Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп’ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота магістра

на тему: «Розробка методики тестування програмного забезпечення у контексті ефективності та практичного застосування для різних сценаріїв у веб-додатках»

Виконав: студента групи К23-1М
Спеціальність 122 «Комп’ютерні науки»
Рудь А.Є.
(прізвище та ініціали)
Керівник к.т.н., доц. Чупілко Т.А.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент Університет митної справи та фінансів
(місто роботи)
доцент кафедри кібербезпеки та інформаційних технологій
(посада)
к.т.н Савченко Ю.В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Рудь А. Є. Розробка методики тестування програмного забезпечення у контексті ефективності та практичного застосування для різних сценаріїв у веб-додатках.

Кваліфікаційна робота на здобуття освітнього ступеня магістра за спеціальністю 122 «Комп'ютерні науки». – Університет митної справи та фінансів, Дніпро, 2024.

Об'єктом дослідження є процес тестування програмного забезпечення у веб-додатках.

Предметом дослідження методики тестування програмного забезпечення з акцентом на їхню ефективність та практичне застосування для різних сценаріїв у веб-додатках.

Мета роботи – розробка методики тестування веб-додатків для визначення найшвидшого та найякіснішого способу тестування.

Робота включає розробку методики тестування веб-додатків. Дослідження зосереджене на пошуку оптимальних підходів для забезпечення якості програмного забезпечення з урахуванням критеріїв ефективності, точності, витрат часу, а також рівня складності впровадження. У ході роботи розглянуто основні аспекти ручного тестування та автоматизованого тестування зокрема можливість адаптації до нестандартних сценаріїв, високу чутливість до деталей, які важко автоматизувати, та вплив людського фактору, що може бути як перевагою, так і недоліком. Також, у роботі досліджується, як комбінування ручного та автоматизованого тестування може забезпечити найкращі результати; перевіряються підходи до тестування UX/UI та нестандартних сценаріїв, регресійного тестування. Також вивчаються математично-технічні аспекти тестування: витрати на впровадження автоматизації, її доцільність у довгостроковій перспективі та вплив на загальний цикл розробки продукту.

Ключові слова: методи тестування, дослідження у тестуванні, ручне тестування, автоматизоване тестування, змішане тестування.

ABSTRACT

Rud A. E. Development of a software testing methodology in the context of efficiency and practical application for various scenarios in web applications.

Qualification work for obtaining a master's degree in specialty 122 "Computer Science". - University of Customs and Finance, Dnipro, 2024.

The object of the study is the process of software testing in web applications.

The subject of the study is software testing methodology with an emphasis on their efficiency and practical application for various scenarios in web applications.

The purpose of the work is to develop a methodology for testing web applications to determine the fastest and highest quality testing method.

The work includes the development of a methodology for testing web applications. The research focuses on finding optimal approaches to ensuring software quality, taking into account the criteria of efficiency, accuracy, time consumption, and the level of complexity of implementation. The paper examines the main aspects of manual testing and automated testing, including the ability to adapt to non-standard scenarios, high sensitivity to details that are difficult to automate, and the influence of the human factor, which can be both an advantage and a disadvantage. The paper also examines how combining manual and automated testing can provide the best results. Testing approaches to UX/UI testing and non-standard scenarios, regression testing. The mathematical and technical aspects of testing are also studied: the costs of implementing automation, its feasibility in the long term, and the impact on the overall product development cycle.

Keywords: testing methods, research in testing, manual testing, automated testing, mixed testing.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ПЗ – програмне забезпечення

UI/UX – User Interface/User Experience

CI/CD – Continuous Integration/Continuous Delivery

ШІ – Штучний інтелект

ВСТУП

Розвиток сучасних інформаційних технологій суттєво вплинув на створення та використання веб-додатків у різних сферах діяльності. Можливо навіть сказати, що вони стали невід'ємною частиною нашого світу. Веб-додатки застосовуються в бізнесі, фінансах, розвагах, освіті й охороні здоров'я, що зумовлює підвищені вимоги до їхньої стабільності, функціональності та надійності. Одним із ключових аспектів створення успішного веб-додатку є процес тестування, який забезпечує відповідність продукту технічним і користувацьким вимогам.

Тестування програмного забезпечення є обов'язковим етапом у процесі створення програмного продукту. У певному сенсі – це один із головних етапів після написання коду програми, тому не варто недооцінювати його важливості або вважати його непотрібним. Перевіряючи продукт на відповідність вимогам, необхідно спочатку визначити його недоліки, а вже потім оцінювати переваги.

На сьогодні існує безліч методів і класифікацій тестування програмного забезпечення. Особливу увагу в цій роботі приділено ручному тестуванню (manual testing) та автоматизованому тестуванню (automation testing).

Ручне тестування передбачає безпосередню участь тестувальника, який виконує всі перевірки вручну, використовуючи свої знання, інтуїцію та професійний досвід, а також метод забезпечує гнучкість у процесі тестування, дозволяючи адаптуватися до різних сценаріїв і перевіряти нестандартні кейси.

Даний вид тестування є незамінним для оцінки користувацького досвіду (UX/UI), оскільки враховує суб'єктивне сприйняття функціональності та дизайну. Попри свої переваги, ручне тестування має певні обмеження, тому його не можна вважати ідеальним підходом, але й недооцінювати не варто.

Автоматизоване тестування, у свою чергу, базується на використанні спеціалізованих інструментів і скриптів для виконання тестових сценаріїв, підхід дозволяє значно пришвидшити процес тестування, зменшити вплив людського фактора та забезпечити високу точність і повторюваність перевірок.

Даний вид тестування є ефективний для регресійного тестування, тестування продуктивності та роботи з великими обсягами даних, що робить її незамінною в умовах сучасної розробки програмного забезпечення.Проте автоматизоване тестування також має свої виклики. Воно вимагає значних початкових вкладень у розробку тестових скриптів, обмежене у перевірці творчих або нестандартних сценаріїв і потребує технічної компетенції команди тестувальників.

Актуальність порівняльного дослідження методів тестування полягає в необхідності визначення оптимального балансу між різними підходами для забезпечення якості веб-додатків та зменшення часу на перевірки. У багатьох випадках розробники стикаються з проблемою вибору відповідного методу тестування, який дозволить досягти високої якості продукту, мінімізуючи витрати часу й ресурсів.

Крім того, зростання складності веб-додатків і підвищення вимог користувачів потребують інтегрованого підходу до тестування, що поєднує переваги ручного та автоматизованого методів.

Мета дослідження – розробка ефективної методики використання видів тестування веб-додатків залежно від специфіки проєкту.

Для досягнення мети в роботі поставлено такі завдання:

1. Провести аналіз існуючих методів і видів тестування веб-додатків, їхніх переваг і недоліків.
2. Дослідити сучасні інструменти тестування.
3. Розглянути приклади практичного застосування методів тестування у проєктах.
4. Визначити критерії вибору методу тестування залежно від типу проєкту, його масштабу й ресурсів.
5. Розробити методику та рекомендації щодо комбінованого використання ручного й автоматизованого тестування для досягнення максимальної ефективності.

Об'єктом дослідження є процес тестування веб-додатків, *предметом дослідження* – методи ручного та автоматизованого тестування, їхні переваги, недоліки й особливості застосування.

Практичне значення отриманих результатів полягає в можливості використання запропонованих рекомендацій для підвищення ефективності тестування веб-додатків, скорочення часу на виявлення дефектів і покращення якості кінцевого продукту.

Наукова новизна роботи передбачає створення методики для коректного вибору методів тестування, розробку універсальної схеми роботи з програмним забезпеченням, а також інтеграцію штучного інтелекту для розширення й прискорення процесів тестування.

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел і додатків. У першому розділі розглянуто основи тестування веб-додатків, проаналізовано сучасні інструменти й технології. В другому розділі розглянути методи, які допомагають вирішити завдання кваліфікаційної роботи. У третьому розділі практично відображено методи тестування та можливості комбінування підходів для досягнення найкращого результату.

Робота містить 77 сторінок, 21 рисунок і 7 таблиць. Список літератури включає 25 джерел.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1. Сучасні підходи до розробки програмного забезпечення

Створення ПЗ є важливим процесом, відповідно до міжнародного стандарту ISO/IEC/IEEE 15288:2023[1] потрібно виділити основні моменти та окреслити мету даного документу. Основною метою даного стандарту є забезпечення структури для планування, управління та виконання процесів розробки та експлуатації систем у різних галузях.

Новий програмний продукт забезпечує інноваційні рішення для бізнесу, науки та повсякденного життя, сприяє розвитку економіки, освіти й медицини. На ринку програмного забезпечення сучасне та якісне ПЗ дозволяє автоматизувати рутинні процеси, оптимізувати витрати й підвищувати продуктивність праці, а також відіграє важливу роль у покращенні взаємодії між компаніями та клієнтами, забезпечуючи ефективну комунікацію через CRM-системи, електронну комерцію та онлайн-сервіси.

У науці ПЗ стає ключовим інструментом для моделювання, обробки великих даних і проведення складних розрахунків. Наприклад, у біології та медицині програмні рішення допомагають розробляти нові ліки, аналізувати генетичні дані та моделювати поведінку клітин. У фізиці та астрономії спеціалізовані додатки використовуються для симуляції явищ, які неможливо дослідити в лабораторних умовах.

В повсякденному житті додатки допомагають людям залишатися на зв’язку, керувати фінансами, навчатися та розважатися. Вони забезпечують доступ до інформації, можливість дистанційного навчання, здорового способу життя та навіть контролю за розумним будинком. Такі технології як штучний інтелект або інтернет речей інтегруються в повсякденні пристрої, роблячи їх більш зручними та функціональними.

Тобто, створення програмного забезпечення не лише відповідає потребам сучасності, а й визначає напрямок розвитку суспільства, сприяючи інноваціям та покращенню якості життя на всіх рівнях. Кожен етап розробки програмного забезпечення є критично важливим для успішного результату, оскільки вони формують єдину цілісну систему, де всі складові взаємопов’язані, приведемо на прикладі моделі Waterfall(Рисунок 1.1).

Waterfall модель – також відома як каскадний підхід, є одним із найдавніших і найкласичніших методів управління проєктами. Її ключова особливість – це лінійна та послідовна структура роботи, яка добре підходить для проєктів із чітко визначеними цілями та вимогами.

Рисунок 1.1– Етапи створення ПЗ

Формування вимог до ПЗ – на початковому етапі розробки проводиться детальне вивчення потреб клієнтів і користувачів, що дає змогу сформулювати вимоги до програмного забезпечення, окреслити функціональність і встановити

ключові цілі проекту. На цьому етапі важливо врахувати всі бізнес-процеси, які потрібно автоматизувати, і специфічні потреби замовника.

Проектування передбачає створення архітектури майбутнього програмного забезпечення. Визначаються основні модулі, структура даних, механізми взаємодії компонентів і розподіл ресурсів. Результатом цього етапу є технічна документація, яка служить основою для подальшої розробки.

Етап реалізації включає написання коду й створення функціональних компонентів системи. Розробники інтегрують усі модулі, тестують їхню взаємодію та забезпечують відповідність вимогам. Даний етап вимагає залучення команди досвідчених інженерів і використання сучасних інструментів.

Тестування проводиться для перевірки функціональності, продуктивності, безпеки та сумісності програмного продукту. Використовуються ручні та автоматизовані методи, які дозволяють знайти дефекти на ранніх етапах і запобігти їхньому впливу на кінцевий продукт.

На етапі введення в дію програмне забезпечення готується до впровадження у виробниче середовище. Включає перевірку готовності системи до експлуатації, навчання персоналу та безпосереднє розгортання продукту для користувачів.

Після запуску продукту в експлуатацію важливо забезпечити його стабільну роботу та супровід продукту. Команда розробників усуває дефекти, додає нові функції та адаптує систему до змін у бізнес-процесах або технологіях. Даний етап гарантує довготривалу ефективність програмного забезпечення.

Як можливо побачити, то кожен етап залежить один від одного, і кожний наступний етап ґрунтується лише на успішному результаті попереднього, що забезпечує послідовність і логічність у процесі розробки, де кожен крок сприяє побудові надійного фундаменту для наступного.

Без якісного аналізу вимог неможливо правильно спроектувати архітектуру програмного забезпечення, а без чіткого проектування розробка може стати хаотичною.

Особливу увагу у кваліфікаційній роботі потрібно приділити етапу тестування. Тестування ПЗ — це процес перевірки програмного продукту з метою виявлення дефектів, помилок та недоліків перед його випуском на ринок або в експлуатацію. Даний процес охоплює запуск програми з різними вхідними даними та умовами, а також аналіз реакції програми на ці дані.

Мета тестування — це підтвердження правильності роботи програми відповідно до вимог до неї, а також забезпечення високої якості та надійності програмного продукту.

Тестування ПЗ, відповідно до стандарту IEEE 829—1998[2] також ділиться на етапи(Рисунок 1.1Рисунок 1.2), кожен з яких допоможе побачити особливості та допомагає розподілити час на якісне та глибоке тестування програмного продукту.

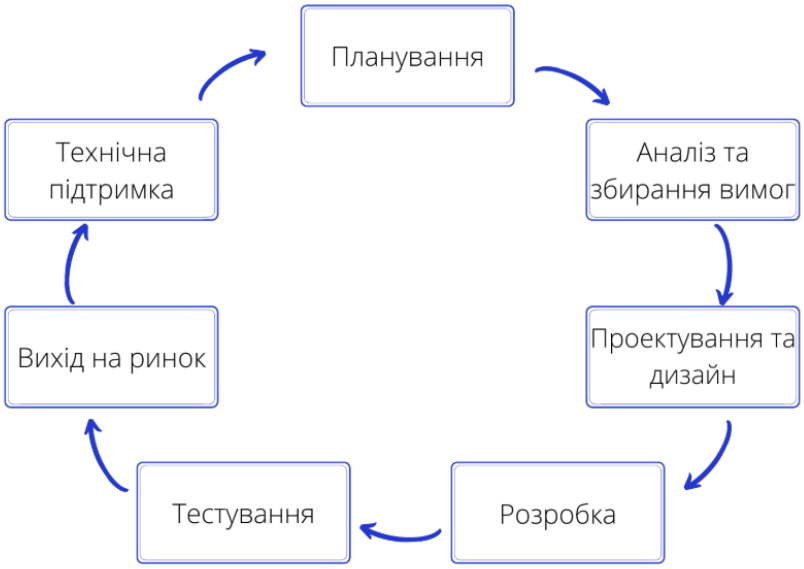


Рисунок 1.2– Етапи тестування ПЗ

На етапі планування визначаються основні цілі проєкту, його бюджет, часові рамки та необхідні ресурси. Команда розробників разом із замовником обговорює ключові завдання, які потрібно вирішити, та створює початковий план дій. Грамотне планування формує основу для подальшого успіху проєкту.

Аналіз та збирання вимог передбачає детальне вивчення потреб замовника та кінцевих користувачів. Збираються всі вимоги до функціональності, інтерфейсу та продуктивності майбутнього продукту. На основі цих даних створюється документація, яка стане базою для роботи розробників і дизайнерів.

На етапі проєктування архітектури ПЗ та дизайну розробляються макети інтерфейсу, визначаються технології, які будуть використовуватися. Ретельне опрацювання деталей на цьому етапі дозволяє уникнути значних помилок у майбутньому.

Етап розробки, на якому команда розробників реалізує задумане у вигляді програмного коду. Завдяки роботі розробників створюється основний функціонал продукту. Розробка часто поділяється на ітерації, щоб контролювати прогрес і забезпечити якість виконання.

Перевірка якості програмного забезпечення є важливим етапом для виявлення помилок і недоліків. Тестувальники виконують різні види тестування, щоб переконатися, що програма відповідає вимогам замовника, працює стабільно та надійно.

Вихід на ринок передбачає передачу програмного забезпечення замовнику та впроваджується в робоче середовище. На цьому етапі забезпечується підготовка документації, підтримка користувачів і навчання персоналу, якщо це передбачено.

