками, бізнес-аналітиками), тоді як TDD більше фокусується на розробників та їхньому розумінні функціональності. BDD зазвичай використовує спеціальні інструменти, такі як Cucumber або JBehave, для створення зрозумілих та легко зчитуваних тестових сценаріїв.

Узагальнюючи, BDD та TDD є методологіями, які сприяють покращенню процесу розробки програмного забезпечення, забезпечуючи високу якість продукту та зниження витрат на виправлення помилок. Обидва підходи можуть використовуватися окремо або разом в залежності від вимог проекту та вподобань команди розробників.

1.5.1 Попарне тестування

Pairwise testing (попарне тестування) – це техніка формування наборів тестових даних з повного набору вхідних даних в системі, яка дозволяє істотно скоротити кількість тест-кейсів [7].

Розглянемо, наприклад, програму, яка приймає два вхідні параметри, кожен з яких може приймати чотири різних значення. Замість того, щоб

тестувати всі 16 можливих комбінацій, попарне тестування дозволяє перевірити лише ті комбінації, де кожна пара значень зустрічається хоча б раз. Це може звести кількість тестів до мінімуму, при цьому забезпечуючи ефективне покриття важливих сценаріїв взаємодії.

Наприклад, якщо ми маємо параметри A і B, кожен з яких може приймати значення 1, 2, 3 або 4, попарне тестування може включати такі комбінації: (A1, B1), (A2, B2), (A3, B3), (A4, B4), (A1, B2), (A2, B3), (A3, B4), (A4, B1) тощо. Кожна пара значень тут представлена хоча б раз, що забезпечує гарне покриття сценаріїв взаємодії.

Переваги попарного тестування полягають у зменшенні кількості необхідних тестів, зберігаючи при цьому високий рівень якості тестового покриття. Це дозволяє ефективно використовувати ресурси, особливо в умовах обмеженого часу та бюджету.

Однак, попарне тестування може не виявити деякі помилки, особливо в тих випадках, коли взаємодія відбувається між більш ніж двома параметрами. Тому цей метод необхідно комбінувати з іншими методами тестування для повного покриття потенційних помилок.

1.5.2 Еквівалентне розбиття

Equivalence Class Partitioning (ECP) – це метод тестування програмного забезпечення, орієнтований на створення кількох умов на основі отриманого тестового випадку. Тестовий приклад використовується для тестування системного модуля, який потрібно перевірити. Випробування проводяться для отримання результатів функціонального виконання роботи модуля. Результати, отримані за цією умовою, є дійсними чи недійсними значеннями. Вхідною умовою в ECP може бути числове значення, значення діапазону або логічна умова [8].

Розглянемо приклад поділу на класи еквівалентності при тестуванні поля «Age» з допустимим діапазоном значень введення від 1 до 100 в формі реєстрації (рисунок 1.2).

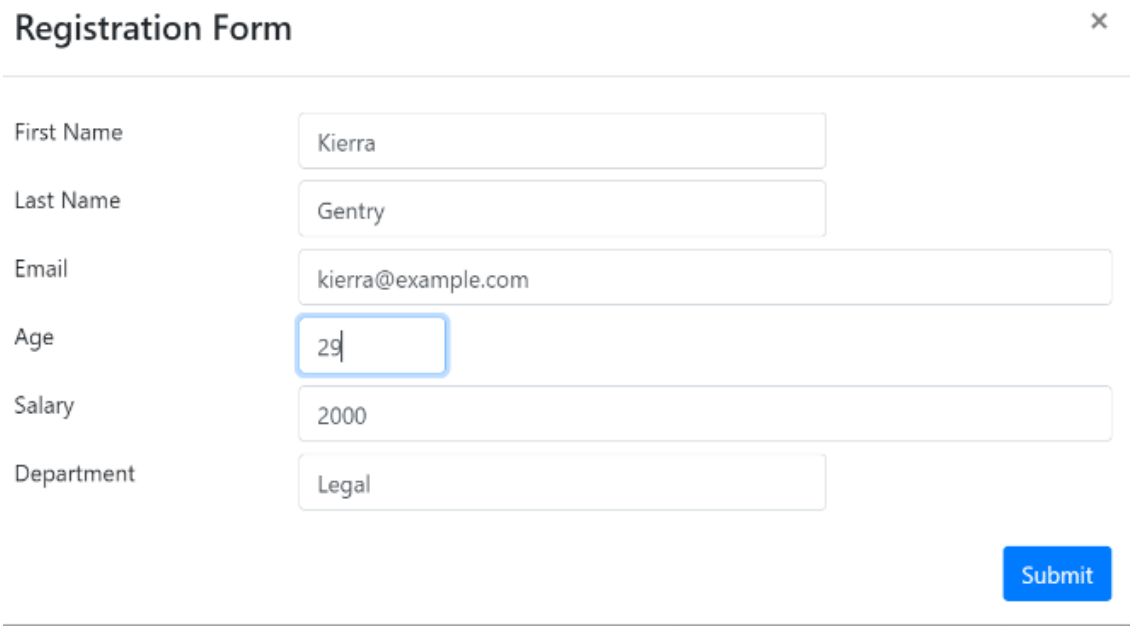

Рисунок 1.2 – Поле «Age» з допустимим діапазоном значень введення

Вводити весь діапазон – досить довгий процес. Тим більше, є ще неприпустимі значення (спецсимволи, негативні числа, букви і та ін.), введення яких потрібно перевірити.

Як говорилося вище, всі значення одного класу еквівалентності однаково впливають на систему, тобто допустимі значення система приймає, а недопустимі – ні.

Таким чином, можна виділити два класи еквівалентності: вибрати зі значень числа від 1 до 100 та неприпустимі значення (числа від -∞ до 0, від 1001 до + ∞, а також всі інші літери і символи).

Клас з неприпустимими значеннями можна розбити на кілька:

– від - ∞ до 0;

– від 101 до + ∞;

– спеціальні символи (# @ + - / _:; " 'і т.д.);

– літери.

В результаті, завдяки класам еквівалентності можна використовувати мінімум 5 тестів для тестування поля введення. Наприклад, в поле ввести наступні дані: 54, -17, 173, Ім'я, $ _ = #.

Техніку поділу на класи еквівалентності застосовують для скорочення числа тестів, при цьому зберігаючи прийнятне тестове покриття. Дана техніка підходить також для текстових або інших типів даних.

Кроки застосування техніки поділу на класи еквівалентності наступні:

1) визначити класи еквівалентності. Від правильності виконання даного кроку залежить ефективність майбутнього тестування;

2) обрати представника кожного класу;

3) виконати тести.

Техніка поділу на класи еквівалентності має свої плюси і мінуси. До переваг можна віднести можливість структурувати процес тестування і, отже, скоротити час на виконання тестування. До недоліків можна віднести пропуск багів при неправильному використанні техніки.

1.5.3 Аналіз граничних значень

Аналіз граничних значень – техніка тест-дизайну, яка спрямована на перевірку поведінки системи на граничних значеннях вхідних даних (кордонах класів еквівалентності) [9].

Дуже важливо перевіряти саме граничні значення, тому що досить часто виникають помилки саме на межах класів еквівалентності.

На кожній межі діапазону потрібно перевірити 3 значення: граничне значення, значення до і після межі.

На прикладі поля «Age» для діапазону значень, що вводяться всі значення від 1 до 100 приведуть до одного і того ж результату, то є двоє граничних значень – нижнє та верхнє.

Перше граничне значення – 1. Друге граничне значення – 100. Додаємо до них, що стоять поруч значення:

– 0, 1, 2;

– 99, 100, 101.

Алгоритм використання техніки граничних значень наступний: визначити класи еквівалентності, визначити граничні значення для кожного класу (важливо розуміти до якого класу належить значення), провести тести з перевірки значення до границі, на границі і відразу після границі.

1.5.4 Аналіз причинно-наслідкових зв'язків

Тестування причинно-наслідкового ефекту – це техніка тест-дизайну, в якій для проєктування використовується графічне відображення вхідних даних (причин) і вихідних даних (наслідків). Різні комбінації причин можуть призвести до різних результатів. Для використання метода потрібне розуміння булівої логіки (логічних операторів – і, або, не). Аналіз побудованих зв’язків дає можливість комплексно обирати високорезультативні тести. Завдяки цьому методу з’являється можливість на ранньому етапі виявляти недоліки вимог чи специфікацій [10].

