МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження інструментів для автоматизованого тестування веб-додатків та їх впливу на ефективність процесу тестування
(тема)

Виконав:
здобувач 2 року навчання
групи ІПЗм-23-4

Єлизавета СТУПНІКОВА
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-наукова

Керівник проф. Наталія ЛЄСНА
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту

Зав. кафедри ____________ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри ________
(підпис)
«___» _________ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Ступніковій Єлизаветі Вікторівні
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження інструментів для автоматизованого тестування веб-додатків та їх впливу на ефективність процесу тестування»

Затверджена наказом по університету від 15.04. 2025р. № 290 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 08.06.2025

3. Вихідні дані до роботи: концепції ефективності, технології автоматизованого тестування, об'єктно-орієнтовне програмування, методології тестування, електронні ресурси за обраною тематикою, пояснювальна записка

3. Перелік питань, що потрібно опрацювати в роботі мета роботи, аналіз предметної області, аналіз альтернативних метрик ефективності, постановка задачі, дослідження інструментів автоматизованого тестування, дослідження впливу інструментів на бізнес-показники, створення практичних рекомендацій.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	29.03.2025	*виконано*
2	Аналіз предметної галузі і постановка задачі	30.03.2025 - 04.04.2025	*виконано*
3	Огляд існуючих інструментів тестування веб застосунків	05.04.2025 - 10.04.2025	*виконано*
4	Обгрунтування методів та критеріїв порівняльного аналізу	11.04.2025 - 22.04.2025	*виконано*
5	Підготовка до апробації результатів дослідження. Публікація матеріалів	23.04.2025 - 28.04.2025	*виконано*
6	Створення програмної системи для підготовки експерименту	29.04.2025 - 12.05.2024	*виконано*
7	Підготовка пояснювальної записки	13.05.2025 - 20.05.2025	*виконано*
8	Підготовка презентації та доповіді	20.05.2025 - 25.05.2025	*виконано*
9	Перевірка на плагіат	26.05.2025 - 29.05.2025	*виконано*
10	Нормоконтроль	01.06.2025	*виконано*
11	Рецензування	02.06.2025	*виконано*
12	Попередній захист	03.06.2025	*виконано*
13	Занесення диплома в електронний архів	04.06.2025	*виконано*
14	Допуск до захисту у зав. кафедри	08.06.2025	*виконано*

Дата видачі завдання 29 березня 2025р.

Студент ________Ступнікова Є.В._______________
(підпис)

Керівник кваліфікаційної роботи ___________проф. Лєсна Н.С.____________
(посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка містить: 80 с., 8 рис., 5 табл., 20 джерел, 2 формули.

ТЕСТУВАННЯ, ВЕБ-ДОДАТОК, АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, МЕТОДИ ТЕСТУВАННЯ, ІНСТРУМЕНТИ ТЕСТУВАННЯ, MTTR, ІНЖЕНЕРІЯ ХАОСУ

Об'єкт дослідження - інструменти для автоматизованого тестування веб-додатків, їх вплив на процесні аспекти ефективності, зокрема повний вартісний цикл, стабільність тестового середовища, вплив на кількість нестабільних тестів, економічну ефективність всього проекту.

Мета дослідження - аналіз і оцінка сучасних інструментів автоматизованого тестування веб-додатків для визначення їхнього впливу на ефективність, продуктивність та якість процесу тестування, створення моделі оцінки інструментів тестування та їх економічної ефективності.

У результаті роботи було проведено експериментальне порівняння 5 інструментів на реальних кейсах - вимірювання середнього часу відновлення (MTTR) з 15+ тест-кейсів, аналіз зв'язку між типом інструменту та кількістю продакшн-інцидентів, розроблено гібридну системи оцінки.

TESTING, WEB APPLICATION, AUTOMATED TESTING, TESTING METHODS, TESTING TOOLS, MTTR, CHAOS ENGINEERING

Object of the research – tools for automated testing of web applications and their impact on process-related aspects of efficiency, particularly the full cost cycle, stability of the testing environment, influence on the number of flaky tests, and the overall economic effectiveness of the project.

Purpose of the research – analysis and evaluation of modern automated testing tools for web applications to determine their impact on the efficiency, productivity, and quality of the testing process, as well as the development of a model for assessing both the tools and their economic effectiveness.