Після виходу на ринок продукт супроводжується технічною підтримкою. Вона включає оновлення, усунення можливих помилок і вдосконалення продукту, щоб забезпечити його відповідність новим потребам і зберігати актуальність.

1.2. Види та методи тестування програмного забезпечення

Обирання видів та методів тестування програмного забезпечення є одним із ключових моментів у забезпеченні його якості та відповідності поставленим вимогам. У сучасному світі, де програмні системи стають дедалі складнішими,

роль тестування як невіддільної частини життєвого циклу розробки ПЗ постійно зростає.

Саме тестування дозволяє не лише виявити помилки на ранніх етапах розробки, але й мінімізувати ризики, пов’язані з їх впливом на користувачів кінцевого продукту. Існує широкий спектр видів та методів тестування, кожен із яких має свої особливості, переваги та недоліки. Наприклад, функціональне тестування. Функціональне тестування – один із видів тестування, спрямованого на перевірку відповідностей функціональних вимог ПЗ його реальним характеристикам. Основним завданням функціонального тестування є підтвердження того, що програмний продукт, який розробляється, володіє усім необхідним замовнику функціоналом.

Якщо ми маємо функціональне тестування, тоді важливо згадати і про нефунціональне тестування, яке також відіграє важливу роль у перевірках на відповідність очікуваного результату з фактичним. Нефункціональне тестування – тип тестування програмного забезпечення, яке виконується для перевірки нефункціональних вимог продукту. Воно перевіряє, чи відповідає поведінка системи вимогам, чи ні. Даний тип тестування – широкий термін, який містить усі аспекти, не пов’язані з функціональністю системи.

Ефективний вибір підходів до тестування дозволяє оптимізувати процес розробки та гарантувати, що кінцевий продукт відповідатиме як технічним, так і бізнес-вимогам. У цьому контексті аналіз видів і методів тестування є важливим завданням для кожної команди розробників, адже саме від нього залежить успіх проєкту та його подальше використання.

Методи тестування визначають підхід до перевірки програмного забезпечення і можуть бути класифіковані залежно від доступу до внутрішньої структури системи. Окрім, методів, які було згадано раніше треба навести класифікацію всіх методів тестування в цілому, щоб розумітись, як і коли треба використовувати відповідний метод у тестуванні(Рисунок 1.1Рисунок 1.2Рисунок 1.3).

Основна класифікація методів тестування:

- За знанням внутрішньої системи;
- За об’єктом тестування;
- За суб’єктом тестування;
- За часом проведення тестування;
- За критерієм позитивності тестування;
- За степенем ізольованості;
- За степенем виконання;
- За степенем підготовки.

За знанням внутрішньої системи тестування поділяється на такі методи:

- Метод “чорної скриньки” – метод передбачає тестування без знання внутрішньої структури коду або логіки роботи програми.
- Метод “білої скриньки” – передбачає тестування з повним доступом до внутрішньої структури коду.
- Метод “сірої скриньки” – поєднує елементи "чорної скриньки" та "білої скриньки".

Рисунок 1.3– Види та методи тестувань

За об’єктом тестування методи поділяються на фунціональне та нефункціональне тестування.

- UI тестування – перевірка якості інтерфейсу користувача для коректної взаємодії ПЗ з клієнтом.
- Тестування локалізації – перевірка коректного перекладання змісту тексту на обрану клієнтом мову.
- Тестування навантаження на застосунок – перевірка, чи коректно працює застосунок в умовах навантаження на ПЗ.
- Тестування продуктивності – перевіряє стабільність та дослідження показників на зовнішні зміни.
- Тестування безпеки – перевіряє наявність захисту даних користувача від зовнішнього впливу.
- Тест сумісності – перевіряє роботу ПЗ у різних середовищах, операційних системах.

За суб’єктом тестування методи поділяються на:

- Альфа тестування – перевірка продукту після його розробки на коректність роботи працівниками компанії (тестувальниками або розробниками).
- Бета тестування – перевірка продукту зовнішніми користувача або залученою групою осіб.

За часом проведення тестування методи поділяються на:

- Димове тестування – перевірка основних моментів роботи ПЗ, без заглибленого перегляду.
- Санітарне тестування – поглиблення перевірка основних моментів роботи ПЗ.
- Приймальне тестування – перевірка на готовність продукту до релізу.
- Регресивне тестування – повторна перевірка ПЗ, по попередньо написаній документації.

За критерієм позитивності сценаріїв методи поділяються на:

- Позитивний – перевірка коректної поведінки вимог до ПЗ.
- Негативний – перевірка некоректної поведінки вимог до ПЗ.

За степенем ізольованості методи поділяються на:

- Компонентний – перевірка конкретного компоненту ПЗ (наприклад: перевірка кнопки, чек-боксу і т.д).
- Інтеграційний – перевірка взаємодії декількох компонентів ПЗ між собою (наприклад: взаємодія кнопки з текстовим полем).
- Системний – перевірка продукту в цілому, чи коректно працюють та взаємодію

За степенем виконування методи поділяються на:

- Ручний – тестування ПЗ, яке передбачає написання тестової документації та виконання тестувань
- Автоматизований – тестування ПЗ, яке передбачає написання скрипту для виконання тестувань.
- Змішаний – тестування ПЗ, яке передбачає принципи застосування автоматизованого та ручного тестування.

За степенем підготовки до тестування методи поділяється на:

- Формальний – тестування ПЗ з підготовленою тестовою документацією, наявність очікуваних результатів.
- Ед-хок – тестування ПЗ без підготовленої тестовою документації та без наявність очікуваних результатів.

Тестування ПЗ передбачає велику кількість методів, які можливо використовувати, головне, зробити правильну класифікацію та мати розуміння, коли та як саме треба починати етап тестування.

1.3. Аналіз та дослідження тестової документації

Тестова документація є фундаментом будь-якого процесу тестування. Вона забезпечує структурованість, послідовність і точність виконання тестів, а

також дозволяє контролювати якість продукту. Без належної документації тестування може стати неефективним, хаотичним і нездатним охопити всі вимоги до системи. У цьому розділі проаналізуємо ключові типи тестової документації, їх значення для ручного, автоматизованого та змішаного тестування, а також надамо чек-ліст для оцінки її якості. Типи тестової документації:

- План тестування (Test Plan) – це документ, який описує увесь об’єм робіт пов’язаних із тестуванням.

План тестування є важливим етапом у процесі тестування, оскільки дозволяє чітко сформулювати цілі, ресурси та терміни для проведення тестування, а також забезпечує узгодженість усіх етапів тестування серед учасників проекту. Він визначає, що саме буде протестовано, яким чином це буде зроблено, і які результати повинні бути досягнуті.

Є основою для організації всього процесу тестування та є важливим інструментом для контролю якості. Документ служить комунікаційним мостом між усіма учасниками проекту — від розробників до тестувальників і замовників, забезпечуючи розуміння цілей, завдань і методів тестування.

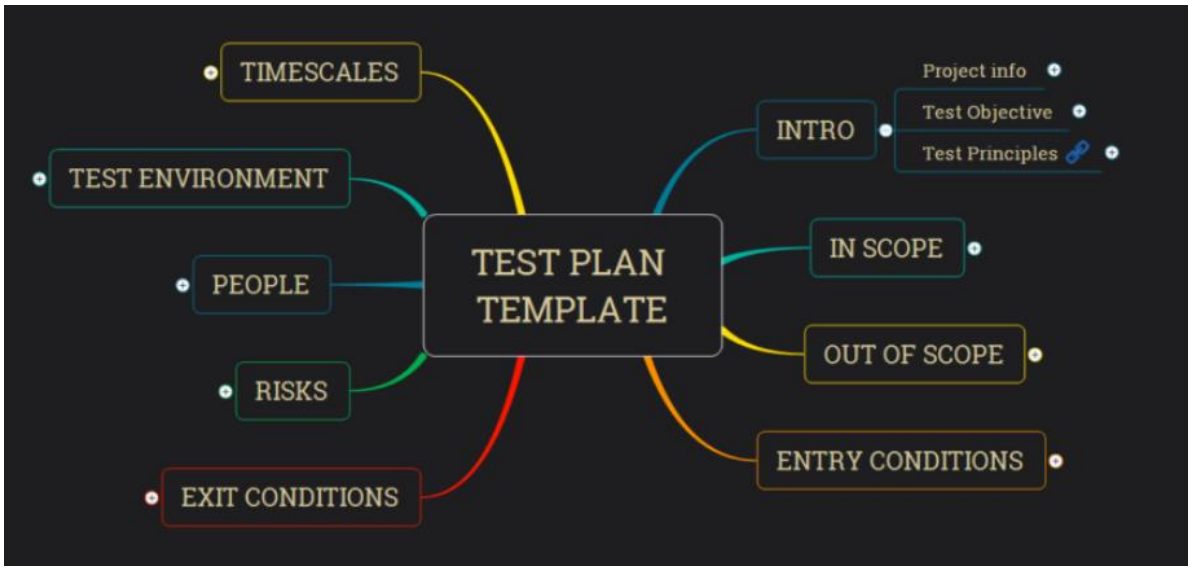


Рисунок 1.4 – Структура тест-плану

Правильно складений план тестування дозволяє ефективно розподіляти ресурси, встановлювати реалістичні терміни та забезпечити належний контроль за виконанням тестування. Він також допомагає визначити критерії для оцінки результатів тестування, що, в свою чергу, дозволяє приймати обґрунтовані рішення щодо якості програмного забезпечення. План тестування є важливим інструментом для зменшення ризиків, пов'язаних із виявленням дефектів на пізніх етапах розробки, оскільки дозволяє передбачити та запобігти потенційним проблемам ще до початку тестування.

- Тест-кейс (Test Case) – це документ, який описує увесь об’єм робіт пов’язаних із тестуванням. Тест-кейси допомагають забезпечити точність і повторюваність тестування, а також дозволяють ефективно відслідковувати виявлені дефекти.

Основна структура тест-кейсу включає такі елементи, як унікальний ідентифікатор, опис тесту, попередні умови, кроки виконання, очікувані результати, фактичні результати та статус тесту, дані елементи дозволяють чітко визначити, що саме тестується, і яких результатів слід очікувати.

Таблиця 1.1

Дія	Очікуваний результат	Фактичний результат
Передумови		
Дія 1	Перевірено «Дія 1»	passed
Дія 2	Перевірено «Дія 2»	failed
Опис тест-кейсу		
Дія 3	Перевірено «Дія 3»	blocked
Постумова		

Використання тест-кейсів дозволяє значно покращити ефективність тестування, зменшити ймовірність пропуску помилок і забезпечити високу якість програмного продукту.

- Чек-лист (Check-list) – це частина плану тестування, конкретний список того, що потрібно перевірити. Вони зазвичай використовуються для перевірки загальних аспектів функціональності, таких як наявність помилок у інтерфейсі, коректність введення даних або відповідність вимогам.

Основною перевагою чек-листів є їхня простота та зручність, оскільки вони дозволяють тестувальникам швидко перевіряти основні функціональні можливості без необхідності виконувати детальне тестування. Чек-лісти також допомагають зберегти організованість і уникнути пропуску важливих перевірок.

Таблиця 1.2

	Іванов А. О.	Руденко Б. В.	Федоров А. А
Сайт “example.ua”	Google Chrome 91.0	Mozilla Firefox 89.01	Opera 77.0
Розділ перевірки «Дія 1»			
Функціонал 1 на «Дія 1»	passed	failed	failed
Функціонал 2 на «Дія 1»	failed	failed	passed
Розділ перевірки «Дія 2»			
Функціонал 1 на «Дія 2»	failed	failed	passed
Функціонал 2 на «Дія 2»	passed	failed	failed
Функціонал 3 на «Дія 2»	passed	passed	passed

- Баг-репорт (Bug Report) - це технічний документ, що описує ситуацію або послідовність дій, яка призвела до некоректної роботи об'єкту тестування, із зазначенням причин і очікуваним результатом.

Крім того, баг-репорт повинен містити інформацію про очікуваний і фактичний результати роботи системи, що допомагає розробникам точніше зрозуміти суть дефекту. Важливими також є дані про середовище тестування (операційна система, браузер, версія програмного забезпечення тощо) та пріоритет і серйозність дефекту, які визначають порядок його виправлення. Окрім цього, баг-репорт має містити кроки для відтворення помилки, щоб забезпечити чітке розуміння того, як саме можна повторити проблему, що

дозволяє не тільки швидше виявити дефект, але й забезпечити його ефективне усунення.

Також варто вказати можливі умови, при яких дефект може виникнути, та екранні знімки або відео, що наочно демонструють проблему. Деталізація кроків допомагає уникнути непорозумінь між тестувальниками та розробниками, що може значно прискорити процес виправлення помилки.

Додатково, баг-репорт має включати інформацію про ступінь відтворюваності дефекту, зокрема, чи виникає він при кожному виконанні певних дій, чи випадковий. Це важливо для розуміння того, наскільки дефект критичний для системи. Також важливо вказати, чи є обмеження по часу для виправлення багу, особливо якщо дефект впливає на роботу користувачів або на важливі функції додатка.

Bug Report: #Bug-ID

Bug ID	***#Bug-ID (e.g. #test_ABC, #123, #home_123)—***
Tester	*Han Solo*
Date (submitted)	*01.11.2018*
Title	*CONTACT FORM - No confirmation message is shown*
Bug Description	
URL	*https://milleniumfalcon.rebells/contact-us*
Summary	The page where the contact form is on does not show any confirmation message after submitting a contact request.
Screenshot	see attached screenshots
Platform	Mac OS 10.14
Browser	Chrome 69.0.3497.100 (Official Build) (64-bit)
Administrative	
Assigned To	*Chewbacca*
Assigned At	*01.11.2018*
Priority	High
Severity	Critical

Рисунок 1.5– Структура баг-репорту

Баг-репорти полегшують комунікацію між тестувальниками та розробниками, забезпечуючи швидке виявлення та усунення помилок. Чіткість і точність у їхньому складанні допомагають заощадити час і ресурси команди, мінімізуючи непорозуміння та знижуючи ризик залишення дефектів у продукті. Вони є ключовим елементом у забезпеченні високої якості програмного забезпечення. Допомагають виявляти і усувати помилки, покращувати користувацький досвід і забезпечувати довготривалу підтримку програмних продуктів. Також варто враховувати, що документація має важливий вплив на кожний метод тестування по своєму:

- Для ручного тестування вона зазвичай включає детальні тест-кейси, сценарії тестування, чеклисти та інструкції для виконання тестів. Дані документи дозволяють тестувальникам чітко розуміти, які дії потрібно виконати, які дані використовувати та які очікувані результати. Документація також допомагає уникнути пропусків у перевірці функціоналу, зменшує ризик помилок через людський фактор та забезпечує прозорість процесу для всіх учасників проєкту. Завдяки ній навіть нові члени команди можуть швидко адаптуватися до тестування.
- Для автоматизації тестування документація стає основою для створення скриптів. Тестові плани та специфікації допомагають визначити, які сценарії варто автоматизувати, а які залишити для ручної перевірки. Технічна документація також містить вимоги до середовища виконання, конфігурацій і використовуваних інструментів. Документація часто включає докладні інструкції щодо написання тестових скриптів, формати входу й виходу даних, а також критерії успішності тестів. Крім того, результати виконання автоматизованих тестів записуються у звіти, що також є частиною документації, яку можна використовувати для аналізу дефектів та оптимізації процесу.
- У змішаному тестуванні документація виконує подвійну функцію. Вона має включати інструкції для ручного тестування окремих аспектів, таких як візуальна перевірка інтерфейсу або складні інтерактивні сценарії.

Паралельно створюється документація для автоматизованих тестів, яка описує скрипти, їх виконання та аналіз результатів. Спрогнозувати важливість ШІ в майбутньому для створення тестової документації та її підтримці. Основні можливості ШІ у питаннях тестової документації:

- ШІ може аналізувати вимоги та генерувати тест-кейси на їх основі.
- ШІ допомагає знайти прогалини в тестовій документації та пропонує додаткові сценарії.
- На основі аналізу дефектів ШІ визначає, які тести найбільш критичні.