Тести будуються в декілька етапів:

– специфікація розбивається на робочі ділянки;

– у специфікації визначаються безліч причин і наслідків, під причиною розуміється окрема вхідна умова або клас еквівалентності, наслідок являє собою вихідну умову або перетворення системи, кожній причині і наслідку присвоюється номер;

– на основі аналізу семантичного змісту специфікації будується таблиця істинності, в якій послідовно перебираються всілякі комбінації причин і визначаються результати (наслідки) для кожної комбінації причин.

Таблиця забезпечується примітками, які задають обмеження і описують комбінації, які неможливі. Недоліком цього підходу є погане дослідження граничних умов.

Це та техніка, яка допомагає скласти кейси для функціональності, враховуючи усі елементи, що взаємодіють і впливають один на інший.

Щоб застосовувати дану техніку, необхідно знати:

– які саме параметри на вході (з чим оперує дана функціональність);

– які дані на виході (для чого взагалі ця функціональність, що вона робить);

– як вхідні параметри впливають на результат.

Кожна попередня комбінація може бути пронумерована і записана в окремі таблиці істинності, в яких істина позначена як «1», неправда як «0». Для невизначених станів використовують позначку «X», що може бути як «1», так і «0».

Усі рядки таблиці істинності перетворюються в тести. При цьому за можливості потрібно поєднувати тести із незалежних таблиць, використовуючи попарне тестування і для класів еквівалентності потрібно додатково прописувати вхідні умови.

Цей метод дозволяє відштовхуватися від причин того, що потрібно зробити в рамках тестового сценарію і плавно переходити до наслідків. Як і всі техніки тест-дизайну, вона допомагає визначити найменшу кількість тестів для того, щоб знайти більше дефектів. З’ясовуючи усі підстави і результати можна переконатися, що під час будь-яких маніпуляцій у системи буде відповідь. Також ця техніка дозволяє знайти недоліки в логіці опису додатка, що в подальшому допоможе покращити документацію.

1.5.5 Аналіз застосування методів ШІ для тестування

Впровадження штучного інтелекту (ШІ) у сферу тестування програмного забезпечення відкриває нові горизонти можливостей. ШІ не лише спрощує та автоматизує традиційні процеси, але й вносить новітні підходи до виявлення та усунення помилок, а також оптимізації тестових стратегій.

З використанням ШІ, процес тестування еволюціонує від простої

автоматизації до розумного аналізу даних і прийняття рішень. ШІ може аналізувати великі обсяги даних про помилки з минулого, визначати залежності та передбачати потенційні проблемні місця у новому коді. Такий підхід дозволяє не лише виявляти помилки, але й прогнозувати їх, що є значним кроком уперед у порівнянні з традиційними методами.

Використання ШІ у тестуванні не обмежується лише технічними аспектами. Це також сприяє глибшому розумінню та адаптації до змінних вимог і очікувань користувачів. Алгоритми ШІ, здатні аналізувати великі масиви користувацьких даних, можуть допомогти у формуванні більш точних тестових сценаріїв, що відображають реальні умови використання програмного продукту.

Втім, впровадження ШІ в тестування також ставить перед фахівцями нові виклики. Один з них - це забезпечення якості і точності алгоритмів ШІ. Неправильно навчений алгоритм може пропускати помилки або генерувати хибні позитиви. Тому, паралельно з розвитком ШІ, потрібно розвивати і знання та навички фахівців у галузі машинного навчання та аналітики даних.

Застосування ШІ у тестуванні - це не лише про автоматизацію, але й про інтелектуальне удосконалення процесу, що включає розуміння вимог, аналіз помилок, прогнозування ризиків та оптимізацію стратегій тестування.

Штучний інтелект (ШІ) пропонує інноваційні методи для підвищення ефективності тестування програмного забезпечення. Ці методи варіюються від автоматизації та оптимізації тестових процесів до вдосконалення якості та надійності тестування.

Методи штучного інтелекту (ШІ) в тестуванні програмного забезпечення відіграють важливу роль у підвищенні ефективності та точності процесу тестування. Ось кілька ключових методів:

– машинне навчання (ML);

– обробка природної мови (NLP);

– нейронні мережі;

– підсилювальне навчання.

Використання ML дозволяє автоматизувати створення тестових сценаріїв. Моделі машинного навчання аналізують історичні дані про помилки і використовують цю інформацію для прогнозування можливих дефектів у новому коді. Це може допомогти виявити потенційні проблеми на ранніх етапах розробки.

NLP може використовуватися для аналізу технічних вимог до програмного забезпечення та автоматичного генерування тестових випадків з цих вимог. NLP займається «використанням людських мов комп'ютером». У нього є багато різних додатків, які відносяться до неструктурованої природної мови людини. Наприклад, його областями застосування є машинний переклад, розпізнавання мови, діалогові системи, розпізнавання іменованих об'єктів, пошук інформації і класифікація тексту. Таким чином, область NLP охоплює всі взаємодії між комп'ютером і людиною з використанням письмової або усної природної мови [11].

Нейронні мережі допомагають виявляти складні взаємозв'язки та закономірності у тестових даних, які можуть бути неочевидні для тестувальників. Це може виявити приховані дефекти або специфічні умови, які викликають помилки.

Підхід підсилювального навчання використовується для оптимізації стратегій тестування. Системи, що базуються на підсилювальному навчанні, вчаться вибирати найбільш ефективні тестові сценарії на основі попереднього досвіду, що може значно знизити час тестування і підвищити якість.

Ці методи дозволяють тестувальникам програмного забезпечення працювати більш ефективно, забезпечуючи вищу якість та надійність продуктів. Нейронні мережі вносять великий вклад у виявлення складних взаємозв'язків і закономірностей у тестових даних, які можуть бути неочевидні для людського аналітика. Вони можуть виявляти приховані шаблони та тенденції, що дозволяє вчасно ідентифікувати потенційні проблеми.

2 ТЕОРЕТИЧНІ РОЗРОБКИ ТА АЛГОРИТМИ ВИРІШЕННЯ ЗАДАЧІ ПРОВЕДЕННЯ ТЕСТУВАННЯ

2.1 Формалізація алгоритму автоматизованого тестування

Основною метою формалізації алгоритму є забезпечення повноти тестування та створення повторюваних та незалежних тестових сценаріїв.

Першим кроком у формалізації алгоритму є визначення тестових сценаріїв. Тестові сценарії повинні відображати різноманітні можливості веб-сайту і охоплювати всі важливі функціональні аспекти. Наприклад, можуть бути створені тестові сценарії для реєстрації користувачів, авторизації, навігації по сторінках, заповнення форм, взаємодії з базою даних тощо. Важливо визначити очікувані результати для кожного тестового сценарію, що допоможе оцінити правильність роботи веб-сайту.

Далі слід визначити набір тестових даних, які використовуються для виконання тестових сценаріїв. Це можуть бути валідні та невалідні дані, деякі рандомні значення, граничні умови тощо. Важливо врахувати різноманітні варіанти, щоб переконатися, що веб-сайт поводиться правильно у всіх можливих ситуаціях.

Після цього необхідно реалізувати автоматизований код, який виконує тестування веб-сайту. Код має відповідати визначеним тестовим сценаріям та виконувати дії, необхідні для перевірки роботи веб-сайту. Також важливо включити перевірки та asserts, які допоможуть оцінити результати тестування та виявити проблеми.

Останнім етапом є запуск автоматизованих тестів і аналіз результатів. Під час запуску тестів необхідно врахувати різні середовища, такі як різні веб-браузери, операційні системи та пристрої, на яких веб-сайт буде використовуватися. Після завершення тестування необхідно проаналізувати результати, виявити помилки та проблеми, інтегрувати автоматизовані тести у процес розробки та підтримки веб-сайту.

Узагальнюючи, формалізація алгоритму автоматизованого тестування веб-сайтів є важливим етапом, який допомагає забезпечити повноту тестування та створити незалежні та повторювані тестові сценарії. Це дозволяє розробникам перевірити функціональність, надійність та продуктивність веб-сайту перед його випуском та підтримкою.

2.2 Існуючі підходи до автоматизації. Паттерн Page Object Model

На даний момент до автоматизації тестування користувача інтерфейсу існує 4 основних підходи:

– capture/playback (записування та відтворення);

– scripting (написання сценарію);

– data-driven testing (управління даними тестування);

– keyword-based (тестування за ключовими словами).

Для довгострокових, складних проектів найбільш підходящими є Scripting та Data-driven testing підходи, вони вимагають більше ресурсів для реалізації, але при цьому окупаються в майбутньому.

Використання патерну Page Object Model (POM) є одним з поширених підходів у автоматизованому тестуванні веб-сайтів. Цей патерн дозволяє структурувати та упорядковувати код тестових сценаріїв шляхом виокремлення веб-елементів та взаємодій з ними в окремий об'єкт, який називається "сторінкою" (Page).