Тестова документація є основою якісного тестування, незалежно від обраного підходу. Чіткі, структуровані й актуальні документи допомагають уникнути помилок, забезпечують прозорість процесу тестування та полегшують аналіз результатів. Інтеграція ШІ у створення тестової документації дозволяє підвищити ефективність і точність, відкриваючи нові можливості для оптимізації процесу.

1.4. Формування задачі дослідження кваліфікаційної роботи

Першим етапом є глибокий аналіз проблемної області, що охоплює досліджувану тему. У випадку тестування програмного забезпечення це може бути аналіз існуючих методів та підходів до тестування, вивчення новітніх інструментів автоматизації, а також виявлення проблем і викликів, з якими стикаються розробники та тестувальники. На цьому етапі важливо зібрати максимум інформації з наукових публікацій, технічної документації, реальних кейсів та практичного досвіду. Особлива увага повинна приділятися сучасним тенденціям в області тестування, таким як інтеграція автоматизованих тестів у процеси CI/CD. CI/CD – неперервна доставка. Даний набір методик дозволяє розробникам частіше і надійніше розгортати зміни в програмному забезпеченні.

На етапі проведення аналізу формулюються основні проблеми та прогалини, які потребують вирішення. Наприклад, у сфері тестування це можуть бути недостатня ефективність існуючих підходів до ручного тестування, висока

вартість автоматизації тестування, проблеми з адаптацією інструментів до різних платформ або недосконаліть методів оцінки ефективності тестування. Визначення цих проблем дозволяє чітко окреслити межі дослідження та визначити, які саме аспекти потребують найглибшого вивчення.

Метою дослідження є відповідь на питання “Як ефективніше буде провести тестування ПЗ?”,” Яким методом краще провести тестування ПЗ?”,” Скільки займе часу тестування ПЗ?”. Мета даної кваліфікаційної роботи може бути розробка єдиної методики оцінки ефективності ручного та автоматизованого тестування, створення нових інструментів для автоматизації або визначення оптимального співвідношення між ручним і автоматизованим тестуванням для різних типів проектів.

Завдання дослідження є конкретизацією мети і виступають як детальне розбиття на окремі етапи, які необхідно виконати для досягнення поставленої мети. Сформулювавши завдання, дослідник отримує чітку інструкцію до дій, що допомагає зберегти фокус на ключових аспектах дослідження та уникнути розпорошення уваги на менш важливі питання.

Задачі дослідження у сфері тестування програмного забезпечення можуть варіюватися в залежності від конкретного аспекту, на якому зосереджено дослідження. Формулювання чітких та конкретних завдань підходу до тестування ПЗ різними методами, як ручним, автоматичним та змішаним є необхідним для забезпечення належної організації та управління процесом дослідження, а також для досягнення високих результатів у розв'язанні поставленої проблеми.

1.5. Висновки до першого розділу

Перший розділ даної роботи присвячено теоретичним основам розробки та тестування програмного забезпечення, що є фундаментом для подальшого вивчення практичних аспектів цієї сфери. Розглядаються основні методи, рівні

та інструменти тестування, які дозволяють забезпечити високу якість програмних продуктів.

Теоретична частина включає аналіз різних підходів та методів тестування, а також основні принципи, які лежать в основі ефективної стратегії тестування. Даний розділ не лише висвітлює основні типи тестування, такі як функціональне, нефункціональне, ручне та автоматизоване тестування, але й аналізує їх застосування в умовах розробки програмного забезпечення.

Методи тестування, наведені у цьому розділі, мають свої унікальні характеристики, переваги та обмеження, які роблять їх придатними для різних типів завдань. Наприклад, метод "чорної скриньки" дозволяє тестувальникам перевіряти функціональність системи без необхідності знання її внутрішньої структури, тоді як метод "білої скриньки" забезпечує детальний аналіз внутрішніх процесів коду.

Окремо було розглянуто поняття нефункціональних аспектів тестування, таких як продуктивність, зручність і безпечність. Вивчення цих аспектів є критично важливим для забезпечення стабільної роботи програмного забезпечення в умовах високого навантаження та у випадках, коли система повинна масштабуватися для підтримки великої кількості користувачів.

Аналізуються сучасні виклики, з якими стикаються тестувальники, що включає необхідність тестування на різних платформах і пристроях, швидке реагування на зміни у вимогах проекту, а також забезпечення автоматизації тестування для підвищення його ефективності.

Важливим елементом є те, що ефективне тестування програмного забезпечення потребує комбінованого підходу, який включає як ручне, так і автоматизоване тестування. Ручне тестування залишається незамінним для оцінки користувацького досвіду та виявлення специфічних дефектів, тоді як автоматизація є ідеальним вибором для рутинних завдань і перевірок великого обсягу даних.

На основі проведеного аналізу можна зробити висновок, що тестування програмного забезпечення є складним і багатогранним процесом, який потребує глибоких знань, досвіду та використання сучасних технологій.

Розуміння методів, рівнів і інструментів тестування дозволяє ефективно планувати та виконувати процеси забезпечення якості, що сприяє створенню надійного та функціонального програмного продукту. Врахування різних підходів до тестування, таких як ручне, автоматизоване та змішане тестування, дозволяє не лише зменшити ймовірність виникнення помилок, але й значно прискорити процес виявлення та виправлення дефектів.

Кожен з методів тестування має свої переваги та недоліки, що визначають їх доцільність у конкретних умовах. Використання автоматизації тестування, зокрема, дозволяє значно знизити час на повторювані операції, підвищуючи ефективність процесу та зменшуючи людський фактор.

Однак ручне тестування залишається необхідним для оцінки користувацького досвіду та тестування складних сценаріїв, де автоматизація може бути неефективною. Тому комбінування цих методів є оптимальним шляхом до досягнення високої якості програмного забезпечення.

РОЗДІЛ 2. ОЦІНКА МЕТОДІВ ТА ПІДХОДІВ ВИРІШЕННЯ ЗАВДАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

2.1. Застосування методів вирішення завдання кваліфікаційної роботи

Метод ручного тестування ПЗ передбачає, що тестувальник безпосередньо взаємодіє з програмним продуктом, виконуючи певні тестові сценарії. Даний підхід дозволяє оцінити функціональність системи, її відповідність вимогам, а також зручність використання для кінцевих користувачів. Незважаючи на розвиток автоматизації, ручне тестування залишається незамінним у багатьох ситуаціях через свою гнучкість і здатність адаптуватися до змінних умов.

Даний метод має низку переваг, які роблять його важливим інструментом у розробці програмного забезпечення. Перш за все, воно дозволяє виявляти помилки, пов’язані з функціональністю, які можуть бути проігноровані автоматизованими системами. Тестувальник, працюючи з інтерфейсом, може оцінити його зручність і логічність, що особливо важливо для продуктів, орієнтованих на кінцевого користувача.

Крім того, ручне тестування дозволяє швидко перевірити нові зміни в програмі без необхідності створення чи оновлення автоматизованих скриптів. Можливість тестування в умовах, наближених до реальних. Наприклад, перевірка поведінки додатку на різних пристроях або в різних браузерах. Тестувальник може використовувати програмне забезпечення так, як це робив би звичайний користувач, що підвищує шанси виявлення помилок, які можуть залишитися непоміченими під час автоматизованої перевірки. Завдяки цьому тестування стає більш гнучким і адаптивним до змін у процесі розробки.

Однак даний метод тестування має і свої недоліки. Одним з найбільш суттєвих є його трудомісткість. Проведення тестів вручну вимагає значних часових витрат, особливо для великих проектів. Крім того, ручний підхід залежить від людського фактору: тестувальник може пропустити помилки через

втому або неуважність, що робить процес менш надійним порівняно з автоматизацією.

Якщо проект вимагає перевірки великої кількості функцій або сценаріїв, ручне тестування стає вкрай неефективним. Для таких випадків автоматизація виграє за рахунок швидкості виконання та повторюваності тестів. Однак навіть автоматизовані системи не здатні повністю замінити мануальне тестування там, де потрібна оцінка користувацького досвіду чи перевірка складних інтерактивних елементів.

Тестувальник відіграє центральну роль у процесі ручного тестування. Він не лише виконує тест-кейси, але й аналізує результати, виявляючи можливі причини дефектів. Часто тестувальник бере участь у розробці тестової стратегії, визначаючи найбільш критичні аспекти програмного забезпечення, які потребують перевірки.

Крім того, тестувальник виступає посередником між командою розробників і кінцевими користувачами. Його завдання — забезпечити, щоб програмне забезпечення відповідало очікуванням користувачів і працювало стабільно в реальних умовах. Процес ручного тестування включає кілька ключових етапів, кожен з яких має свою специфіку та значення:

- Планування тестування – на цьому етапі визначається стратегія тестування, обираються цілі та завдання. Формуються основні критерії якості, які будуть перевірятися, а також сценарії тестування.
- Розробка тест-кейсів, вони детально описують кожен крок, який тестувальник повинен виконати, а також очікувані результати – етап вимагає глибокого розуміння функціональності системи.
- Тестувальник вручну проходить кожен тест-кейс, записуючи результати. Виявлені помилки фіксуються у спеціалізованих системах, наприклад такої як JIRA.

JIRA — це система управління проєктами, яка дозволяє закривати майже всі завдання PM-а в рамках одного інструмента: від планування до контролю процесів та результатів.

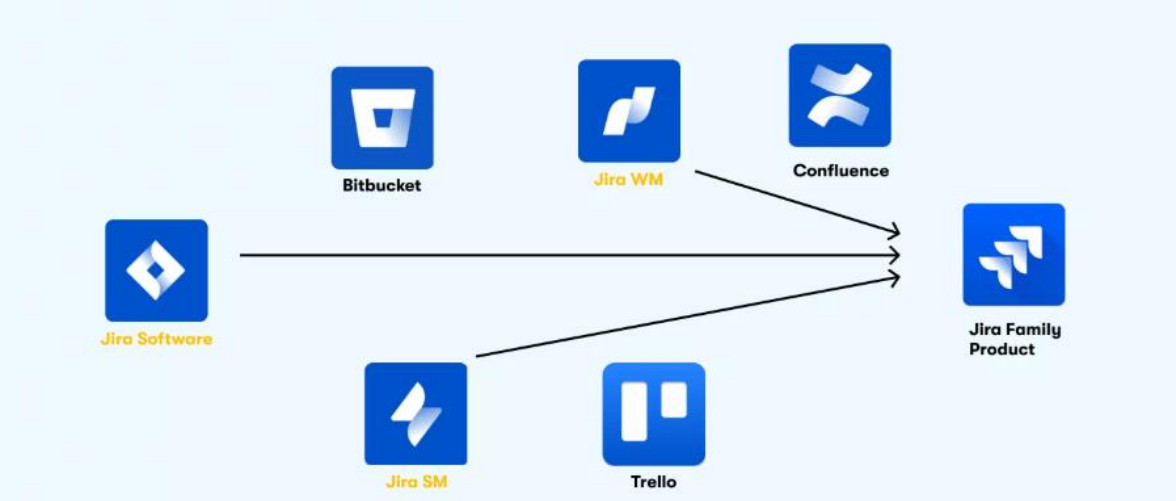


Рисунок 2.1– Jira та його компоненти

Після завершення тестування зібрані дані аналізуються, щоб визначити, чи відповідає продукт встановленим вимогам. Виявлені дефекти пріоритизуються для їх подальшого усунення. Ручне тестування дозволяє перевіряти різноманітні аспекти роботи програмного забезпечення:

- Функціональність – перевіряє, чи працює програма відповідно до технічних вимог, що включає тестування основних функцій, обробку помилок і сценарії крайніх значень.
- Сумісність, що включає перевірку роботи програми на різних платформах, браузерах або пристроях.
- Продуктивність – використовуватися для оцінки часу відгуку системи в реальних умовах, хоча для детального аналізу зазвичай застосовуються інструменти автоматизації.

Одним із найбільших викликів для ручного тестування є масштабованість. Для великих проектів, де кількість тест-кейсів обчислюється сотнями або тисячами, ручне тестування стає надто трудомістким, що особливо актуально для проектів із частими оновленнями, де необхідно регулярно проводити перевірки.

Інший виклик – це суб’єктивність результатів. Тестувальники можуть мати різний рівень досвіду, що впливає на якість виконання тестів. До того ж, людський фактор завжди залишається джерелом потенційних помилок.

Документація тестування є невід’ємною частиною процесу. Вона забезпечує прозорість роботи та дозволяє повторювати перевірки. Документація не лише підтримує високу якість роботи, але й дозволяє новим членам команди швидко зануритися в проект.

Отже, ручне тестування є основою для навчання нових спеціалістів у сфері QA. Воно дозволяє їм зрозуміти основні принципи перевірки програмного забезпечення, навчитися створювати тест-кейси, аналізувати дефекти та співпрацювати з командою.

Автоматизоване тестування стає дедалі популярнішим методом забезпечення якості програмного забезпечення, оскільки воно дозволяє значно оптимізувати процес тестування. Використовуючи сучасні інструменти, можна не лише прискорити перевірку функціональності, але й підвищити точність результатів. Даний метод тестування стає особливо важливим для великих проєктів, де обсяг тестових сценаріїв перевищує можливості ручної перевірки.

Автоматизація тестування базується на використанні спеціальних програмних інструментів, які дозволяють створювати скрипти для перевірки різних аспектів роботи програмного забезпечення. Основні особливості автоматизованого тестування включають:

- Повторюваність – написаний тестовий сценарій може виконуватися необмежену кількість разів без змін, забезпечуючи стабільність перевірок.
- Масштабованість – дозволяє перевіряти одночасно кілька середовищ, платформ або пристроїв.
- У порівнянні з ручним тестуванням, автоматизація дозволяє виконувати тести в рази швидше.
- Помилки, пов’язані з людським фактором, практично виключаються завдяки стандартизованому виконанню скриптів.

Процес автоматизованого тестування складається з кількох основних етапів, які забезпечують його ефективність:

- Аналіз потреб проєкту. На цьому етапі оцінюються потреби замовника, визначаються ключові функції, які потребують автоматизації, а також вимоги до інструментів.
- Вибір інструментів. Інструменти автоматизації, наприклад як Selenium та обираються залежно від специфіки проєкту. Selenium – це масштабний open source проект, а точніше, browser automation framework, у межах якого розробляється серія програмних продуктів для автоматизованого тестування, зазвичай використовуються для тестування веб-додатків.
- Планування тестування. Формулюються цілі автоматизації, визначається набір тестових сценаріїв для реалізації, створюється план їх реалізації.
- Розробка сценаріїв – етап включає написання тестових скриптів, які реалізують заплановані перевірки.
- Виконання тестів – тестові скрипти запускаються автоматично, результати зберігаються у вигляді звітів.
- Аналіз результатів – отримані дані аналізуються, виявлені дефекти фіксуються, проводяться оптимізації.

Переваги автоматизованого тестування:

- Ефективність – дозволяє швидко перевіряти великі обсяги даних, що знижує витрати часу.
- Точність і послідовність – виконуються ідентично кожного разу, що забезпечує стабільність результатів.
- Автоматизовані тести ідеально підходять для перевірки стабільності системи після оновлень.
- Інструменти автоматизації часто автоматично генерують звіти про виконання тестів.

Попри численні переваги, автоматизація має певні виклики:

- Розробка тестових сценаріїв вимагає інвестицій часу та ресурсів.
- Вибір неправильного інструменту може знизити ефективність тестування.
- Автоматизація не завжди підходить для перевірки нестандартних або творчих підходів до роботи системи.

Документація є невід'ємною складовою процесу автоматизації. Вона охоплює:

- Опис тестових сценаріїв.
- Інструкції для налаштування середовища виконання тестів.
- Звіти про виконання тестів із зазначенням виявлених помилок.

Якісна документація забезпечує прозорість процесу, дозволяє ефективніше адаптувати скрипти до змін у проєкті, а також полегшує інтеграцію нових членів команди.

Автоматизоване тестування у рамках кваліфікаційної роботи дозволяє не лише показати знання теорії, але й продемонструвати практичні навички роботи з інструментами, що відкриває перспективи для подальших досліджень, таких як оптимізація існуючих підходів, інтеграція автоматизації на ранніх етапах розробки чи розробка нових інструментів. Даний метод тестування значною мірою перевищує ручне за швидкістю, точністю та можливістю масштабування. Однак у порівнянні з ручним тестуванням, автоматизація має певні обмеження. Наприклад, вона не може замінити людську інтуїцію у виявленні дефектів користувацького інтерфейсу або незручностей у використанні продукту.

У той же час ручне тестування залишається актуальним для перевірки нестандартних сценаріїв та дослідження поведінки програми в умовах, які не були передбачені.

Автоматизація – стає невід'ємною частиною життєвого циклу розробки програмного забезпечення, адже вона дозволяє:

- Знижувати ризики помилок на ранніх етапах.
- Гарантувати стабільність продукту перед релізом.

- Прискорювати впровадження нових функцій.

Практичне значення автоматизації у тестування в реальних проєктах – дозволяє значно знизити витрати часу на перевірку стабільності програмного забезпечення. Наприклад, у випадку розробки веб-додатків автоматизовані тести можуть виконуватися щоразу після внесення змін у код, забезпечуючи впевненість у стабільності системи.

З кожним роком, автоматизоване тестування розвивається, стаючи ще більш гнучким і доступним. Одним із ключових напрямків – є впровадження штучного інтелекту та машинного навчання. Дані технології дозволяють автоматизувати навіть ті процеси, які раніше потребували втручання людини, наприклад:

- Автоматичне написання тестових сценаріїв.
- Ідентифікація дефектів за поведінкою системи.
- Оптимізація тестового покриття.

Крім того, розвиток хмарних технологій сприяє зниженню вартості автоматизації. Хмарні платформи дозволяють запускати тести паралельно на десятках або сотнях серверів, що значно скорочує час виконання.

Отже, автоматизація тестування є незамінним інструментом для забезпечення якості сучасного програмного забезпечення. У рамках кваліфікаційної роботи її використання дозволяє:

- Підтвердити знання теоретичних аспектів тестування.
- Продемонструвати практичні навички роботи з інструментами автоматизації.
- Підвищити загальну цінність роботи завдяки її практичному значенню для галузі.

Інтеграція автоматизації в кваліфікаційній роботі підкреслює інноваційний підхід дослідника, його готовність до вирішення реальних задач та здатність працювати з сучасними технологіями.

Змішане тестування поєднує найкращі практики ручного та автоматизованого підходів, дозволяючи забезпечити високу якість програмного забезпечення. Даний метод передбачає адаптивне використання кожного типу тестування залежно від специфіки завдання, що вирішується. У контексті кваліфікаційної роботи, змішане тестування стає оптимальним варіантом для комплексного аналізу функціональності, продуктивності та зручності використання програмного забезпечення.

Переваги змішаного тестування дозволяють об’єднати переваги ручного та автоматизованого тестування, створивши гнучку та ефективну методику. Основними перевагами такого підходу є:

- Комбінація швидкості та точності.
- Оптимізація ресурсів.
- Широке покриття тестування.

Ручне тестування у змішаній методиці відіграє важливу роль у перевірці нестандартних сценаріїв та оцінці досвіду користувача. Автоматизовані тести у змішаній методиці виконують завдання, які потребують швидкості та повторюваності. Ефективне поєднання ручного та автоматизованого тестування у змішаній методиці вимагає ретельного планування. Основними етапами інтеграції є:

- Визначення області відповідальності.
- Підготовка тестових сценаріїв.
- Моніторинг ефективності.

Змішане тестування є ефективним підходом у проектах різного масштабу. Наприклад, у розробці веб-додатків даний метод дозволяє виконувати швидке автоматизоване тестування базової функціональності, проводити ручне тестування для перевірки інтерактивних елементів та адаптивності дизайну, застосовувати автоматизовані інструменти для перевірки продуктивності та безпеки.

Такий підхід забезпечує високий рівень якості продукту, знижуючи ризики випуску дефектів у виробниче середовище.

Метод змішаного тестування є універсальним і гнучким підходом, що поєднує переваги двох основних методів тестування. У контексті кваліфікаційної роботи це дозволяє досягнути високої якості дослідження, забезпечуючи комплексний підхід до перевірки програмного забезпечення.

Попри численні переваги, змішаний підхід має свої труднощі, які варто враховувати:

- Високі вимоги до кваліфікації команди.
- Потреба в додаткових ресурсах.
- Ускладнення організації процесів.

Однак правильне розуміння цих викликів та їх врахування дозволяє мінімізувати ризики і забезпечити успішне впровадження змішаного тестування.

Реалізація змішаного тестування сприяє значному підвищенню якості програмного забезпечення. Завдяки цьому підходу можна зменшити кількість помилок, що потрапляють у виробниче середовище, забезпечити відповідність продукту вимогам замовника, покращити досвід користувачів завдяки оцінці зручності інтерфейсу.

Крім того, змішане тестування дозволяє зменшити час виходу продукту на ринок завдяки швидшому виявленню та виправленню помилок. У великих проектах, де обсяги коду та кількість сценаріїв тестування надзвичайно великі, змішане тестування стає незамінним. Воно дозволяє оптимізувати витрати., виконуючи автоматизовані тести для рутинних задач і залишати ручне тестування для унікальних сценаріїв, забезпечувати гнучкість, адаптувавши процеси до змін у вимогах проекту, розподіляти задачі між членами команди залежно від їхніх навичок та кваліфікації.

Метод змішаного тестування поєднує в собі переваги обох підходів, що робить його універсальним інструментом для забезпечення якості програмного забезпечення. У контексті кваліфікаційної роботи даний метод дозволяє

виконати комплексний аналіз, продемонструвати практичні навички та забезпечити цінність дослідження для галузі.

Для порівняння зазначених методів тестування будуть розглянуті ключові аспекти кожного методу, такі як точність, швидкість, витрати ресурсів та зручність застосування, а також можливість адаптації до різних типів веб-додатків. Особлива увага приділятиметься ситуаціям, коли кожен з методів може бути оптимальним, а також випадкам, де комбіновані підходи дають найбільшу ефективність.

Для аналізу було визначено основні критерії порівняння методів тестування, такі як:

- швидкість виконання,
- вартість,
- виявлення дефектів,
- обсяг покриття,
- масштабованість,
- вимоги до кваліфікації тестувальників.

На основі цих критеріїв створено узагальнену порівняльну таблицю:

Таблиця 2.1

Метод тестування	Швидкість виконання	Вартість	Виявлення дефектів	Обсяг покриття	Масштабованість
Ручне	Низька	Низька	Висока	Обмежений	Обмежена
Автоматизоване	Висока	Висока	Висока	Широкий	Висока
Змішане	Середня	Середня	Висока	Середній	Середня

Тепер проведемо аналіз даної таблиці по атрибутам таблиці:

- Швидкість виконання – автоматизовані тести виконуються значно швидше за ручні завдяки використанню інструментів.
- Вартість – ручне тестування дешевше в короткостроковій перспективі, але автоматизоване є більш економічним на довгих проєктах.

- Виявлення дефектів – дана можливість реалізована на високому рівні в кожному з методів тестувань
- Обсяг покриття – автоматизоване тестування дозволяє перевірити велику кількість сценаріїв.
- Масштабованість – ввтоматизоване тестування легко масштабується завдяки можливості паралельного запуску тестів.

2.2. Інструментальні засоби вирішення завдання кваліфікаційної роботи

Інструменти тестування програмного забезпечення відіграють ключову роль у забезпеченні якості продукту. Їх правильний вибір визначає ефективність, швидкість і точність процесу перевірки. У контексті кваліфікаційної роботи важливо враховувати специфіку проекту, функціональні вимоги, масштаб, типи тестування та доступні ресурси. Їх можна розділити на кілька категорій залежно від їхнього призначення:

- Інструменти для ручного тестування.
- Інструменти для автоматизації тестування.
- Інструменти для тестування продуктивності.
- Інструменти для управління тестуванням.
- Інструменти для інтеграції та безперервного тестування.

Вибір інструментів залежить від кількох факторів, серед яких:

- Тип проекту;
- Масштаб проекту;
- Рівень автоматизації;
- Бюджет.

Огляд популярних інструментів для проведення тестування:

- Selenium – це один із найпопулярніших інструментів для автоматизації тестування веб-додатків(Рисунок 1.1Рисунок 1.2Рисунок 1.3Рисунок 2.1Рисунок 1.4Рисунок 1.5Рисунок 1.6Рисунок 1.7Рисунок 2.2). Основні перевагами у Selenium є підтримка різних мов програмування. Selenium

сумісний із Python, Java, C#, що дозволяє інтегрувати його у більшість проектів, мультиплатформність, бо Інструмент працює на різних операційних системах та браузерах, його можна інтегрувати з іншими інструментами.

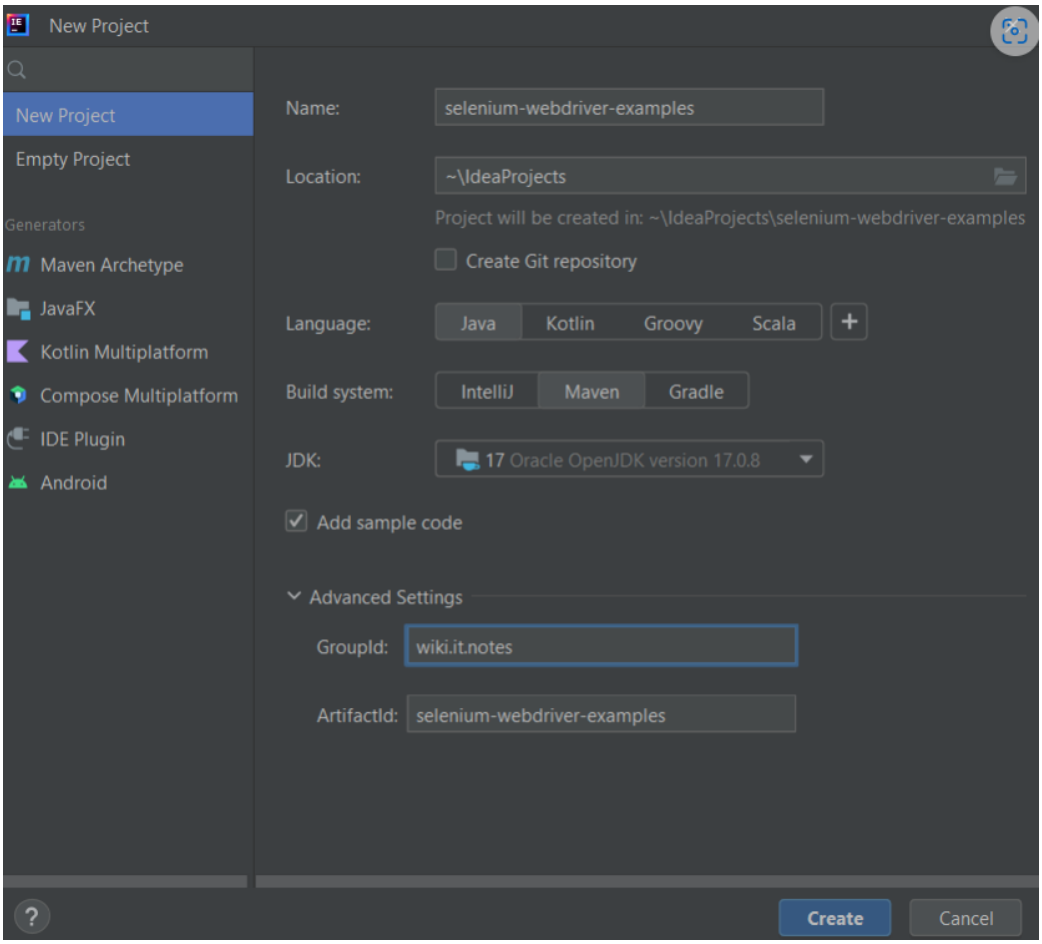


Рисунок 2.2– Застосування Selenium в проекті

JIRA – це система для управління завданнями та відстеження дефектів. Вона широко використовується у процесах тестування завдяки таким можливостям. Дозволяє інтеграціяю з іншими інструментами, користувачі можуть створювати кастомні робочі процеси та звіти,а також всі дефекти, знайдені під час тестування, систематизуються та зберігаються для подальшого аналізу.

У рамках кваліфікаційної роботи використовується кілька інструментів для забезпечення всебічного тестування. Наприклад:

- Для автоматизації – Selenium допоможе створити автоматизовані тести для основної функціональності.
- Для управління процесом – JIRA дозволить відстежувати помилки та координувати роботу команди.

Тобто, інтеграція кількох інструментів у процес тестування надає значні переваги, які позитивно впливають на якість і швидкість роботи команди. По-перше, підвищується ефективність процесу. Автоматизація рутинних завдань, таких як запуск тестів, створення звітів чи аналіз логів, значно зменшує час, необхідний на виконання повторюваних дій. Крім того, централізація даних дозволяє швидше знаходити потрібну інформацію, що оптимізує роботу тестувальників. По-друге, використання спеціалізованих інструментів сприяє поліпшенню якості тестування. Завдяки їхнім можливостям можна виявляти складні дефекти, які важко помітити за допомогою ручного підходу. Наприклад, інструменти для тестування навантаження або аналізу безпеки дають змогу виявляти проблеми, які могли б залишитися непоміченими.

Також, інтеграція інструментів забезпечує гнучкість у масштабуванні тестування. У міру зростання потреб проєкту ці інструменти легко адаптуються до нових вимог, додають підтримку для більших обсягів даних або більш складних сценаріїв тестування. Це дозволяє ефективно підтримувати якість продукту навіть на пізніх етапах його розробки.

Сучасні інструменти тестування постійно розвиваються, додаючи нові функції для адаптації до викликів галузі. У майбутньому очікується:

- Ширше застосування штучного інтелекту для автоматизації складних сценаріїв тестування.
- Розширення можливостей інтеграції з DevOps-процесами.
- Збільшення кількості хмарних рішень для тестування.

Аналіз інструментів тестування дозволяє вибрати оптимальні рішення для забезпечення якості програмного забезпечення. У контексті кваліфікаційної

роботи правильний вибір інструментів демонструє здатність дослідника ефективно використовувати сучасні технології для вирішення складних завдань.

Математичне моделювання є потужним інструментом для оцінки ефективності різних підходів до тестування. Воно дозволяє проводити об'єктивний аналіз витрат, часу та результатів, що забезпечує оптимізацію процесів розробки програмного забезпечення.

Математичне моделювання є важливим інструментом для вирішення складних завдань у галузі тестування програмного забезпечення. Воно дозволяє формалізувати процеси, оцінювати ефективність різних підходів до тестування, прогнозувати результати та оптимізувати витрати часу, ресурсів і зусиль. У рамках кваліфікаційної роботи застосування математичних моделей відкриває можливість для глибокого аналізу ручного, автоматизованого та змішаного підходів до тестування.

Математична модель – це абстрактне представлення реальної системи або процесу у вигляді математичних залежностей, рівнянь та змінних. У контексті тестування програмного забезпечення моделі дозволяють:

- Оцінити кількість тестів, необхідних для забезпечення заданого рівня покриття коду.
- Визначити оптимальний баланс між ручним і автоматизованим тестуванням.
- Прогнозувати час виконання тестів залежно від обсягу роботи та доступних ресурсів.

Ручне тестування часто вимагає багато часу, тому математичні моделі допомагають оптимізувати витрати. Наприклад, можна використати модель розрахунку часу виконання тестів залежно від кількості тестувальників та складності тестових сценаріїв:

$$T_{\text{метод тестування}} = \frac{T*C}{K} \tag{2.1}$$

де:

T – загальний час тестування по методу,

N – кількість тестів,

C – середня складність,

K – швидкість виконання тестів одним тестувальником.