Основна ідея патерну POM полягає в тому, щоб кожній сторінці веб-сайту, з якою взаємодіє автоматизований тест, відповідало окремий клас (або об'єкт). Цей клас містить в собі всі необхідні веб-елементи та методи, які дозволяють взаємодіяти з цими елементами. Наприклад, для сторінки реєстрації веб-сайту може бути створений клас «RegistrationPage», який включає в себе поля для введення даних, кнопку «Зареєструватися» та методи для заповнення полів та виконання дій.

Переваги використання патерну Page Object включають покращену читабельність та підтримку коду, підвищену стабільність тестів, повторне використання коду та покращену співпрацю між командами. Використовуючи цей патерн, розробники можуть легко розбити код на окремі класи, що спрощує розуміння та зміни в майбутньому. Крім того, патерн Page Object допомагає знизити залежності між тестами та структурою веб-сайту, забезпечуючи стабільність тестових сценаріїв. Цей підхід також сприяє ефективному використанню коду, оскільки одні й ті ж класи Page можна використовувати для різних тестових сценаріїв, що уникне дублювання коду та спростить підтримку (рисунок 2.1).

POM клас повинен бути нейтральним об'єктом, який взаємодіє та збирає інформацію про вашу програму. Методи мають бути нейтральними і переконайтеся, що вони досить універсальні, щоб їх можна було використовувати у будь-якому тесті, який хоче взаємодіяти зі сторінкою. Це означає, що ці методи не повинні входити твердження [12].

На жаль, за наявності великої кількості тестів користувача шляху, багато з яких повторюють один і той же базовий набір кроків (наприклад, вхід в систему, вибір валідного елемента і т.д.), перехід від імперативного до декларативного стилю інтерфейсу може бути недостатнім. У подібних випадках потрібно використовувати ще більш високий рівень абстракції, який фактично охоплює об'єкти сторінки. Для вирішення цієї проблеми можна вдатися до застосування моделі актора чи агрегатора. Терміни "актор" та "агрегатор" є взаємозамінними.

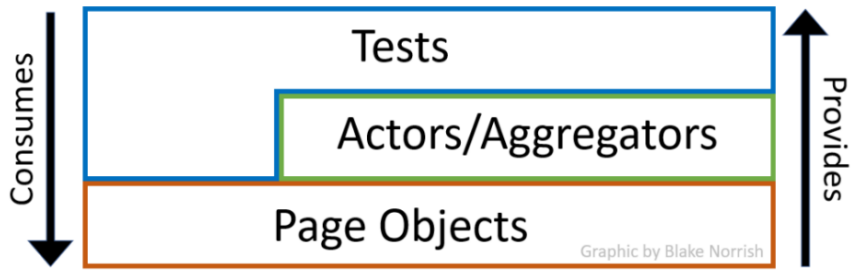

Рисунок 2.1 – Схема патерну Page Object Model

Актори об'єднують дію між об'єктами, щоб представити ці загальні агреговані сторінки дії на сторінці у вигляді фрагментів, що багато разів повторюються [13]. Вони мають інтерфейс вищого рівня для тестів. Таким чином, тесту не потрібно знову реалізовувати цю послідовність, він просто використовує інтерфейс, актором.

2.3 Створення тестових сценаріїв

Тестові сценарії в атоматизованому тестуванні є документованою послідовністю кроків, які виконуються для перевірки певної функціональності або властивостей програмного забезпечення. Вони є основою для розробки та виконання автоматизованих тестів. Тестові сценарії описують необхідні дії, вхідні дані та очікувані результати, які тест повинен перевірити. Створення ефективних тестових сценаріїв вимагає докладної розробки, чіткого розуміння функціональних вимог та врахування потенційних сценаріїв використання програмного забезпечення. Це допомагає забезпечити якість тестування та ефективне виявлення помилок.

В процесі створення тестових сценаріїв, відбувається за допомогою концепції BDD (Behavior-Driven Development) – методології розробки програмного забезпечення, яка спрямована на співпрацю між розробниками, тестувальниками та бізнес-аналітиками. BDD дозволяє створювати зрозумілі та легко зчитувані тестові сценарії, які фокусуються на поведінці системи з точки зору бізнес-вимог.

У нашому випадку, тестові сценарії описують дії (Додаток А), очікувані результати та перевірки для перевірки правильності роботи системи з точки зору вимог бізнесу. Застосування BDD дозволяє команді розробників та тестувальників зосередитися на функціональних можливостях системи та впевнитися, що вони виконують вимоги замовника.

Порівняно з BDD, TDD (Test-Driven Development) – це методологія розробки програмного забезпечення, яка базується на написанні тестів перед

написанням функціональності. У TDD розробник спочатку пише тест, який описує очікувану поведінку коду, а потім реалізує функціонал для його проходження. TDD акцентується на покритті коду тестами та забезпеченні правильності роботи окремих функцій.

BDD та TDD мають багато спільного, але основна відмінність полягає в тому, що BDD зорієнтована на співпрацю між всіма сторонами проекту (розробниками, тестувальниками, бізнес-аналітиками), тоді як TDD більше фокусується на розробників та їхньому розумінні функціональності. BDD зазвичай використовує спеціальні інструменти, такі як Cucumber або JBehave, для створення зрозумілих та легко зчитуваних тестових сценаріїв.

Узагальнюючи, BDD та TDD є методологіями, які сприяють покращенню процесу розробки програмного забезпечення, забезпечуючи високу якість продукту та зниження витрат на виправлення помилок. Обидва підходи можуть використовуватися окремо або разом в залежності від вимог проекту та вподобань команди розробників.

3 ВИБІР ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ

3.1 Аналіз програмних середовищ для автоматизації тестування

У автоматизованому тестуванні мови програмування використовуються для розробки тестових сценаріїв, взаємодії з фреймворками та інструментами автоматизації. Вибір мови програмування залежить від ваших вимог, навичок команди тестувальників та технологій, що використовуються в проекті.

Java є однією з найпоширеніших мов програмування в автоматизованому тестуванні. Вона має велику спільноту, розгалужені фреймворки, такі як Selenium, TestNG і JUnit, які підтримують автоматизацію тестування веб-додатків і API. Java також підтримує багато інших інструментів для тестування, таких як Appium і Cucumber [14].

Обґрунтування вибору Java, Selenium, Lombok, TestNG, Maven і Allure-Maven для написання фреймворку тестування базується на їхніх перевагах та властивостях, які забезпечують ефективну та зручну розробку автоматизованих тестів.

Java широко використовувана мова програмування зі значною підтримкою та розширеними можливостями. Вона надає потужність та гнучкість для розробки різноманітних функцій.

Selenium є відомим інструментом для автоматизації тестування веб-додатків. Він забезпечує можливість ефективно взаємодіяти з елементами сторінок, виконувати дії користувача та перевіряти очікувані результати. Архітектура Selenium дозволяє використовувати його як локально, так і у розподіленому режимі, а також інтегрувати з різними інструментами для Continuous Integration (CI).

Lombok допоміжна бібліотека, яка дозволяє автоматично генерувати методи доступу, такі як геттери та сеттери, а також зменшує кількість коду.

Використання Lombok спрощує розробку, поліпшує читабельність коду та прискорює процес вирішення завдань.

Таблиця 3.1 – Популярні фреймворки для автоматизації тестування користувацького інтерфейсу.

Фреймворк	Переваги	Недоліки
Selenium	Велика спільнота та документація. Широкий функціонал. Можливість паралельного виконання тестів.	Потребує встановлення браузерних драйверів. Обмежена підтримка мобільних додатків.
Cucumber	Легка зрозумілість для нетехнічних учасників проекту. Можливість використання як замовниками, так і розробниками.	Вимагає додаткової інтеграції з іншими фреймворками. Велика кількість файлів та налаштувань для запуску тестів.
TestNG	Широкий спектр аннотацій, надає широкі можливості для налаштування й організації тестових сценаріїв.	Обмежена підтримка мов програмування, відсутність вбудованої підтримки для BDD.
Cypress	Швидкість виконання тестів. Простота візуальних перевірок.	Обмежена підтримка браузерів (Chrome і Chromium).

Вище наведена загальна інформація популярних фреймворків, їх можливості, переваги та недоліки (таблиця 3.1). Кожен з них має свою архітектуру, структуру, модульність та розширюваність, спрямовані на полегшення процесу розробки й підтримки тестових сценаріїв.