Дана формула допомагає отримати результат виконання 1 тест-кейс за хвилину, в залежності від методу тестування.

Ще одним важливим аспектом математичного моделювання є прогнозування кількості знайдених дефектів. Для цього використовуються моделі, засновані на теорії ймовірностей:

$$P(D) = 1 - (1 - p)^N \quad (2.2)$$

де:

P(D) – ймовірність виявлення дефекту,

p – ймовірність виявлення дефекту одним тестом,

N – кількість тестів.

Дана формула дозволяє прогнозувати кількість знайдених дефектів у тест-кейсах, в залежності від методу тестування.

У рамках кваліфікаційної роботи математичні моделі можуть бути застосовані для:

- Оцінки ефективності запропонованих підходів до тестування.
- Визначення оптимальних параметрів тестування, таких як кількість тестів чи рівень автоматизації.
- Прогнозування результатів і витрат на основі реальних даних проекту.
- Моделі дозволяють уникнути суб’єктивних оцінок, базуючи рішення на фактичних даних.
- Можна адаптувати до потреб конкретного проекту.

- Дозволяють прогнозувати результати ще до початку тестування.

Застосування математичних моделей у тестуванні програмного забезпечення дозволяє зробити процес більш структурованим, ефективним і прогнозованим. Такі моделі дають змогу детально аналізувати залежності між ключовими параметрами, як-от час виконання тестів, витрати ресурсів і якість отриманих результатів, що дозволяє створювати оптимізовані підходи до тестування, підвищуючи його точність та надійність. У рамках кваліфікаційної роботи використання математичних моделей демонструє здатність дослідника до комплексного аналізу даних і вирішення складних завдань, що підкреслює важливість дослідження як інструменту для вдосконалення процесу тестування й розвитку галузі в цілому.

Також не слід забувати за сучасні тенденції використання ШІ у різних системах як розробки так і тестування ПЗ. Використання технологій штучного інтелекту у тестуванні програмного забезпечення є важливим кроком до автоматизації та підвищення ефективності процесів перевірки якості програм.

Тестування є невід’ємною частиною розробки програмного забезпечення, оскільки дозволяє виявити помилки і дефекти, що можуть призвести до серйозних проблем у роботі кінцевого продукту. Традиційно тестування програмного забезпечення проводиться вручну, що потребує значних людських та часових ресурсів. Однак, з розвитком технологій штучного інтелекту, з'явилася можливість автоматизувати багато етапів тестування , що дозволяє значно скоротити час виконання тестів, підвищити точність результатів та знизити витрати на тестування.

Штучний інтелект, в свою чергу, включає в себе кілька підходів, зокрема машинне навчання та глибинне навчання, які дозволяють створювати системи, здатні до самонавчання та автоматичного удосконалення. Завдяки таким технологіям, ШІ здатен виконувати складні задачі, аналізувати великі обсяги даних і приймати рішення на основі отриманих результатів. У тестуванні програмного забезпечення ШІ може бути застосований для автоматизації процесів генерації тестових сценаріїв, виявлення помилок, оптимізації тестових

набірів, а також для прогнозування потенційних проблем до початку виконання тесту, що забезпечує значне підвищення ефективності та результативності тестування ПЗ.

Окрім цього, використання ШІ дозволяє знижувати ймовірність людських помилок, які можуть виникати під час виконання тестів вручну. Враховуючи, що сучасні програмні комплекси стають все складнішими, автоматизація тестування за допомогою ШІ є важливою умовою для забезпечення їхньої якості. Застосування ШІ в тестуванні дозволяє не лише автоматизувати існуючі процеси, але й створювати нові стратегії тестування, які раніше були б недоступні через обмеження традиційних методів. Це забезпечує більш ефективне виявлення дефектів, зменшення кількості пропущених помилок і скорочення витрат на тестування. Одним із основних аспектів використання ШІ в тестуванні є його здатність обробляти великі обсяги даних, що є важливим при тестуванні складних програмних систем. ШІ здатен швидко обробляти інформацію, що дозволяє значно зменшити час на виконання тестів та підвищити їх точність.

Крім того, технології ШІ дозволяють впроваджувати інтелектуальні стратегії тестування, які адаптуються до змін в програмному забезпеченні, що робить тестування більш гнучким і ефективним. В цьому контексті важливо зазначити, що технології ШІ відкривають нові можливості для автоматизації не лише тестування, а й інших етапів розробки програмного забезпечення, таких як виявлення помилок на етапі проектування або навіть під час виконання коду.

Штучний інтелект також дозволяє створювати більш інтелектуальні інтерфейси тестування, які не лише автоматично виконують тести, але й аналізують їх результати, надаючи рекомендації щодо подальших кроків або оптимізації тестових сценаріїв, що дозволяє зменшити необхідність у ручному втручанні і зробити процес тестування більш ефективним.

Застосування ШІ у тестуванні програмного забезпечення охоплює кілька ключових напрямків, кожен з яких має своє значення і дозволяє досягти кращих результатів у порівнянні з традиційними методами. Одним з основних напрямків

є автоматичне генерування тестових даних. Традиційно тестові дані для програми створюються вручну, що займає значний час і ресурси. ШІ дозволяє автоматично генерувати дані, які можуть бути використані для тестування різних сценаріїв роботи програмного забезпечення, що особливо корисно в тих випадках, коли потрібно перевірити програму на великих обсягах даних або з різними вхідними параметрами.

Автоматичне генерування тестових даних за допомогою ШІ дозволяє не лише зекономити час, але й покращити якість тестів. ШІ здатен генерувати тестові дані, які є більш різноманітними і наближеними до реальних умов експлуатації програмного забезпечення. Наприклад, для тестування веб-додатків ШІ може створювати великі набори даних, що включають різні варіанти введення користувачем інформації, що дозволяє тестувати додаток на різних типах вхідних даних і перевіряти його стійкість до різноманітних ситуацій.

Іншим важливим напрямком є тестування продуктивності та навантаження, що включає в себе моделювання реальних умов навантаження на програму, що дозволяє оцінити, як програма буде працювати при високих навантаженнях або в умовах обмежених ресурсів. За допомогою ШІ можна адаптувати тестові сценарії залежно від умов, створюючи найскладніші тести, які імітують реальні умови використання програмного забезпечення. Наприклад, для тестування веб-додатка з високим трафіком ШІ може створювати сценарії, які будуть генерувати навантаження, що наближаються до реальних умов, перевіряючи, як додаток справляється з великою кількістю одночасних запитів.

Виявлення помилок і дефектів є ще одним важливим напрямком, де ШІ активно використовується. Алгоритми машинного навчання здатні виявляти дефекти, які важко помітити при традиційному тестуванні. За допомогою аналізу логу роботи програми, вихідних даних та інших параметрів, ШІ може швидко і точно знайти помилки. Враховуючи, що сучасне програмне забезпечення часто має велику кількість взаємозалежних компонентів, використання ШІ для виявлення помилок дозволяє виявляти проблеми, які можуть бути пропущені іншими методами тестування. Наприклад, ШІ здатен виявити аномалії в роботі

програми, які можуть призвести до неочікуваних результатів або збою системи, що не було б помічено вручну.

Завдяки ШІ можна автоматизувати тестування таких додатків на різних пристроях, операційних системах та версіях. Це дозволяє значно зменшити час, необхідний для проведення тестів, і підвищити точність результатів. ШІ може адаптувати тестові сценарії до різних версій і платформ, що робить процес тестування більш ефективним. Наприклад, для тестування мобільних додатків можна автоматично генерувати сценарії, що імітують різні умови використання на різних пристроях, таких як смартфони, планшети, з різними операційними системами та рівнями оновлень.

Інтелектуальні інтерфейси тестування є важливим напрямком, де ШІ може бути використаний для створення спеціальних інтерфейсів, які не лише автоматично виконують тести, але й аналізують їх результати, надаючи рекомендації щодо подальших кроків або оптимізації тестових сценаріїв. Це дозволяє зменшити необхідність у ручному втручанні і зробити процес тестування більш ефективним. Наприклад, інтерфейс на основі ШІ може автоматично вибирати найбільш релевантні тести для певної ситуації, ураховуючи зміни в програмі та попередні результати тестування.

Алгоритми машинного навчання є основою багатьох рішень, що застосовуються в тестуванні програмного забезпечення. Вони дозволяють автоматизувати багато процесів, таких як виявлення дефектів, оптимізація тестових сценаріїв та аналіз результатів тестування. Одним з основних алгоритмів, що застосовуються в тестуванні, є класифікація. Вона використовується для категоризації дефектів або тестів. Наприклад, алгоритми класифікації можуть автоматично визначати тип дефекту, що дає можливість швидко ідентифікувати і виправити помилки. Завдяки класифікації можна організувати багатофункціональні процеси тестування, в яких кожен дефект буде віднесений до відповідної категорії, що дозволить швидше знаходити і виправляти помилки.

Іншим популярним методом є кластеризація. Вона використовується для групування схожих об'єктів чи дефектів, що особливо корисно, коли потрібно знайти пат

Регресія — це ще один важливий метод, який використовується для прогнозування результатів тестування. За допомогою регресії можна передбачити, як зміни в коді програми вплинуть на її роботу та які дефекти можуть виникнути після внесення змін. Це дозволяє проводити більш точні тести і знижувати ймовірність виникнення помилок.

Технології машинного навчання можуть бути використані для передбачення ймовірних проблем або дефектів у програмному забезпеченні до того, як вони будуть виявлені в результатах тестів. Наприклад, на основі аналізу історичних даних про дефекти та тестування, ШІ може створювати моделі, які дозволяють прогнозувати, в яких частинах програми ймовірніше за все виникнуть помилки при внесенні змін.

Один з популярних підходів, що використовується для прогнозування результатів тестування, це класифікація ризиків**,** яка дозволяє оцінити, які частини програмного продукту мають найбільшу ймовірність дефектів або збоїв. Алгоритми машинного навчання аналізують історію помилок, проблеми з продуктивністю та інші фактори, щоб передбачити, де в майбутньому можуть виникнути проблеми. Це дозволяє командам тестувальників зосередитися на найбільш критичних ділянках коду, що потребують ретельнішого тестування.

ШІ також може бути використаний для виявлення шаблонів або аномалій у процесі тестування. Це дозволяє виявляти неочевидні проблеми, які можуть бути незрозумілими або складними для ручного аналізу. Наприклад, якщо тестування показує, що певні сценарії викликають непередбачувану поведінку програми, алгоритми ШІ можуть допомогти виявити шаблони, які дозволяють зрозуміти, в чому саме полягає причина помилки, що дозволяє швидко реагувати на потенційні проблеми, покращуючи ефективність тестування.

Крім того, в контексті прогнозування результатів тестування використання ШІ допомагає зменшити кількість пропущених помилок і збоїв, особливо в

складних програмних системах, де взаємозалежність компонентів може призвести до несподіваних наслідків. Моделі, побудовані на основі ШІ, здатні автоматично виявляти взаємозв'язки між різними частинами програми, що дозволяє вчасно виявити потенційні проблеми навіть до початку фактичного тестування.

ШІ активно використовується для автоматизації тестування інтерфейсу користувача та досвіду користувача**.**

Тестування UI є складним процесом, який включає перевірку не лише функціональності елементів інтерфейсу, але й їхньої зручності, доступності та відповідності вимогам користувачів. Тестування UX включає в себе більш глибокий аналіз взаємодії користувача з додатком, зокрема перевірку того, наскільки інтерфейс є інтуїтивно зрозумілим і зручним.

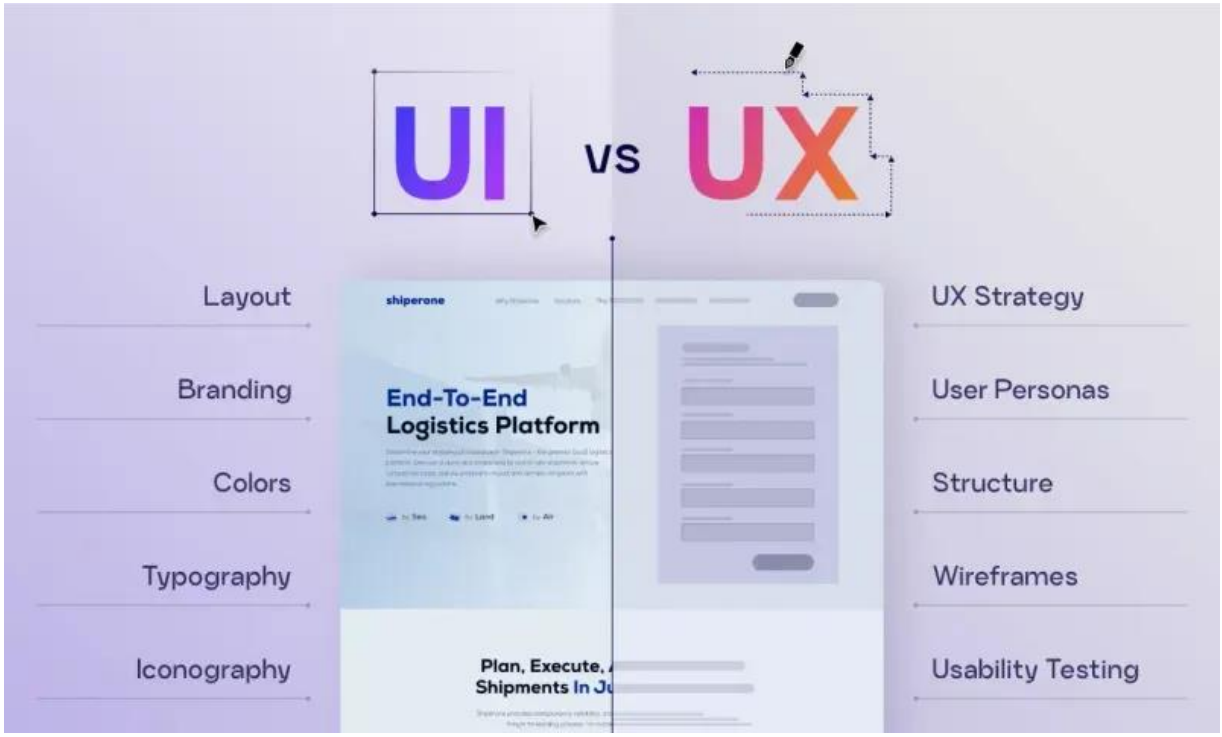


Рисунок 2.3 – UI та UX модулі

ШІ може значно полегшити цей процес, автоматизуючи перевірку різних аспектів UI та UX. Наприклад, алгоритми ШІ можуть перевіряти, чи коректно відображаються всі елементи інтерфейсу на різних пристроях і платформах, і чи

виконуються всі функції за належним чином. Вони також можуть перевіряти, чи відповідає зовнішній вигляд інтерфейсу вимогам доступності, зокрема для користувачів з обмеженими можливостями.

Використання ШІ в тестуванні UX дозволяє надавати більш глибокі інсайти щодо того, як користувачі взаємодіють з додатком. Наприклад, ШІ може аналізувати поведінку користувачів під час тестування, визначати зони, де виникають труднощі або де користувачі проводять більше часу, а також надавати рекомендації щодо покращення взаємодії. Даний підхід дозволяє тестувати не лише функціональність програми, але й її зручність для кінцевих користувачів, що є важливим аспектом розробки сучасних веб-додатків і мобільних програм.

Окремо варто відзначити застосування ШІ в тестуванні безпеки програмного забезпечення**.** З настанням ери цифрових технологій, коли програмні продукти стають все більш складними та взаємозалежними, питання безпеки набуває особливої важливості. ШІ може відігравати важливу роль у виявленні вразливостей у програмному забезпеченні, таких як SQL-ін’єкції, XSS-атаки та інші типи загроз.

ШІ здатен автоматично сканувати код на наявність потенційних вразливостей, аналізуючи патерни, що можуть вказувати на можливі загрози. Наприклад, алгоритми ШІ можуть виявляти незахищені точки входу в систему або перевіряти, чи правильно обробляються чутливі дані. Завдяки використанню машинного навчання, ШІ може адаптуватися до нових загроз, що з’являються в кіберпросторі, і своєчасно оновлювати стратегії тестування.

Крім того, ШІ може бути використаний для симуляції атак**,** що дозволяє перевірити, як програма реагує на різні типи кіберзагроз, і виявити слабкі місця, які можуть бути використані зловмисниками. Це дозволяє здійснювати тестування безпеки на більш високому рівні, ніж традиційні методи, і забезпечувати надійний захист від різноманітних атак.

ШІ може бути застосований на різних етапах розробки програмного забезпечення, від початкової стадії до моменту випуску продукту.

На етапі проектування ШІ може допомогти в аналізі вимог до програмного забезпечення, створюючи моделі, які враховують різні сценарії використання продукту. Це дозволяє визначити, які функціональні можливості потребують найбільш ретельного тестування, а також створити сценарії для тестування на основі аналізу вимог.

На етапі розробки ШІ може бути використаний для автоматичного тестування коду, виявлення помилок на ранніх стадіях та перевірки виконання тестів у реальному часі. Це дозволяє знизити ймовірність помилок, які можуть виникнути в кінці розробки, і забезпечити більш стабільну та надійну роботу програмного продукту.

Рисунок 2.4 – Принципи роботи сучасного ШІ

На етапі експлуатації ШІ може бути використаний для моніторингу роботи програми в реальних умовах, виявлення потенційних проблем і автоматичного тестування змін, що вносяться до коду, що дозволяє вчасно виявляти дефекти, які можуть з'явитися після випуску продукту, і забезпечити безперервну перевірку якості програмного забезпечення.

Використання штучного інтелекту в тестуванні програмного забезпечення є потужним інструментом для автоматизації та покращення процесу забезпечення якості продуктів. ШІ дозволяє знизити витрати на тестування, підвищити ефективність і точність результатів, а також забезпечити більш глибокий аналіз дефектів та аномалій.

Також ШІ дозволяє тестувати програмне забезпечення в нових, раніше недоступних напрямках, таких як тестування безпеки, прогнозування дефектів та автоматизація тестування UI/UX. Він вже активно використовується в ряді компаній для автоматизації тестування, і з кожним роком ця технологія набирає популярності. Враховуючи швидкий розвиток технологій ШІ, можна очікувати, що в майбутньому його роль у тестуванні програмного забезпечення лише зростатиме.

2.3. Висновки до другого розділу

Другий розділ присвячений опису основних застосованих методів тестування для вирішення завдання кваліфікаційної роботи, а саме: ручного, автоматизованого та комбінованого. Було детально розглянуто їхні переваги та недоліки, які впливають як на процес тестування, так і на якість результатів. Проведено систематизацію цих характеристик і визначено сфери оптимального застосування кожного методу. Особлива увага була приділена таким аспектам, як швидкість виконання, особливості тестування та якість отриманих результатів.

Розроблені математичні моделі забезпечили можливість формалізованого порівняння обох методів на основі аналітичних підходів, що дозволило виявити залежності між витратами часу, ресурсами та обсягом виконуваних тестів, що дало змогу здійснити кількісний аналіз ефективності методів.

Зокрема, виявлено, що автоматизоване тестування є ефективнішим при великому обсязі рутинних перевірок, тоді як ручне тестування виявляється корисним у ситуаціях, що вимагають творчого підходу чи врахування

нестандартних сценаріїв. Додатково розглянуто перспективи інтеграції штучного інтелекту в процес тестування. Теоретичні дослідження показали, що ШІ має значний потенціал у вдосконаленні кожного з методів. У ручному тестуванні він може сприяти автоматизації генерації тест-кейсів, аналізу поведінки користувачів і виявленню аномалій у даних. В автоматизованому тестуванні ШІ дозволяє покращити механізми аналізу, створювати оптимізовані тестові сценарії та ефективніше виявляти приховані дефекти. Тому застосування ШІ є важливою частиною під час формулювання методики тестування.

Також було проведено аналіз ролі комбінованого тестування, яке поєднує сильні сторони обох підходів. Зокрема, дослідження підтвердили, що правильне використання такого підходу сприяє досягненню оптимального співвідношення між витратами, якістю та швидкістю виконання тестування.

Теоретичне дослідження підтвердило, що вибір методу тестування має базуватися на специфіці проєкту, масштабах тестування, складності системи та потребах команди розробників. Комплексний підхід із використанням аналітичних моделей та сучасних технологій, включаючи штучний інтелект, може забезпечити високу якість тестування та зменшення витрат на його виконання.

РОЗДІЛ 3. РОЗРОБКА МЕТОДИКИ ТЕСТУВАННЯ ВЕБ-ДОДАТКІВ

3.1. Критерії створення методики тестування веб-додатку

Щоб досягти успішного функціонування програмного забезпечення, необхідно, щоб усі сценарії тестування були виконані без помилок. Для цього потрібен час, ресурси та спеціальні знання. Враховуючи ці фактори, для оптимізації процесу перевірки конкретного програмного забезпечення необхідно розробити власну методику тестування, що базується на практичному досвіді. Одним із ключових елементів такої методики є розуміння того, як застосовувати різні типи тестування, зокрема ручне, автоматизоване та змішане тестування, для досягнення найбільш ефективних результатів.

Важливим аспектом є також вибір інструментів тестування, які відповідають специфікаціям програмного забезпечення і забезпечують максимальну покриття усіх можливих сценаріїв, що можуть виникнути у процесі експлуатації продукту. Наприклад, для тестування веб-додатків необхідно враховувати різноманітні браузери, операційні системи та пристрої, на яких буде працювати додаток, що вимагає використання як автоматизованих тестів для швидкості, так і ручних тестів для глибшої перевірки конкретних функцій або взаємодії з користувачем.

Крім того, важливою складовою є вибір стратегії тестування, що дозволяє ефективно використовувати ресурси, що може включати в себе пріоритизацію тестів залежно від критичності функцій програми, а також застосування методів, які дозволяють скоротити час на виконання тестування без втрати якості, наприклад, використання тестів, які автоматично генеруються на основі змін у коді.

За допомогою таких підходів можна створити методику, що дозволяє не лише знижувати витрати часу і ресурсів, але й підвищувати якість програмного забезпечення. Важливим є також інтеграція тестування у процес розробки, що

дозволяє виявляти дефекти на найраніших етапах і запобігати їх поширенню в подальших версіях продукту.

Першим кроком у створенні такої методики є визначення ключових критеріїв, що дозволять зробити її універсальною та ефективною. Визначення цих критеріїв є важливим етапом для забезпечення правильності та всебічності підходу до тестування. Серед таких критеріїв слід включити такі моменти, як:

- Тип проєкту – невеликі, короткотривалі проєкти краще тестувати вручну, щоб уникнути витрат на розробку автоматизованих тестів. Для довготривалих проєктів з великою кількістю повторюваних сценаріїв автоматизоване тестування є ефективнішим.
- Бюджет, якщо він обмежений, краще зосередитися на ручному тестуванні. Для середніх бюджетів доцільно використовувати змішаний підхід, розподіляючи ресурси залежно від критичності завдань.
- Час на тестування – за умов жорстких дедлайнів автоматизація дозволяє значно скоротити час виконання тестів. Ручне тестування підходить для гнучких часових рамок.
- Складність проєкту – у складних проєктах із широким функціоналом доцільно використовувати автоматизовані інструменти для тестування окремих компонентів.
- Критичність до якості, де помилка може спричинити значні втрати, рекомендується застосовувати змішаний підхід, поєднуючи точність ручного тестування з масштабованістю автоматизації.

Методика враховує ключові особливості проєктів і дозволяє оптимізувати процеси, адаптуючи їх до реальних умов.

3.2. Дослідження методів тестування веб-додатків

Для дослідження ефективності різних методів тестування було обрано веб-додаток Приват24. Завдання тестування полягає у виявленні дефектів в

основному фукціоналі веб-додатку, такі як: забезпечення коректної роботи форми входу та перевірці обробки помилкових сценаріїв.

Для тестування було створено 10 тест-кейсів та зображено їх у вигляді таблички, яка дозволяє перевірити основний функціонал:

Таблиця 3.1

№	Назва тесту	Опис тесту	Очікуваний результат
1	Вхід в систему	Перевірка входу до акаунту з правильними даними	Користувач успішно входить в систему
2	Неправильний пароль	Перевірка спроби входу з неправильним паролем	Виведення повідомлення про помилку
3	Відновлення пароля	Перевірка відновлення пароля за допомогою електронної пошти	Користувач отримує лист з інструкціями
4	Переведення коштів	Переведення грошей з одного рахунку на інший	Успішне виконання транзакції
5	Перевірка балансу акаунту	Перевірка відображення балансу акаунту	Баланс відображається коректно
6	Перегляд історії транзакцій	Перевірка історії всіх транзакцій акаунту	Відображення всіх транзакцій за вибраний період
7	Вихід із системи	Перевірка виходу з системи	Користувач успішно виходить з системи
8	Активація картки	Перевірка активації картки через додаток	Успішна активація картки
9	Блокування картки	Перевірка блокування картки через додаток	Картка успішно заблокована

10	Зміна особистих даних	Перевірка зміни особистих даних користувача	Дані успішно змінюються

Для початку, щоб перевірити всі можливі варіанти, потрібно застосувати методи ручного тестування ПЗ. Так, як це традиційний підхід, який широко використовується в розробці програмного забезпечення.

Ручне тестування вважається надійним методом, оскільки дає можливість детально перевірити функціональність програми, проте цей метод має і свої недоліки, зокрема, він займає багато часу і може бути схильним до людських помилок. Однак, незважаючи на ці недоліки, ручне тестування залишається важливим етапом, оскільки дозволяє отримати точні результати в умовах невеликої кількості тестів чи складних сценаріїв, де автоматизація ще не є доцільною.

Для ручного тестування кожен тест-кейс виконувався вручну, і час виконання кожного з них був зафіксований, що дозволяє оцінити ефективність тестування, а також виявити потенційні проблеми, які можуть виникнути при виконанні тестів вручну. Ось результати тестування, які демонструють, як кожен тест-кейс проходить через процес перевірки, а також інтерпретацію часу та ефективності цього методу.

Відобразимо результати проходження 10 тест-кейсів за допомогою методу ручного тестування:

Таблиця 3.2

№	Назва тесту	Час виконання (хв)	Кількість дефектів	Коментарі
1	Вхід в систему	3	0	Успішний вхід до акаунту
2	Неправильний пароль	2	0	Повідомлення про помилку
3	Відновлення пароля	4	0	Лист на пошту отримано
4	Переведення коштів	4	0	Транзакція виконана
5	Перевірка балансу акаунту	2	0	Баланс відображено правильно

6	Перегляд історії транзакцій	3	0	Історія транзакцій коректна
7	Вихід із системи	3	0	Вихід виконано успішно
8	Активація картки	5	0	Картка активована
9	Блокування картки	4	0	Картка заблокована
10	Зміна особистих даних	6	0	Дані успішно змінено

Загалом, ручне тестування показало непогану ефективність, час проходження 10 тест-кейсів дорівнює 44 хв у виконанні основних сценаріїв, при цьому більшість тестів пройшли без виявлення дефектів. Всі ключові функції, такі як вхід до системи, переведення коштів, виведення коштів, перевірка балансу акаунту, блокування картки, зміна особистих даних та двофакторна аутентифікація, продемонстрували стабільну роботу без помилок.

В цілому, хоча ручне тестування дозволяє виявити важливі дефекти та перевірити основні функціональні можливості, цей процес займає чимало часу та ресурсів. Для великих проектів або частих оновлень програмного забезпечення це може бути не найбільш ефективним методом, оскільки кожен тест потребує ручного виконання і фіксації часу. Тому для забезпечення більшої ефективності варто розглянути можливість автоматизації тестування, що дозволить значно зменшити витрати часу та ресурсів.

Продовженням роботи стало проведення перевірки за допомогою автоматизованого методу тестування. У рамках цієї частини дослідження були розроблені та реалізовані автоматизовані тести для кожного з 10 тест-кейсів, які раніше були визначені для ручного тестування. Для цього використовувалася мова програмування Java, а основним інструментом автоматизації став Selenium WebDriver, який забезпечує високий рівень інтеграції з веб-додатками.

Автоматизовані тести також враховували специфічні вимоги до продуктивності та стабільності системи, виконуючи багаторазові ітерації перевірок для виявлення можливих помилок у роботі додатку. Завдяки цьому вдалося значно скоротити час, необхідний для проведення тестування, та забезпечити більш детальну перевірку кожного окремого функціонального

модуля. Перелік тестів із кодом автоматизації написаний мовою програмування Java у програмному середовищі IntelliJ IDEA Community Edition 2024.3:

- Тест-кейс 1:

```
public void testLogin() {
    // Назва тесту: Вхід в систему
    driver.get("https://api.privat24.ua/");

    WebElement loginField = driver.findElement(By.id("login"));
    WebElement passwordField = driver.findElement(By.id("password"));
    WebElement loginButton = driver.findElement(By.id("submit"));

    loginField.sendKeys("correctUsername");
    passwordField.sendKeys("correctPassword");
    loginButton.click();

    WebElement accountPage = driver.findElement(By.id("accountPage"));
    assert accountPage.isDisplayed();
}
```

Рисунок 3.1 – Код автоматизація для входу у систему

Час виконання**:** 5 секунд

Результат**:** Тест пройшов успішно, користувач успішно увійшов у систему.

- Тест-кейс 2:

```
public void testInvalidPassword() {
    // Назва тесту: Неправильний пароль
    driver.get("https://api.privat24.ua/");

    WebElement loginField = driver.findElement(By.id("login"));
    WebElement passwordField = driver.findElement(By.id("password"));
    WebElement loginButton = driver.findElement(By.id("submit"));

    loginField.sendKeys("correctUsername");
    passwordField.sendKeys("wrongPassword");
    loginButton.click();

    WebElement errorMessage = driver.findElement(By.id("errorMessage"));
    assert errorMessage.isDisplayed();
}
```

Рисунок 3.2 – Код автоматизація для отримання нотифікації неправильного паролю

Час виконання**:** 4 секунди

Результат**:** Тест пройшов успішно, помилка виведена для неправильного пароля.

- Тест-кейс 3:

```
public void testPasswordRecovery() {
    // Назва тесту: Відновлення пароля
    driver.get("https://api.privat24.ua/");

    WebElement forgotPasswordLink = driver.findElement(By.linkText("Forgot Password"));
    forgotPasswordLink.click();

    WebElement emailField = driver.findElement(By.id("email"));
    WebElement recoverButton = driver.findElement(By.id("recover"));

    emailField.sendKeys("testemail@example.com");
    recoverButton.click();

    WebElement recoveryMessage = driver.findElement(By.id("recoveryMessage"));
    assert recoveryMessage.isDisplayed();
}
```

Рисунок 3.3 – Код автоматизація для відновлення паролю

Час виконання**:** 7 секунд

Результат: Відновлення пароля пройшло успішно, надіслано посилання для зміни пароля.

- Тест-кейс 4:

```
public void testTransferFunds() {
    // Назва тесту: Переведення коштів
    driver.get("https://api.privat24.ua/");

    WebElement transferTab = driver.findElement(By.id("transferTab"));
    transferTab.click();

    WebElement amountField = driver.findElement(By.id("amount"));
    WebElement sendButton = driver.findElement(By.id("send"));

    amountField.sendKeys("100");
    sendButton.click();

    WebElement successMessage = driver.findElement(By.id("successMessage"));
    assert successMessage.isDisplayed();
}
```

Рисунок 3.4 – Код автоматизація для переведення коштів

Час виконання**:** 8 секунд

Результат: Переведення коштів завершено успішно.