TestNG є потужним фреймворком для тестування, який надає багато функціональних можливостей, таких як групування тестів, налаштування тестових сценаріїв, покриття коду, паралельне виконання тестів та інше. TestNG дозволяє ефективно організовувати та виконувати тестові сценарії, забезпечуючи надійність та стабільність тестування.

Кожен з цих фреймворків та бібліотек має свої переваги та недоліки, але загалом вони спрямовані на полегшення процесу автоматизації тестування й надають широкі можливості для розширення функціональності. Їх архітектура, модульність та розширюваність дозволяють розробникам створювати потужні й гнучкі тестові набори, а також інтегрувати їх у різні процеси розробки та CI/CD.

Вибір мови програмування в автоматизованому тестуванні залежить від конкретних потреб проекту, наявних навичок команди тестувальників. Кожна мова має свої переваги і специфіку, тому важливо вибрати мову, яка найкраще відповідає потребам проекту.

3.2 Обґрунтування вибору інструментів для інтеграції ШІ в автоматизоване тестування

Тестування на неузгодженості в макеті за допомогою машинного навчання вимагає використання спеціалізованих моделей, які можуть виявляти дефекти користувацького інтерфейсу веб-сайтів.

Основною мовою програмування більшості задач пов’язаних з ШІ фахівців є Python. Python досить проста мова, в ній реалізовані бібліотеки для обробки та аналізу даних, розглянемо деякі з них:

– TensorFlow;

– Keras;

– Python Imaging Library.

Tensorflow – це бібліотека, яка допомагає інженерам створювати та тренувати моделі глибокого навчання. Він надає всі інструменти, необхідні

для створення нейронних мереж. Ми можемо використовувати tensorflow для навчання простих і складних нейронних мереж, використовуючи великі набори даних.

Tensorflow використовується в різноманітних програмах, від розпізнавання зображень і мови до обробки природної мови та робототехніки. TensorFlow дає нам змогу швидко та легко створювати потужні моделі ШІ з високою точністю та продуктивністю [15].

Keras був створений для того, щоб його вивчення було простим та простим. Він пропонує послідовні та прості API, чітко пояснює помилки користувачів. Час прототипування в Keras невеликий, що призводить до швидкої реалізації. Keras також глибоко інтегрований із TensorFlow. Він також плавно працює як у процесорі, і на графічному процесорі. Він підтримує практично всі нейронні мережі. Більше того, у Keras дуже широке співтовариство, тому знайти допомогу з будь-якої проблеми не складе труднощів. Документація велика і зрозуміла [16].

Python Imaging Library додає можливості обробки зображень до вашого інтерпретатора Python. Ця бібліотека забезпечує розширену підтримку форматів файлів, ефективне внутрішнє представлення та досить потужні можливості обробки зображень.

3.3 Розробка загального алгоритму функціонування фреймворку

Розробка загального алгоритму функціонування фреймворку для автоматизованого тестування користувацького інтерфейсу веб-сайтів з використанням Java, Selenium, Lombok, TestNG, Maven та Allure-Maven включає кілька важливих кроків. Першим етапом є визначення функціональних вимог, що передбачаються для фреймворку.

Наступним кроком є створення базових класів та інтерфейсів, які будуть використовуватися в фреймворку, таких як класи для сторінок та тестів.

Далі, необхідно реалізувати класи сторінок, що відповідають кожній окремій сторінці веб-сайту. Ці класи міститимуть елементи сторінок та методи, що дозволяють взаємодіяти з ними. Використання аннотацій Lombok допоможе спростити процес створення цих класів шляхом автоматичного генерування методів, таких як геттери та сеттери.

Засобами TestNG, реалізовані тестові сценарії. Ці сценарії використовують класи сторінок для взаємодії з елементами сторінки та перевірки очікуваних результатів. TestNG забезпечує багато можливостей для організації тестів, включаючи пакетування тестів, групування, використання аннотацій та залежностей між тестами.

Крім того, використовуючи Maven, налаштовані залежності та збірку проекту. Maven дозволяє ефективно управляти бібліотеками та налаштуваннями проекту, спрощуючи процес розробки та забезпечуючи повторне використання коду. Для створення звітів про результати тестування використана бібліотека Allure-Maven. Цей інструмент надає можливість створювати зрозумілі звіти, які включають інформацію про пройдені тестові сценарії, їх статус, скріншоти та інші деталі.

В цілому, розробка загального алгоритму функціонування фреймворку для автоматизованого тестування веб-сайтів включає визначення вимог, створення базових класів та інтерфейсів, реалізацію класів сторінок, написання тестових сценаріїв та налаштування збірки проекту, а також генерацію звітів. Структура розробленого проекту передбачає організацію коду у різні пакети. Пакет "configuration" містить класи WebDriverProvider і ConfigProvider, які відповідають за налаштування драйвера та завантаження конфігураційних параметрів з файлу config.properties. Пакет "webPages" містить класи, які представляють окремі сторінки веб-додатка. Абстрактний клас BasePage надає базовий функціонал для роботи зі сторінками, такий як відкриття сторінки, прокрутка до елемента, очікування видимості елемента тощо (Додаток Б).

Клас WebDriverProvider використовується для отримання екземпляра WebDriver (лістинг 3.1), який потрібен для автоматизованого керування браузером. Він динамічно вибирає відповідний драйвер (ChromeDriver, FirefoxDriver або SafariDriver) залежно від конфігураційного параметра driver, який зчитується з класу ConfigProvider. Цей клас дозволяє легко змінювати і вибирати різні драйвери залежно від потреб проекту.

Лістинг 3.1 – Реалізація класу WebDriverProvider

```
public class WebDriverProvider {
public static WebDriver getDriver(){
String driverName = ConfigProvider.getDriverName();
switch (driverName){
case "chrome":return WebDriverManager.chromedriver().create();
case "firefox":return WebDriverManager.firefoxdriver().create();
case "safari":return WebDriverManager.safaridriver().create();}}}
```

Клас ConfigProvider використовується для зчитування конфігураційних параметрів з файлу config.properties. Він забезпечує централізований доступ до параметрів, таких як базова URL-адреса (BASE_URL) і тайм-аут (TIMEOUT), що використовуються в різних частинах програми. Клас також надає метод getDriverName(), який повертає назву драйвера для використання в WebDriverProvider (лістинг 3.2).

Лістинг 3.2 – Реалізація класу ConfigProvider

```
public static Properties initProperties (){
    try{properties.load(ClassLoader
.getSystemResourceAsStream("config.properties"));}catch
(IOException e){
System.err.println("Could not load properties.");}
        return properties;}
```

Абстрактний клас BasePage є базовим класом для сторінок веб-додатка. Він містить загальні методи та функціональність, які можуть бути використані на будь-якій сторінці, такі як open(), scrollToElement() і waitForVisibility() (лістинг 3.3). Клас також містить посилання на WebDriver і WebDriverWait, що дозволяє здійснювати дії з елементами сторінки, очікувати певних умов та взаємодіяти з браузером.

Лістинг 3.3 – Реалізація класу BasePage

```
public abstract class BasePage {
    protected WebDriver driver;
    protected WebDriverWait wait;
protected final Logger logger = LoggerFactory
                                    .getLogger(getClass());
     public BasePage(WebDriver driver) {
this.driver = driver;
this.wait = new WebDriverWait(driver,Duration.ofSeconds(10));
PageFactory.initElements(driver, this);}
     public void open(String path){driver.get(BASE_URL+path);}
     public void scrollToElement(Point point) {
((JavascriptExecutor) driver)
          .executeScript("window.scrollTo" + point.toString());  }
     public void waitForVisibility(WebElement element) {
WebDriverWait
wait=new WebDriverWait(driver,Duration.ofSeconds(5));
wait.until(ExpectedConditions.visibilityOf(element));}
```

Клас BasePageTest є базовим класом для тестування сторінок веб-додатка. Він використовує WebDriverManager для автоматичного керування встановленням необхідних драйверів. Клас надає методи setUp() і tearDown(), які викликаються перед і після кожного тесту відповідно. У методі setUp(), створюється екземпляр WebDriver і встановлюється максимальний розмір

вікна браузера. У методі tearDown(), закривається браузер (лістинг 3.4). Цей клас дозволяє ініціалізувати тестове середовище перед кожним тестом і забезпечити коректне завершення після кожного тесту.