- Тест-кейс 5:

```
public void testCheckBalance() {
    // Назва тесту: Перевірка балансу акаунту
    driver.get("https://api.privat24.ua/");

    WebElement balanceTab = driver.findElement(By.id("balanceTab"));
    balanceTab.click();

    WebElement balance = driver.findElement(By.id("balance"));
    assert balance.isDisplayed();
}
```

Рисунок 3.5 – Код автоматизація для перевірки балансу рахунку

Час виконання: 3 секунди

Результат: Баланс успішно відображається.

- Тест-кейс 6:

```
public void testTransactionHistory() {
    // Назва тесту: Перегляд історії транзакцій
    driver.get("https://api.privat24.ua/");

    WebElement historyTab = driver.findElement(By.id("historyTab"));
    historyTab.click();

    WebElement historyTable = driver.findElement(By.id("historyTable"));
    assert historyTable.isDisplayed();
}
```

Рисунок 3.6 – Код автоматизація для перегляду історії транзакцій

Час виконання: 5 секунд

Результат: Історія транзакцій відображається коректно.

- Тест-кейс 7:

```
public void testLogout() {
    // Назва тесту: Вихід із системи
    driver.get("https://api.privat24.ua/");

    WebElement logoutButton = driver.findElement(By.id("logoutButton"));
    logoutButton.click();

    WebElement loginPage = driver.findElement(By.id("loginPage"));
    assert loginPage.isDisplayed();
}
```

Рисунок 3.7 – Код автоматизація для виходу із аккаунту Приват24

Час виконання: 3 секунди

Результат: Користувач успішно вийшов із системи.

- Тест-кейс 8:

```
public void testActivateCard() {
    // Назва тесту: Активація картки
    driver.get("https://api.privat24.ua/");

    WebElement cardSection = driver.findElement(By.id("cardSection"));
    cardSection.click();

    WebElement activateButton = driver.findElement(By.id("activateCard"));
    activateButton.click();

    WebElement successMessage = driver.findElement(By.id("successMessage"));
    assert successMessage.isDisplayed();
}
```

Рисунок 3.8 – Код автоматизація для активації картки після блокування

Час виконання: 6 секунд

Результат: Картка активована успішно.

- Тест-кейс 9:

```
public void testBlockCard() {
    // Назва тесту: Блокування картки
    driver.get("https://api.privat24.ua/");

    WebElement cardSection = driver.findElement(By.id("cardSection"));
    cardSection.click();

    WebElement blockButton = driver.findElement(By.id("blockCard"));
    blockButton.click();

    WebElement successMessage = driver.findElement(By.id("successMessage"));
    assert successMessage.isDisplayed();
}
```

Рисунок 3.9 – Код автоматизація для блокування картки

Час виконання**:** 6 секунд

Результат: Картка заблокована успішно.

- Тест-кейс 10:

```
public void testChangePersonalData() {
    // Назва тесту: Зміна особистих даних
    driver.get("https://api.privat24.ua/");

    WebElement profileTab = driver.findElement(By.id("profileTab"));
    profileTab.click();

    WebElement nameField = driver.findElement(By.id("name"));
    WebElement saveButton = driver.findElement(By.id("saveButton"));

    nameField.clear();
    nameField.sendKeys("New Name");
    saveButton.click();

    WebElement successMessage = driver.findElement(By.id("successMessage"));
    assert successMessage.isDisplayed();
}
```

Рисунок 3.10 – Код автоматизація для зміни особистих даних

Час виконання**:** 7 секунд

Результат**:** Особисті дані успішно змінено

Усі тест-кейси, які були розроблені для автоматизації тестування на сайті Приват24, спрямовані на перевірку основних функцій, які є важливими для користувачів цієї платформи. Загалом, автоматизоване тестування показало швидку ефективність проходження, час проходження 10 тест-кейсів дорівнює 1 хвилину.

Розробка автоматизованих тестів для таких функцій, як вхід у систему, відновлення пароля, переведення коштів, активація картки та інші, забезпечує точність перевірок та дозволяє швидше виявляти дефекти.

Кожен тест-кейс був розроблений таким чином, щоб перевіряти конкретну функціональність системи та її відповідність вимогам. Наприклад, тест на вхід до акаунту забезпечує перевірку коректності авторизації, що є основною частиною будь-якої онлайн-платформи. Тест на неправильний пароль дозволяє перевірити, чи система коректно обробляє помилку при спробі входу з неправильними даними. Відновлення пароля через електронну пошту тестує можливість відновлення доступу до акаунту, що є важливим для безпеки користувачів.

Для фінансових операцій, таких як переведення коштів та виведення коштів, автоматизовані тести перевіряють правильність виконання транзакцій. Важливо забезпечити, щоб усі фінансові операції проходили без помилок, адже навіть незначні збій у цих процесах може призвести до серйозних фінансових наслідків. Тести на перевірку балансу акаунту та історії транзакцій дозволяють переконатися, що всі дані відображаються коректно і відповідають реальним значенням.

Тестування таких функцій, як блокування картки, активація картки та зміна особистих даних, є важливим етапом у забезпеченні безпеки користувачів. Кожна з цих функцій має бути перевірена на можливість несанкціонованих змін або доступу до персональних даних.

Приват24, як популярний сервіс для онлайн-банкінгу, має забезпечувати високу доступність і безпеку своїх послуг. Отже, автоматизовані тестування допомагають не лише підвищити ефективність, а й гарантувати стабільність та безпеку платформи. Всі наведенні тести повинні бути частиною комплексної стратегії тестування, яка включає ручне тестування для перевірки нестандартних ситуацій і автоматизацію для масштабованості та швидкості перевірок.

Автоматизація тестування є ключовим елементом для забезпечення високої якості програмного забезпечення, особливо для таких великих платформ, як Приват24. Вона дозволяє знизити ризики, пов'язані з людським фактором, підвищити швидкість виявлення дефектів і забезпечити стабільну та безпечну роботу системи. Виконання автоматизованих тестів на всіх етапах розробки та впровадження дозволить створити надійну платформу, яка буде задовольняти вимоги користувачів і відповідати високим стандартам якості.

Змішане тестування є важливою частиною сучасних підходів до тестування програмного забезпечення, особливо для великих і складних веб-додатків. Воно поєднує в собі переваги як ручного, так і автоматизованого тестування, що дозволяє забезпечити високу ефективність і гнучкість процесу тестування. У даному розділі ми детально розглянемо, як використання змішаного підходу до тестування може підвищити ефективність перевірки веб-додатку Приват24.

Принципи змішаного тестування полягає в комбінації двох основних методів — ручного та автоматизованого. Даний підхід дозволяє використовувати переваги кожного з методів у залежності від характеру тестованих функцій і складності тесту. Автоматизовані тести чудово підходять для повторюваних завдань, таких як перевірка базових функціональних можливостей або стабільності роботи додатку. Ручне тестування, в свою чергу, є незамінним для перевірки складних сценаріїв, що включають людський фактор, або коли необхідно виконати неформальні тести, які не піддаються автоматизації.

Для проведення змішаного тестування ми використали 10 тест-кейсів, які охоплюють всі важливі функціональні можливості веб-додатку Приват24.

Зокрема, автоматизовані тести було застосовано до тих функцій, які потребують багаторазового повторення, таких як перевірка балансу акаунту, вхід у систему, переведення коштів тощо.

Ручне тестування було застосовано для більш складних сценаріїв, таких як реєстрація нового користувача або зміна особистих даних, оскільки вони потребують взаємодії з користувачем, а також можуть включати етапи, які неможливо автоматизувати.

Всі тести, що передбачають високий рівень повторюваності або взаємодію з простими елементами інтерфейсу, були автоматизовані:

- Вхід у систему;
- Перевірка балансу акаунту;
- Переведення коштів;
- Перегляд історії транзакцій.

Дані функції потребують постійної перевірки кожного разу, коли вносяться зміни до системи, і тому автоматизація значно економить час і зменшує ймовірність людських помилок. вДля перевірки більш складних функцій, таких як:

- Відновлення пароля;
- Зміна особистих даних;
- Активація картки;
- Блокування картки.

було застосовано ручне тестування. Тести не тільки взаємодіють з інтерфейсом, але й часто вимагають додаткових маніпуляцій або перевірок, які важко або неможливо автоматизувати. Такі функції потребують гнучкості та можливості реагувати на різні умови, що може бути досягнуто тільки під час ручного тестування.

Для забезпечення максимального покриття тестуванням було використано інтеграцію автоматизованих і ручних тестів, що дозволило спочатку виконати всі автоматизовані тести, що займають менше часу, і забезпечити стабільність

роботи основних функцій веб-додатку. Потім проводилося ручне тестування, щоб перевірити більш складні сценарії і забезпечити точність та детальність перевірок.

Після виконання змішаного тестування всі 10 тест-кейсів пройшли успішно. В процесі тестування не було виявлено серйозних дефектів або збоїв, що вказує на стабільність і надійність веб-додатку Приват24

Автоматизація повторюваних тестів дозволила значно знизити час, витрачений на перевірку одних і тих самих функцій, і зосередитися на більш складних аспектах. Завдяки автоматизації процесу перевірки основних функцій зменшується ймовірність пропуску дефектів у критичних частинах додатку.

Ручне тестування дозволило більш ретельно перевірити складні сценарії, які важко моделювати в автоматизованих тестах, такі як зміна особистих даних або реєстрація нового користувача.

Загальний час для виконання всіх тестів у змішаному підході склав 1хв 35 секунд, що значно менше порівняно з повністю ручним тестуванням, де для кожного тест-кейсу необхідно було вручну виконати всі кроки. Автоматизація дозволяє швидко виконати перевірку стандартних функцій, таких як вхід у систему чи переведення коштів, що значно зменшує загальний час тестування.

Таблиця 3.3

Тест-кейс	Тип тесту	Час виконання (секунди)
Вхід у систему	Автоматизований	5
Неправильний пароль	Автоматизований	4
Відновлення пароля	Ручний	40
Переведення коштів	Автоматизований	8
Виведення коштів	Автоматизований	6
Перевірка балансу акаунту	Автоматизований	3
Перегляд історії транзакцій	Автоматизований	5
Вихід із системи	Автоматизований	3
Активація картки	Ручний	6
Блокування картки	Ручний	6
Зміна особистих даних	Ручний	25

Таким чином, змішане тестування є найбільш ефективним підходом до перевірки складних веб-додатків, що дозволяє максимально використовувати переваги як ручного, так і автоматизованого тестування для досягнення оптимальних результатів.

3.3 Математично-технічні значення результатів тестування

Для об’єктивного порівняння ефективності ручного, автоматизованого та змішаного тестування використано математичні моделі. Розрахунки виконано за формулами (2.1-2.2) другого розділу, вони враховують час, складність тестів, ймовірність виявлення дефектів та інші показники.

Розрахунок значень відбувався таким чином:

- Ручний метод тестування ПЗ:

Час виконання тест-кейсу

Дано:

T= 44 хв,

C = 1.5 (умовна оцінка складності)

N = 10

$$T_{\text{ручний}} = \frac{44*1,5}{10} = 6,6 \text{хв./тест.}$$

Імовірність виявлення дефекту

Дано:

p=0.15,

N = 10

$$P(D) = 1 - (1 - 0.1)^{10} \approx 0.6513$$

- Автоматизований метод тестування ПЗ:

Час виконання тест-кейсу

Дано:

T= 1 хв,

C = 2 (умовна оцінка складності)

N = 10

$$T_{\text{автоматизований}} = \frac{1*2}{10} = 0{,}3\text{хв./тест.}$$

Імовірність виявлення дефекту

Дано:

p=0.15,

N = 10.

$$P(D) = 1 - (1 - 0.15)^{10} \approx 0.8031$$

- Змішаний метод тестування ПЗ:

Час виконання тест-кейсу

T= 1.23 хв,

C = 2.1 (умовна оцінка складності)

N = 10

$$T_{\text{змішанний}} = \frac{1.23*2.1}{10} = 2{,}4\text{хв./тест.}$$

Імовірність виявлення дефекту

Дано:

p=0.12,

N = 10.

$$P(D) = 1 - (1 - 0.12)^{10} \approx 0.6941$$

Загальні результати прорахунку виконаних видів тестувань наведену у таблиці нижче:

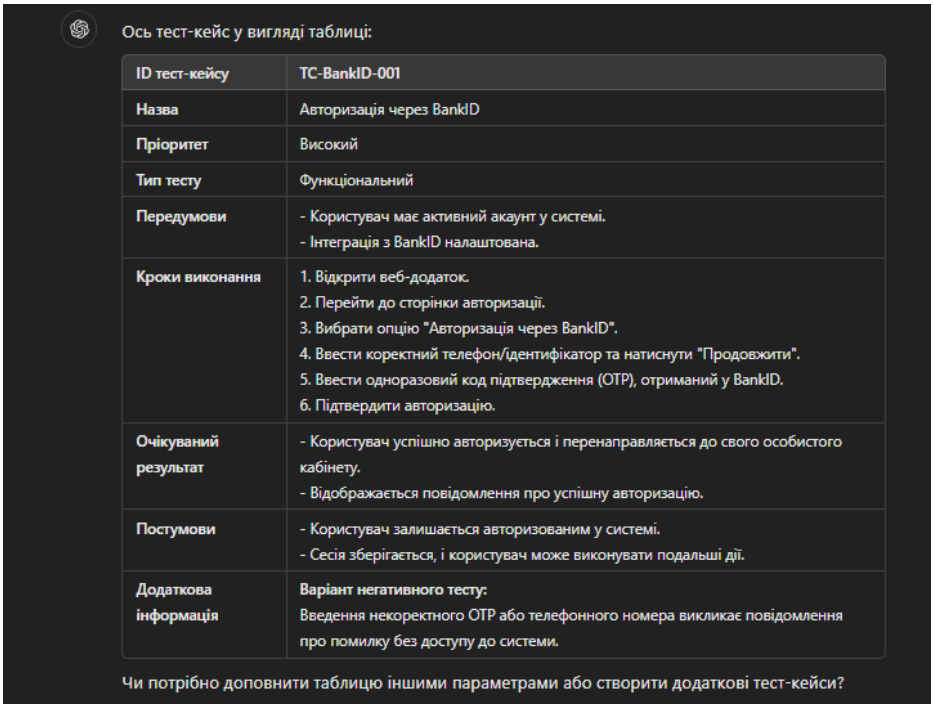
Таблиця 3.4

Показник	Ручне тестування	Автоматизоване тестування	Змішане тестування
Середній час на тест	6.6 хв.	0.3 хв.	2.4 хв.
Ймовірність виявлення дефекту	79%	80%	69%
Загальний час тестування	44 хв.	1 хв.	1.23 хв.

Відповідно до даних із таблиці бачимо, що автоматизований метод тестування виконує тестування тестових сценарії швидше всього. З цього можна зробити висновок, для успішного створення методики універсального тестування програмного забезпечення треба обирати змішаний метод, бо у випадку тестувань UI/UX дизайну даний метод зможе показати універсальність елементів застосуванням ручного тестування, а для частини з функціями краще застосовувати автоматизоване тестування, що з найменший час зможе перевірити більшу частину тестових сценаріїв

3.4. Використання штучного інтелекту в методах тестування

Застосування штучний інтелекту в тестуванні веб-додатків дозволяє вирішити низку проблем, пов'язаних із масштабованістю, ефективністю та точністю. Прикладом застосування у ручному методів тестування передбачає те, що надавши основні моменти або вимоги до створення тест-кейсів GPT система може поступово створювати тест-кейси(0)Рисунок 3.11).



Ось тест-кейс у вигляді таблиці:

ID тест-кейсу	**TC-BankID-001**
Назва	Авторизація через BankID
Пріоритет	Високий
Тип тесту	Функціональний
Передумови	- Користувач має активний акаунт у системі. - Інтеграція з BankID налаштована.
Кроки виконання	1. Відкрити веб-додаток. 2. Перейти до сторінки авторизації. 3. Вибрати опцію "Авторизація через BankID". 4. Ввести коректний телефон/ідентифікатор та натиснути "Продовжити". 5. Ввести одноразовий код підтвердження (OTP), отриманий у BankID. 6. Підтвердити авторизацію.
Очікуваний результат	- Користувач успішно авторизується і перенаправляється до свого особистого кабінету. - Відображається повідомлення про успішну авторизацію.
Постумови	- Користувач залишається авторизованим у системі. - Сесія зберігається, і користувач може виконувати подальші дії.
Додаткова інформація	**Варіант негативного тесту:** Введення некоректного OTP або телефонного номера викликає повідомлення про помилку без доступу до системи.