Лістинг 3.4 – Реалізація класу BasePageTest

```
public class BasePageTest{
     @BeforeMethod
public void setUp() {log.info("Initialize web driver.");
    driver = WebDriverManager.chromedriver().create();
    driver.manage().window().maximize();}
public void open(String path){driver.get(BASE_URL + path);}
     @AfterMethod
    public void tearDown(){log.info("Initialise driver quit.");
     driver.quit();}}
```

В розробці програмного коду використовується підхід Behaviour-Driven Development (BDD), який фокусується на описі поведінки системи з точки зору користувача. У цих методиках тести використовуються як спосіб опису і перевірки очікуваної поведінки системи. Програмний код реалізований з урахуванням принципів ООП (об'єктно-орієнтованого програмування) та чіткої структуризації, що дозволяє підтримувати читабельність, розширюваність та перевикористання коду (Додаток В). Тестові класи, як AddItemToWishListTest (лістинг 3.5), розміщені у відповідних пакетах, які відповідають за тестування функціональності. Ці класи використовують об'єкти сторінок та методи для виконання різних дій та перевірки результатів.

Лістинг 3.5 – Тестовий клас AddItemToWishListTest

```
public class AddItemToWishListTest extends BasePageTest{
    @Test(groups = "smoke")
```

```
    @Description("Adding a Product to Wishlist.")
    public void testAddItemToWishList(){
HomePage homePage = new HomePage(driver);open("");
homePage.dismissNoticeMessage();
homePage.clickMyAccountLink();
LogInPage logInPage = new LogInPage(driver);
logInPage.login("Student", "password_password");
logInPage.clickLogo();
homePage.addToWishListPinkDropItem();
homePage.clickToWishList();
WishListPage wishListPage = new WishListPage(driver);
    Assert.assertTrue(wishListPage.isWishListTitleDisplayed(),
"Wish list title is not displayed");
     Assert.assertTrue(wishListPage
     .isPinkDropTitleWishListDisplayed(),
     "Added item is not displayed in Wish List");}}
```

Структура проекту дотримується принципів модульності та логічного групування компонентів. Кожен клас відповідає за конкретну функціональну частину системи, що сприяє легкості розробки, розширення та підтримки коду. Крім того, використання класу-батька BasePageTest для тестових класів дозволяє спільне використання ресурсів та налаштувань для всіх тестів.

Організація структури проекту та використання методики розробки програмного коду допомагають підтримувати код організованим, читабельним та масштабованим. Такий підхід сприяє якості та стабільності програмного продукту, а також полегшує спільну роботу розробників у командному середовищі.

В роботі була проведена розробка візуального інтерфейсу проекту для побудови звіту про виконання тестів та аналізу результатів було використано інструмент Allure – потужним інструментом для побудови інтерфейсу, він надає детальну інформацію про виконання тестів, включаючи результати, статуси, повідомлення про помилки та скріншоти (рисунок 3.1, 3.2).

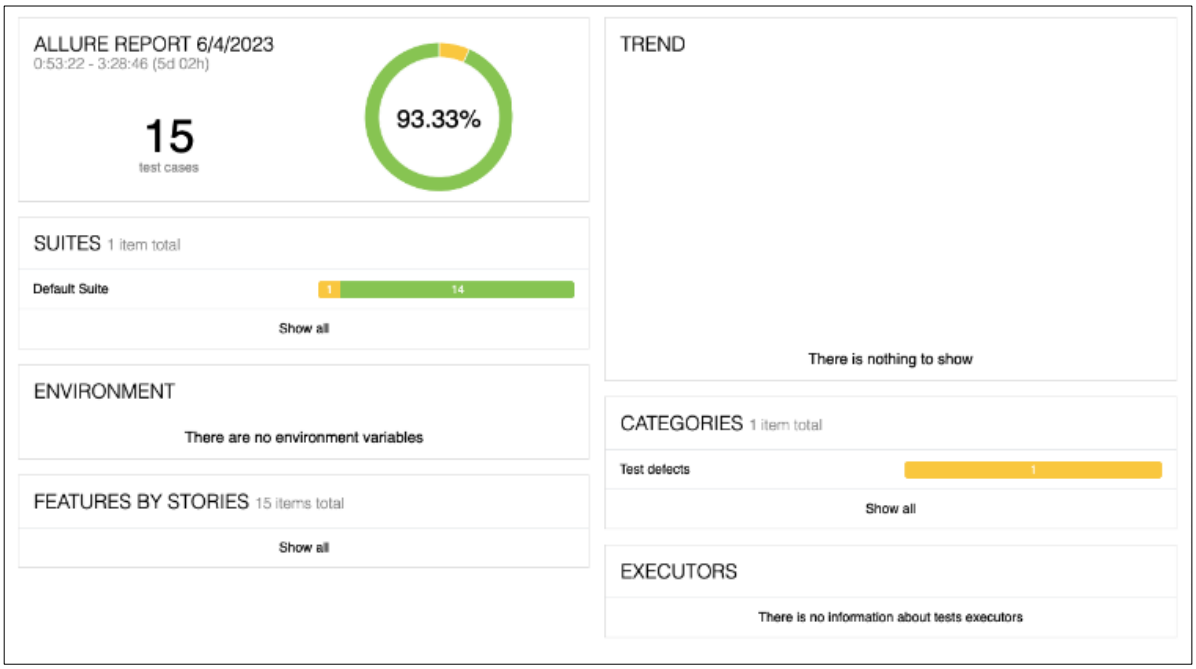

Рисунок 3.1 – Приклад візуального звіту про виконання тестів

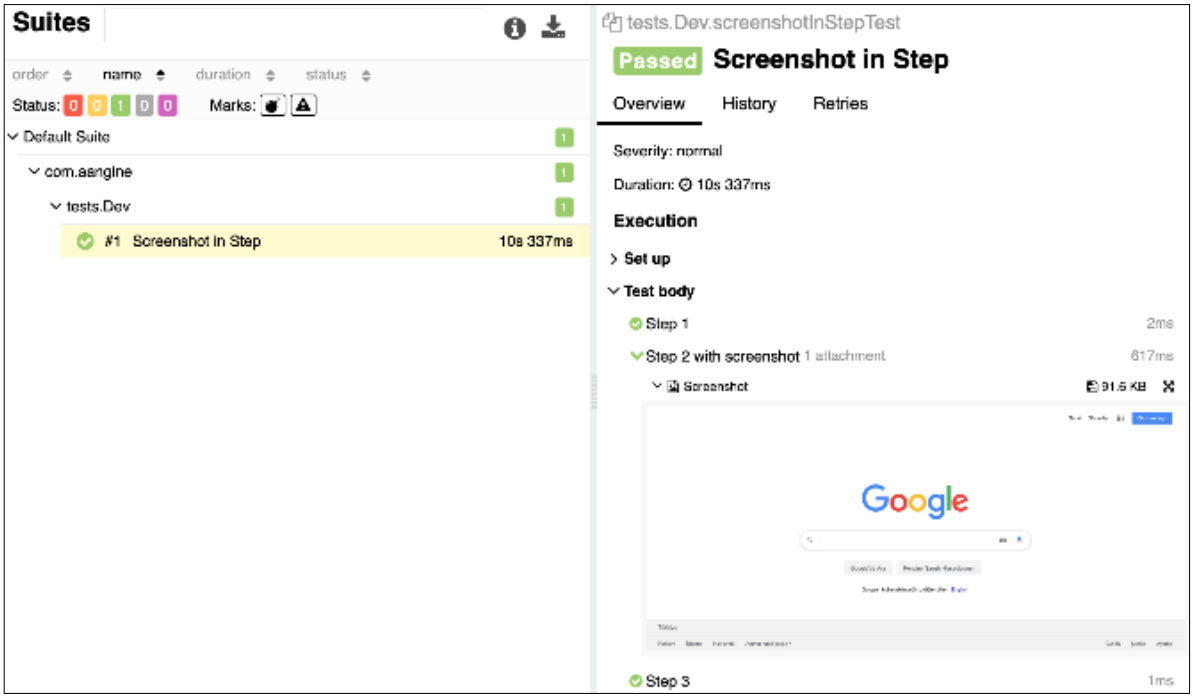

Рисунок 3.2 – Приклад розгорнутого візуального звіту

В процесі розробки було налаштовано залежності Maven, включаючи TestNG і Allure, для забезпечення необхідних функціональних можливостей. Для цього була внесена необхідна конфігурація у файл pom.xml проекту (лістинг 3.6).

Лістинг 3.6 – Налаштування залежностей в файлі конфігурації

```
<dependencies>
     <dependency>
                <groupId>io.qameta.allure</groupId>
               <artifactId>allure-testng</artifactId>
               <version>${allure.testng.version}</version>
               <scope>test</scope>
        </dependency>
        <dependency>
               <groupId>io.qameta.allure</groupId>
               <artifactId>allure-maven</artifactId>
               <version>${allure.maven.version}</version>
        </dependency>
<dependencies>
```

Для написання тестових сценаріїв були використані аннотації TestNG. Тестові сценарії можуть бути запущені, використовуючи команду Maven mvn test або з запуску тестового класу безпосередньо з інтегрованої середовища розробки (IDE). Після виконання тестів у проекті була створена папка allure-results, де зберігаються результати виконання тестів у форматі JSON. Ці результати включають в себе інформацію про стан виконання тесту, повідомлення про помилки, скріншоти та інші деталі.

Використання Allure у роботі дозволило ефективно вести тестування проекту та забезпечити зручний докладний звіт для аналізу результатів. Інструмент допоміг у виявленні та виправленні помилок, а також забезпечив стабільність та якість розробленого програмного забезпечення.

4 ЗАСТОСУВАННЯ ШТУЧНОГО ІНТЕЛЕКТУ В ТЕСТУВАННЯ

4.1 Визначення методу навчання та обробки інформації

Для вирішення задачі автоматизованого тестування користувацького інтерфейсу веб-додатків з інтеграцією ШІ найбільш зручним та ефективним підходом є використання CNN.

Згорткові нейронні мережі (CNN) виявилися надзвичайно ефективними у розпізнаванні зразків та особливостей на зображеннях. Їх можна застосувати для тренування моделей на датасетах скріншотів веб-сайтів, де кожен скріншот маркований як коректний або містить помилки у макеті. Це дозволяє моделі автоматично ідентифікувати і класифікувати помилки у веб-дизайні.

Згорткові нейронні мережі – це клас глибинних штучних нейронних мереж прямого поширення, який застосовується до аналізу зображень.

Архітектура CNN складається з декількох шарів: згорткових шарів, шарів пулінгу та повнозв'язних шарів. Згорткові шари використовують фільтри (ядро згортки) для сканування вхідних зображень і виділення важливих особливостей. Шари пулінгу зменшують розміри отриманих карт особливостей, вибірково зберігаючи найбільш важливу інформацію. Повнозв'язні шари, які зазвичай розміщуються в кінці архітектури, виконують класифікацію на основі отриманих особливостей (рисунок 4.1).

Використання CNN для автоматичного виявлення аномалій вимагає створення датасету з валідними скріншотами веб-сайту. Модель навчається розуміти, що є стандартним для макету сайту, і використовується для ідентифікації відхилень від цього нормального стану.

Семантичне сегментування – це техніка, яка дозволяє CNN розуміти та відрізняти різні частини веб-сторінки (наприклад, заголовки, текстові блоки, кнопки). Модель може виявляти, чи знаходяться ці елементи у відповідних

місцях макету, що є критично важливим для забезпечення консистентності веб-дизайну.

Техніки глибокого навчання можуть бути застосовані для навчання моделей виявляти та класифікувати окремі елементи на скріншотах, такі як меню, логотипи, кнопки. Це дозволяє перевіряти їх правильність та консистентність на різних сторінках сайту, що покращує загальну якість та надійність веб-проекту.

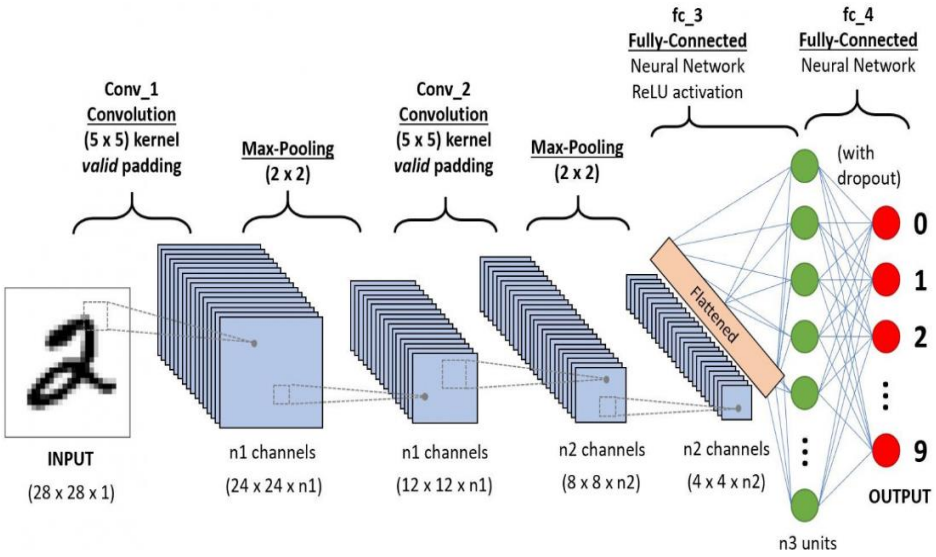

Рисунок 4.1 – Принцип роботи CNN

Розглянемо переваги CNN:

– автоматичне виявлення особливостей. На відміну від традиційних методів, де особливості потрібно визначати та кодувати вручну, CNN автоматично виявляють і навчаються розпізнавати особливості з даних.

– збереження просторової ієрархії. CNN ефективно обробляють просторові відносини між об'єктами на зображенні, зберігаючи просторову ієрархію особливостей.

– зменшення кількості параметрів. Завдяки спільному використанню ваг і згорткам, CNN вимагають значно меншої кількості параметрів порівняно з повнозв'язними мережами, що робить їх більш ефективними з точки зору обчислень.

4.2 Розробка та інтеграція згорткової нейронної мережі

Першим кроком для автоматизованого тестування дизайну веб-сайту з використанням штучного інтелекту та машинного навчання є збір порівнювальних даних (лістинг 4.1). У цьому лістингу використовуються TakesScreenshot для отримання скріншоту та Apache Commons IO для збереження файлу скріншоту.

В кваліфікаційній роботі використовується ШІ як один зі способів тестування. Тому були розроблені допоміжні методи для МН як кроки виконання тестів.

Лістинг 4.1 – реалізація збору даних

```
@Step
    public static void takeScreenshots() {
        try {
            // Відкриття сторінки
            driver.get("https://demoqa.com/");
            // Створення скріншоту
            File screenshot = ((TakesScreenshot) driver)
                              .getScreenshotAs(OutputType.FILE);
            FileUtils.copyFile(screenshot,
new File("images-for-ml/gotten-images/original.png"));
        } catch (IOException e) {
            e.printStackTrace();
        }}
```

Наступним кроком є автоматизація попередньої обробки даних, зокрема, обробка скріншотів, використовуючи Java (лістинг 4.2). Обробка може включати зміну розміру зображень, обрізку, корекцію кольору, перетворення в інший формат тощо. Один з популярних інструментів для обробки зображень у Java – це бібліотека Apache Commons Imaging.

Лістинг 4.2 – реалізація обробки даних

```
@Step
  public static void resizeImage(String inputImagePath, String outputImagePath, int scaledWidth, int scaledHeight) throws IOException {
        try {
        // Читання зображення
        File inputFile = new File(inputImagePath);
        BufferedImage inputImage = ImageIO.read(inputFile);

        // Створення нового зображення зміненого розміру
        BufferedImage outputImage = new BufferedImage(scaledWidth, scaledHeight, inputImage.getType());
        // Масштабування вхідного зображення до вихідного
        outputImage.getGraphics().drawImage(inputImage, 0, 0, scaledWidth, scaledHeight, null);
        // Збереження зміненого зображення
        File outputFile = new File(outputImagePath);
        ImageIO.write(outputImage, "png", outputFile);
        } catch (IOException e) {
            e.printStackTrace();
        }}
```

Створення базової моделі машинного навчання для тестування веб-сторінок веб-сайтів може бути виконано у кілька кроків.

Потрібно зібрати скріншоти сторінок сайту. В нашому проекті є дві категорії скріншотів – "правильні", які вже зібрані і оброблені; "неправильні", з візуальними вадами або помилками макету. Зображення потрібно перетворити в єдиний формат і розмір, щоб їх можна було використовувати для навчання моделі.

Для інтеграції моделі згорткової мережі, валідації та обробки результатів в цьому проекті використане Rest API. API – це механізм, який дозволяє

програмі або службі отримувати доступ до ресурсу в іншій програмі або службі [17].

Звернемо увагу, що для реального використання такої моделі потрібно значно більше даних та більш детальна підготовка. Тут розглядається простий приклад на Python з використанням Keras та TensorFlow для створення простої моделі згорткової нейронної мережі (Додаток Г).

Для реалізації цього був розроблений FastAPI додаток, що відкриває ендпойнт для отримання даних, додана логіка для обробки даних в моделі машинного навчання (Додаток Д), отримані результати інтегровані в процес тестування на Selenium та оброблені (лістинг 4.3). Схема реалізації автоматизованого тестування дизайну веб-сайту з використанням штучного інтелекту та машинного навчання наведена на рисунку 4.2.