Чи потрібно доповнити таблицю іншими параметрами або створити додаткові тест-кейси?

Рисунок 3.11– Відповідь від GPT

Якщо переглядити можливість використання ШІ в автоматизованому методу тестування, то існують спеціалізовані інструменти, які використовують штучний інтелект для автоматизації тестування, такі як Mabl(0)Рисунок 3.11Рисунок 3.12). Вони застосовують машинне навчання для створення, виконання та підтримки тестів, зменшуючи потребу у людському втручанні.

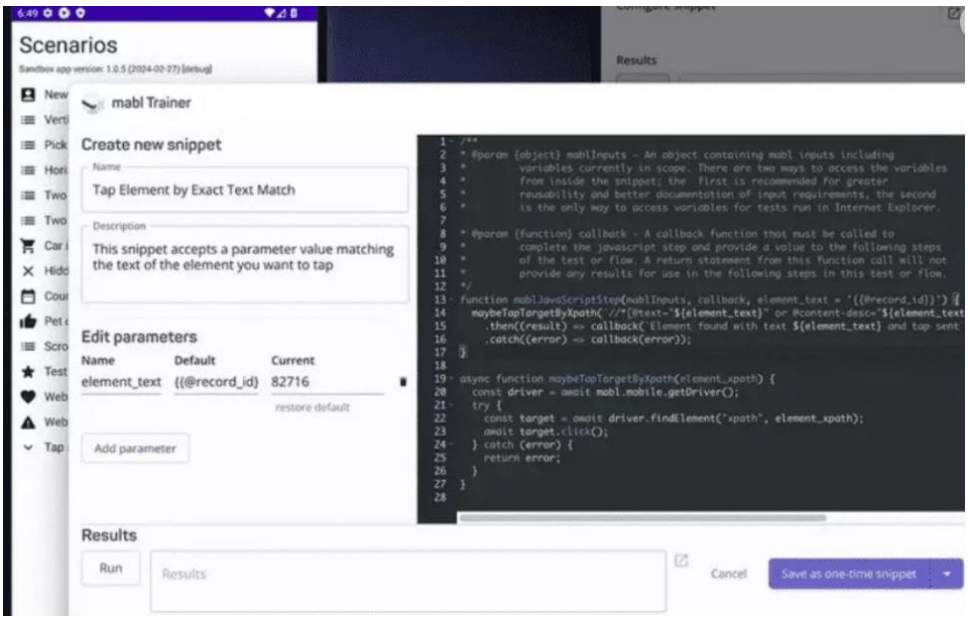


Рисунок 3.12 – Приклад застосування Mabl

Прикладом використання змішаного тестування є поєднання обох видів тестування, тобто, ми можемо одразу використати інструмент Mabl та створювати тест-кейси за допомогою GPT-систем.

Аналіз результатів тестування з використанням ШІ дозволяє виявляти патерни в дефектах, що допомагає точніше класифікувати помилки. Наприклад, класифікація помилок за їх впливом на користувачів дозволяє швидко визначити, які дефекти потребують термінового виправлення. ШІ також активно застосовується для тестування продуктивності. Моделюючи поведінку користувачів у реальному часі, ШІ дозволяє тестувати навантаження на сервери та інші частини системи, оцінюючи їх здатність витримувати велику кількість одночасних користувачів.

Прогнозування дефектів за допомогою ШІ дозволяє виявляти потенційні помилки ще на етапі розробки. Це знижує ймовірність появи критичних дефектів на пізніших етапах тестування. Однією з головних переваг використання ШІ в тестуванні є економія часу. Завдяки автоматизації багатьох процесів, таких як генерація тестів, можна значно скоротити час, що витрачається на рутинні операції. Наприклад, генерація тестових сценаріїв за допомогою GPT займає лише кілька хвилин, що дає змогу тестувальникам зосередитися на більш складних завданнях.

ШІ також сприяє підвищенню точності тестування, знижуючи кількість людських помилок. Оскільки алгоритми працюють за чіткими правилами, ймовірність помилок значно зменшується. Ще однією перевагою є масштабованість. ШІ може адаптуватися до великих проєктів із тисячами тест-кейсів, що робить його ідеальним для великих систем і додатків.

Крім того, ШІ дозволяє прогнозувати ризики, ідентифікуючи ділянки коду, в яких можуть виникнути дефекти. Це дозволяє завчасно зосередити зусилля на найбільш уразливих місцях.

Впровадження ШІ має й певні недоліки. Одним із них є висока вартість впровадження. Розробка та інтеграція рішень на основі ШІ потребує значних ресурсів, як фінансових, так і людських. Також він вимагає великих обсягів

даних для якісного навчання моделей, що може бути проблемою, якщо дані недоступні або неповні.

Залежність від технологій також є значним мінусом. У разі збою системи ШІ можуть виникнути помилки в тестуванні, що ускладнить процес виявлення дефектів. Ще одним обмеженням є те, що ШІ не завжди здатний врахувати всі нюанси складних сценаріїв. Тому в деяких випадках людська інтуїція та творчий підхід можуть бути незамінними.

Використання ШІ в тестуванні дозволяє значно зменшити час виконання тестів, підвищити точність та якість роботи. Однак для повноцінного впровадження таких рішень важливо враховувати початкові витрати та обмеження технологій.

3.5. Висновки до третього розділу

Третій розділ передбачає практичну розробку методики тестування веб-додатків – було розглянуто три основні підходи: ручний, автоматизований, змішаний та розглянуті ключові фактори кожного з видів.

Кожен з них має свої переваги та недоліки, що визначають їх ефективність залежно від конкретної ситуації. Ручне тестування є доцільним для складних або нестандартних сценаріїв, адже воно дозволяє більше уваги приділити специфічним умовам, проте воно вимагає значних часових і людських ресурсів.

Автоматизація тестування дозволяє значно зекономити час, особливо при повторюваних тестах, але потребує початкових витрат на розробку автоматизованих тестів.

Змішаний підхід комбінує переваги обох методів, створюючи оптимальне поєднання ефективності, часу та ресурсів для тестування різних типів функціоналу.

Практичне застосування розробленої методики тестування, виконане в рамках дослідження, показало, що автоматизоване тестування значно знижує час на виконання тестів, але потребує попередніх витрат на налаштування

інструментів і написання коду. Ручне тестування, хоча й більш трудомістке, залишається актуальним для тестування складних сценаріїв, які не можуть бути повністю автоматизовані. Змішане тестування дозволяє зберегти гнучкість, комбінуючи переваги обох методів і оптимізуючи використання ресурсів. Результати порівняння показали, що при застосуванні змішаного підходу можна досягти кращого балансу між якістю та швидкістю тестування.

Враховуючи постійні зміни в технологіях та вимогах до веб-додатків, також була розглянута необхідність створення нової методики тестування, яка буде більш гнучкою та адаптованою до сучасних умов. Така методика повинна включати інтеграцію новітніх технологій, таких як штучний інтелект, для автоматизації рутинних завдань і підвищення ефективності тестування, що дозволить забезпечити більш високу якість тестування за менший час, зменшивши витрати на ресурси. Сучасні методи тестування мають бути адаптовані до змінюваних умов і здатні швидко реагувати на нові вимоги.

Застосування штучного інтелекту в тестуванні відкриває нові можливості для автоматизації не лише самого процесу тестування, а й підготовки тестових сценаріїв, оптимізації їх виконання та аналізу результатів. ШІ може допомогти в передбаченні потенційних дефектів, що дозволить зменшити витрати на виправлення помилок на пізніших етапах розробки.

Загалом, було розроблено методику, яка передбачає використання сукупності методів, підходів, технологій. Впровадження нової методики, побудованій на власному досвіді роботи з тестуванням ПЗ, інтеграція штучного інтелекту та адаптивність до змін забезпечать високий рівень ефективності в процесі тестування і дозволять досягти найкращих результатів.

ВИСНОВКИ

Мета кваліфікаційної роботи передбачає розробку методики для тестування веб-додатків та була успішно досягнута завдяки дослідженню сучасних підходів, методів і технологій тестування. Результатом проведеного дослідження були виявлені особливості, переваги та недоліки кожного з підходів, що дозволило сформувати рекомендації щодо оптимального використання залежно від типу та складності проекту.

В рамках роботи були розглянути такі моменти: проведено огляд сучасних інструментів тестування веб-додатків, проведено порівняння ручного та автоматизованого підходів на основі конкретних кейсів, реалізовано автоматизовані сценарії тестування для імітації реальних сценаріїв використання додатків. Робота включала математично-технічну оцінку ефективності впровадження автоматизації тестування в процес розробки програмного забезпечення.

Реалізація наукової новизни у кваліфікаційній роботі була детально описана та застосована на невеликій кількості тест-кейсів, за допомогою якої вдалося встановити відповідність у кількості витраченого часу та ймовірність виникнення дефекту. Також, були застосовані елементи ШІ, за допомогою яких можливо формувати тестові сценарії, тест-кейси та чек-листи, що також є елементом для спрощення роботи бізнес-аналітиків.

Ручне тестування, як показало дослідження, є ефективним у ситуаціях, коли необхідно перевірити складні або творчі сценарії, наприклад, оцінити користувацький досвід (UX/UI), перевірити поведінку системи в нестандартних умовах або знайти помилки, які важко передбачити алгоритмами автоматизації. Воно є незамінним на етапах ранньої розробки, коли функціонал часто змінюється, а витрати на створення автоматизованих тестів є недоцільними. Однак воно має значні обмеження: високу трудомісткість, суб’єктивність оцінок та обмеженість охоплення.

Автоматизоване тестування, навпаки, забезпечує високу швидкість і точність перевірки повторюваних процесів, таких як регресійне тестування, тестування продуктивності або навантажувальні тести. Інструменти автоматизації, такі як Selenium, що дозволяють автоматизувати великий обсяг завдань і забезпечити швидке виявлення помилок у великих і складних системах. Даний вид тестування виявився надзвичайно ефективним в контексті великих проектів, де значна частина тестових сценаріїв є повторюваною.

Проведений аналіз дозволив оцінити технічно-економічну ефективність автоматизації – використання змішаного тестування дозволяє знизити витрати на тестування на 70%. Окремо варто зазначити математично-технічну доцільність змішаного тестування в довгостроковій перспективі. Хоча початкові витрати на впровадження автоматизації можуть бути значними (розробка скриптів, інтеграція інструментів, навчання команди), вони окуповують себе вже після кількох циклів тестування.

Таким чином, технічно-економічна ефективність даного виду тестування є очевидною, вона дозволяє скоротити час виконання тестів, знизити витрати на робочу силу, мінімізувати ризики помилок і підвищити загальну якість програмного продукту.

На сьогодні, коли темпи розробки програмного забезпечення стрімко зростають, автоматизація стає не розкішшю, а необхідністю для забезпечення конкурентоспроможності компаній на ринку. Результати роботи також включають розробку базових сценаріїв тестування для перевірки різних аспектів роботи веб-додатків, що дозволяє значно підвищити якість програмного продукту, скоротити час на тестування та забезпечити більш високу надійність. Практичне значення отриманих результатів полягає у створенні методичних рекомендацій щодо вибору між ручним і автоматизованим тестуванням, зокрема:

1. Ручне тестування доцільно застосовувати на ранніх стадіях розробки або для перевірки нестандартних сценаріїв використання.

2. Автоматизоване тестування рекомендовано для великих проектів з повторюваними задачами або стабільним функціоналом.

3. Змішане тестування важливо доцільно використовувати для великих проектів з елементами UI/UX тестування.

Перспективи подальших досліджень включають більш детальне вивчення інтеграції автоматизації з системами штучного інтелекту, розширення можливостей автоматизованих інструментів для роботи з нестандартними сценаріями та розвиток нових методів оцінки ефективності тестування. Інновації у цій сфері здатні зробити процес тестування ще більш гнучким, швидким та доступним для команд будь-якого розміру.

Проведені дослідження можуть бути корисними як для практиків у сфері тестування програмного забезпечення, так і для розробників, які прагнуть оптимізувати свої процеси, забезпечити високу якість продуктів і скоротити витрати на розробку та тестування. Таким чином, отримані результати мають значний потенціал для практичного використання, сприяючи розвитку індустрії тестування програмного забезпечення та підвищенню конкурентоспроможності ІТ-компаній.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ISO/IEC/IEEE 15288:2023 URL: https://www.iso.org/standard/81702.html

2. 829-1998 - IEEE Standard for Software Test Documentation URL: https://ieeexplore.ieee.org/document/741968

3. Мартін Р. Чиста архітектура: Мистецтво розроблення програмного забезпечення / пер. з англ. І. Бондар-Терещенко – Харків: вид-во «Ранок»: Фабула, 2021. – 368с

4. Мартін Р. Чистий код: створення і рефакторинг за допомогою Agile / пер. з англ. І. Бондар-Терещенко – Харків: вид-во «Ранок»: Фабула, 2021. – 448с

5. Яків К. Вступ до Розробки Програмного Забезпечення – Київ: 2018. – 149с

6. Луїза Т. Введення у тестування програмного забезпечення. : перевед. з англ. - М.: Видавничий дім "Вільямс", 2018. - 368 с.

7. Майєрс Г. Мистецтво тестування програм/Переклад. з англ. під ред. Б. А. Позина – М: Фінанси та статистика, 2019. – 176с.

8. Рекс Блек. Ключові процеси тестування/Переклад. з англ.: 2018. – 566с.

9. Елфрід Д., Джефф Р., Джон П. Автоматизоване тестування програмного забезпечення/Переклад. з англ. під ред. М. Павлов: 2015. – 589с

10. Куликов С. Тестування програмного забезпечення. Базовий курс: 2024. – 303с

11. Кейнер К., Бах Д.Тестування програмного забезпечення: контекстно орієнтований підхід/Переклад. з англ.: 2020. – 335с

14. Беннетт В. Методи тестування програмного забезпечення / Переклад з англ. – М.: Видавничий дім «Діалектика», 2017. – 420 с.

15. Мур К. Основи автоматизованого тестування / Переклад з англ. – М.: Видавничий дім «Вільямс», 2020. – 310 с.

16. Блум Д. Тестування програмних систем: Принципи та практика / Переклад з англ. – Київ: Лібра Терра, 2016. – 330 с.

17. Кемпбелл Р. Тестування програмного забезпечення: теорія і практика / Переклад з англ. – М.: Фінанси та статистика, 2019. – 512 с.

18. Джеймс К. Техніки тестування програмного забезпечення / Переклад з англ. – Харків: ХНУ, 2020. – 280 с.

19. Кернінгем Д. Програмування з тестуванням / Переклад з англ. – Київ: Видавничий дім «Київська книга», 2018. – 400 с.

20. Кларк Т. Тестування програмного забезпечення в умовах Agile / Переклад з англ. – М.: Фінанси та статистика, 2021. – 450 с.

21. Хайман С. Автоматизація тестування в сучасному розробленому середовищі / Переклад з англ. – М.: Видавничий дім «Російська школа», 2022. – 372 с.

22. Леонард Д. Моделювання та тестування програмних систем / Переклад з англ. – Київ: Наукова думка, 2019. – 324 с.

23. Хартман Т. Основи тестування програмного забезпечення / Переклад з англ. – М.: Видавничий дім "Російський програміст", 2021. – 412 с.

24. Вудс Д. Автоматизація тестування: стратегії та інструменти / Переклад з англ. – Київ: Видавничий дім «Академія», 2018. – 298 с.

25. Штайнберг М. Тестування програмного забезпечення: теорія та практика на прикладі Java / Переклад з англ. – Харків: Видавництво «КСД», 2020. – 360 с.