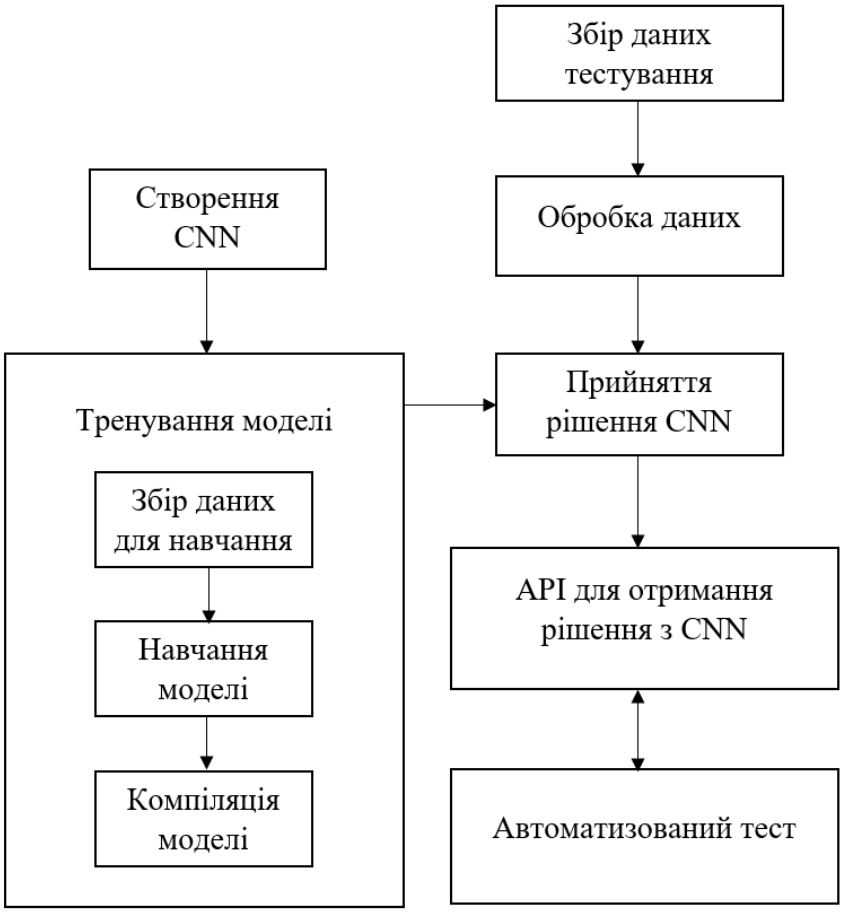

Рисунок 4.2 – Схема реалізації автоматизованого тестування з використанням штучного інтелекту та машинного навчання

Лістинг 4.3 – реалізація отримання результату в процесі тестування

```
@Step
    public static void getResultFromMaModel () {
        CloseableHttpClient httpClient =
HttpClients.createDefault();
        try { HttpPost uploadFile = new
HttpPost("http://localhost:5000/predict");
MultipartEntityBuilder builder = MultipartEntityBuilder.create()
String filePath = "path/to/your/screenshot.png";
builder.addBinaryBody("image", new File(filePath),
ContentType.APPLICATION_OCTET_STREAM, "screenshot.png");
            HttpEntity multipart = builder.build();
            uploadFile.setEntity(multipart);
            CloseableHttpResponse response =
httpClient.execute(uploadFile);
            HttpEntity responseEntity = response.getEntity();
String result = EntityUtils.toString(responseEntity);
            System.out.println(result);
        } catch (IOException | ParseException e) {
            e.printStackTrace();}}
```

5 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТІВ ТА АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

У процесі проведення дослідження з тестування поля вводу віку персонала у програмному забезпеченні, було застосовано комплексний підхід, що включає методи класів еквівалентності та граничних значень. Такий підхід забезпечує глибокий аналіз поведінки системи при різноманітних вхідних даних та дозволяє ефективно ідентифікувати потенційні помилки.

Перший етап дослідження базувався на використанні еквівалентних класів з даними, що явно виходили за рамки визначених коректних значень (<=0, >99, a-z, !@#). Це дозволило оцінити реакцію системи на некоректні дані.

Другий етап включав тестування з використанням граничних значень. Тут основна увага була приділена аналізу поведінки системи на межах допустимого діапазону вікових показників, а саме 0 та 100. Такий підхід є критичним для виявлення помилок, які часто виникають у граничних умовах.

Третій етап об'єднав попередні методи і додав до тестування вхідні дані, які включали крайні значення (<0, 0, 100, >100) разом із буквено-цифровими та спеціальними символами. Це забезпечило комплексну оцінку здатності системи обробляти різні типи даних.

Для кількісної оцінки результативності тестування було застосовано поняття тестового покриття, яке визначається як відсоток випробуваних варіантів від загально можливої кількості. Формула для розрахунку тестового покриття може бути представлена як:

$$\text{Тестове Покриття} = \frac{\text{Кількість Випробуваних Варіантів}}{\text{Загальна Кількість Можливих Варіантів}} \times 100\% \qquad (5.1)$$

У контексті даного дослідження, загальна кількість можливих варіантів включає всі допустимі та недопустимі значення поля, а кількість

випробуваних варіантів включає усі випадки, які були фактично протестовані. Це дозволяє отримати об'єктивну оцінку комплексності та глибини проведеного тестування.

У рамках дослідження ефективності попарного тестування фільтрів на сторінці інтернет-магазину (рисунок 5.1), було проведено три серії тестів, що охоплюють різні комбінації вхідних даних (таблиця 5.1). Ці тести мали на меті перевірити сумісність, виробників і тип підключення.

Таблиця 5.1 – тест-кейси попарного для попарного тестування

№	Кроки	Очікуваний результат	Вхідні дані
1	Перейти до сторінки "Home"	Сторінка відкрита	
2	Перейти до сторінки "Spekers"	Сторінка відкрита	
3	Обрати значення фільтру "Compatibility"	Встановлено прапорець	"Any device that has bluetooth enabled"
4	Обрати значення фільтру "Manufacturer"	Встановлено прапорець	"HP"
5	Обрати значення фільтру "Wireless technology"	Встановлено прапорець	"Bluetooth"
6	Перевірити калькість відображеного товару		

Перший тест включав випадково генеровані значення для трьох фільтрів: сумісності, виробника та типу бездротового з'єднання. Використання випадково генерованих даних дозволяє перевірити систему на велику кількість можливих комбінацій і виявити неочікувані помилки або проблеми у взаємодії між фільтрами.

Другий тест був більш структурованим і включав конкретні комбінації фільтрів. Враховуючи, що тест включав усі можливі комбінації цих трьох фільтрів, ми маємо можливість провести всебічну перевірку взаємодії фільтрів. Такий підхід дозволяє ідентифікувати специфічні сценарії, в яких можуть виникнути помилки або непередбачені поведінки системи.

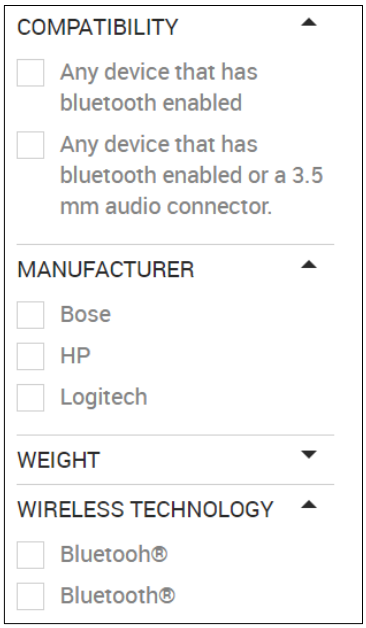

Рисунок 5.1 – Інтерфейс тестованого функціоналу

Третій тест був заснований на ортогональному масиві, що дозволяє зменшити загальну кількість тестів, зберігаючи при цьому високий рівень покриття комбінацій. Тут були вибрані специфічні комбінації фільтрів, які представляють собою різні аспекти сумісності, виробників і типів підключення, забезпечуючи репрезентативний зразок можливих сценаріїв.

Попарне тестування в даному випадку дозволило ефективно оцінити взаємодію різних фільтрів на сайті. Цей метод ефективний для виявлення проблем у взаємодії компонентів, особливо в умовах, де кількість можливих комбінацій є великою. Випадкова генерація даних у першому тесті дозволила виявити неочікувані помилки, тоді як структуровані сценарії у другому і третьому тестах забезпечили глибоке розуміння специфічних взаємодій між фільтрами.

Використання ортогонального масиву у третьому тесті демонструє ефективний підхід до зменшення кількості тестів без значного зниження якості тестового покриття. Це особливо важливо в контексті обмежених ресурсів та часу для тестування, дозволяючи оптимізувати процес і забезпечити ефективну перевірку взаємодії фільтрів.

У рамках дослідження тестування графічного користувацького інтерфейсу (GUI), було розглянуто два прогресивні методи: порівняльне тестування з використанням піксельного порівняння зображень і тестування за допомогою машинного навчання.

Перший метод базується на піксельному порівнянні зображень, отриманих з екрану GUI. Такий підхід включає автоматизоване отримання скріншотів екрана та їх подальше порівняння з еталонними зображеннями. Важливою складовою є аналіз розмірів зображень та індивідуальних пікселів, що дозволяє точно визначити невідповідності між актуальним станом інтерфейсу та його очікуваним дизайном. Такий метод є особливо ефективним для виявлення зовнішніх відхилень, які легко фіксуються на рівні зображення, проте він може бути нечутливим до складніших аспектів інтерфейсу, таких як взаємодія з користувачем та внутрішня логіка роботи.

Другий метод використовує машинне навчання, зокрема, глибокі нейронні мережі для аналізу зображень GUI. Цей підхід орієнтований на тренування моделі на основі набору даних, що включає зображення з різними варіантами GUI. Модель навчається розпізнавати певні патерни та відхилення від норми, що забезпечує більш глибокий аналіз інтерфейсу, ніж просте порівняння пікселів. Використання машинного навчання може бути значно ефективнішим у виявленні складних помилок у GUI, що включають динамічну взаємодію та контекстуальні аспекти інтерфейсу.

У процесі аналізу часових витрат та швидкості роботи двох методів тестування GUI – порівняльного тестування з використанням піксельного порівняння зображень і тестування за допомогою машинного навчання, слід враховувати ряд ключових аспектів. Піксельне порівняння зображень є

відносно швидким у виконанні, оскільки воно полягає в прямому порівнянні двох зображень без необхідності глибокого аналізу чи обробки даних. Цей метод не вимагає значного часу на підготовку або налаштування, окрім первинного створення еталонних зображень. Тестування може бути виконане досить швидко після отримання необхідних скріншотів (рисунок 5.2).

Рисунок 5.2 – Результати виконання тестів

Машинне навчання вимагає значних обчислювальних ресурсів, особливо під час фази тренування моделі. Однак, після тренування, сам процес використання моделі для аналізу GUI може бути швидким. Підготовка до тестування за допомогою машинного навчання є значно більш часовитратною. Це включає збір та обробку набору даних для тренування, сам процес тренування моделі, а також її налаштування та тестування.

Також, можна відмітити, що вони доповнюють один одного. Піксельне порівняння ефективне для швидкого виявлення візуальних невідповідностей, тоді як машинне навчання дозволяє проводити глибший аналіз з більшим фокусом на функціональні та контекстуальні аспекти інтерфейсу. Використання цих методів у комбінації може значно підвищити якість та ефективність процесу тестування GUI, забезпечуючи більш повне покриття потенційних помилок та відхилень.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було здійснено комплексне дослідження основних методик в області автоматизованого тестування веб-сайтів, з акцентом на використанні згорткових нейронних мереж для аналізу графічного інтерфейсу користувача (GUI). Робота демонструє високу ефективність та точність автоматизованого тестування порівняно з традиційними методами, що суттєво знижує час та витрати на тестування..

В роботі проаналізовані сучасні підходи до автоматизованого тестування, зокрема методів, які включають попарне тестування, еквівалентне розбиття, аналіз граничних значень та причинно-наслідкові зв'язки. Значну увагу було приділено застосуванню методів штучного інтелекту, насамперед згорткових нейронних мереж, для аналізу та тестування GUI. Для розробки та тренування моделі машинного навчання були застосовані такі методи як згорткові нейронні мережі (CNN), обробка зображень та алгоритми класифікації. Дослідження включало етапи підготовки даних, валідації моделі та її тестування на практичних даних.

Аналіз світових трендів та розробок в області автоматизації тестування показав, що запропоновані у роботі методики та алгоритми є актуальними та конкурентоспроможними. Вони відповідають сучасним вимогам якості та ефективності процесів розробки програмного забезпечення.

Новизна дослідження полягає у розробці та інтеграції моделі машинного навчання в класичний процес тестування програмного веб-дизайну, яка була навчена на датасеті, що містить різноманітні скріншоти веб-сторінок, із застосуванням методів глибокого навчання для класифікації макетів. Наукова цінність полягає в розробці методу, який підвищує точність і ефективність автоматизованого тестування, зменшуючи потребу в ручному аналізі.

Практична цінність розробки полягає в значному зниженні часу та ресурсів, необхідних для тестування веб-дизайну. Автоматизація процесу

тестування дозволяє швидко ідентифікувати дефекти, покращує якість кінцевого продукту та забезпечує високий рівень впевненості в надійності веб-сторінок. Застосування моделі також відкриває шлях для подальшої оптимізації процесів розробки та тестування.

У загальному, робота демонструє успішне поєднання теоретичних знань та практичних навичок у галузі автоматизації тестування спеціалізованих комп'ютерних систем, пропонуючи інноваційні підходи та рішення для сучасних викликів у цій області.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Дідковська М. В. Розробка та тестування програм: лекція.

2. Rashid H. DevOps guide: challenges, practices & solutions for businesses. – Globallogic, 2022.

3. Norrish B. Going deeper into the page object model, 2022.

4. Кріспін Л., Грегорі Дж. Гнучке тестування. – Вільямс, 2016. – 460 с.

5. Sharma S. Graphical user interface. – Amexfel Publishers Pvt Ltd, 2003. – 330 с.

6. Certified tester advanced level test analyst (CTAL-TA). ISTQB not-for-profit association. [Електронний ресурс]. – Режим доступу: https://www.istqb.org/certifications/test-analyst.

7. Pairwise Testing [Електронний ресурс]. – Режим доступу: https://training.qatestlab.com/blog/technical-articles/pairwise-testing/.

8. Fang L., Li G. Test selection with equivalence class partitioning // 2015 2nd international symposium on dependable computing and internet of things (DCIT). – Wuhan, China, 16–18 листоп. 2015 р.

9. Класи еквівалентності, граничні значення [Електронний ресурс]. – Режим доступу: https://training.qatestlab.com/blog/technical-articles/equivalence-classes-and-boundary-values/ (дата звернення: 25.10.2023).

10. Аналіз причинно-наслідкових зв’язків як техніка тест-дизайну [Електронний ресурс]. – Режим доступу: https://training.qatestlab.com/blog/technical-articles/analysis-causality/ (дата звернення: 25.10.2023).

11. Topchii M. [Електронний ресурс]. – Режим доступу: https://ela.kpi.ua/bitstream/123456789/31357/1/Topchii_magistr.pdf.

12. Jones A. Tips for healthy page object classes, Angiejones.tech, 2020.

13. Rouvinez T. Comparison of active objects and the actor model. – Universite de neuch´atel, 2014.

14. Ляшенко Д. Автоматизоване тестування UI: стисло про головне. – Codeguida, 2021.

15. TensorFlow 101: Basics for beginners [Електронний ресурс]. – Режим доступу: https://medium.com/turingtalks/tensorflow-101-basics-for-beginners-43f619a4d72a.

16. Keras [Електронний ресурс]. – Режим доступу: https://boringowl.io/tag/keras.

17. REST APIs [Електронний ресурс]. – Режим доступу: https://www.ibm.com/topics/rest-apis.

18. Tomachynska V. S., Korsun D. M., Rozhnova T. H. Life cycle models, principles and methodologies of software development. Theory and practice of science: key aspects : Proceedings of the 7th International Scientific and Practical Conference, Rome, 20 December 2022. - 2022.- P. 394–401.

19. Томачинська В. С., Рожнова Т. Г. Алгоритм розробки фреймворку для автоматизованого тестування користувацького інтерфейсу. Міждисциплінарні наукові дослідження та перспективи їх розвитку : Міжнар. студ. наук. конф., м. Дніпро, 10 листоп. 2023 р. – Вінниця, 2023. – С. 110–112.

20. Томачинська В. С., Рожнова Т. Г. Метод попарного тестування з використанням ортогонального масиву. Комп’ютерної інженерії та захисту інформації : Матеріали 27-го Міжнар. Молодіж. форуму, м. Харків, 10 травня 2023 р. – Харків, 2023. – С. 57–58.