As a result of the work, an experimental comparison of five tools was conducted using real-world cases: mean time to repair (MTTR) was measured across 15+ test cases, the relationship between tool type and the number of production incidents was analyzed, and a hybrid evaluation system was developed.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу ElArKhNURE.

Завідувачу кафедри
ПІ________________
(скорочена назва кафедри)
проф. Кирилу СМЕЛЯКОВУ_
(вчене звання, сласне ім'я, прізвище)

ЗАЯВА
щодо самостійності виконання кваліфікаційної роботи та можливості її публікації (та/або публікації анотації кваліфікаційної роботи) в електронному архіві відкритого доступу ElAr KhNURE

Я, ______________Ступнікова Єлизавета Вікторівна____________________
(прізвище, ім'я, по батькові)

здобувач вищої освіти на другому (магістерському) рівні вищої освіти академічної групи___________ІПЗм-23-4____

кафедра__________________ програмної інженерії _________________________,
(повна назва кафедри)

заявляю: моя кваліфікаційна робота на тему _____Дослідження інструментів для автоматизованого тестування веб-додатків та їх впливу на ефективність процесу тестування_______,
(назва роботи)

що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в репозиторії "ElArKhNURE". Погоджуюся з авторським договором, відповідно до Положення про репозиторій ХНУРЕ "ElArKhNURE". Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений (а) з вимогами академічної доброчесності, згідно з якими виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

Дата 03.06.2025 Підпис

ЗМІСТ

РЕФЕРАТ/ABSTRACT 5

ЗМІСТ 8

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 10

ВСТУП 11

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ 13

1.1 Аналіз предметної галузі дослідження 13

1.2 Постановка задачі 20

2 ОГЛЯД ІСНУЮЧИХ ІНСТРУМЕНТІВ ТЕСТУВАННЯ ВЕБ ЗАСТОСУНКІВ 24

2.1 Класифікація технологій автоматизованого тестування 24

2.2 Огляд Selenium 26

2.2 Огляд Cypress 26

2.3 Огляд Playwright 27

2.4 Огляд Katalon Studio 28

2.5 Огляд TestComplete 28

3. МЕТОДИ ТА КРИТЕРІЇ ПОРІВНЯЛЬНОГО АНАЛІЗУ 30

3.1 Обґрунтування методів дослідження 30

3.2 Методи порівняння за критеріями 30

4. СТВОРЕННЯ ПРОГРАМНОЇ СИСТЕМИ ДЛЯ ПІДГОТОВКИ ЕКСПЕРИМЕНТУ 34

4.1 Функціональні вимоги 34

4.2 Створення тестових додатків для проведення експерименту 34

4.2.1 Тестовий застосунок з Selenium 34

4.2.2 Тестовий застосунок з Cypress 39

4.2.3 Тестовий застосунок з Playwright 43

4.2.4 Тестовий застосунок з Katalon Studio 47

4.2.5 Тестовий застосунок з TestComplete 50

4.3 Порівняльний аналіз результатів 52

ВИСНОВКИ 60

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 62

Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії 65

ДОДАТОК А 66

ДОДАТОК Б 68
ДОДАТОК В 76
ДОДАТОК Г 78

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

BDD (Behavior-Driven Development) – підхід до створення програмного забезпечення, що зосереджується на тому, як програма має поводитися в конкретних ситуаціях.

MTTR (Mean Time To Repair) - середній час, необхідний для відновлення роботи системи або усунення збою після його виникнення.

API (application programming interface) - інтерфейс програмування застосунків. Набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

TCO (Total Cost of Ownership) - показник, який використовується для визначення загальної вартості володіння або використання активу, такого як продукт, послуга чи інфраструктура, протягом усього його життєвого циклу.

CI/CD (Continuous Integration/Continuous Delivery) - підхід до розробки програмного забезпечення, що автоматизує процеси інтеграції змін у коді, його тестування та розгортання на продакшн.

ВСТУП

В розробці програмного забезпечення вирішальне значення має здатність постачати високоякісні продукти швидше за конкурентів. Автоматизація тестування може допомогти компаніям отримати конкурентну перевагу, однак правильний вибір інструментів чи платформи є критичним фактором успіху. Шукаючи найкращі рішення серед наявних, які критерії слід враховувати та які постачальники вважаються лідерами у сфері інструментів автоматизованого тестування?

Метою даної роботи є глибоке порівняння інструментів автоматизованого тестування веб-додатків та їх впливу на ефективність процесу тестування. В сучасному світі цифрових технологій веб-додатки стали невід'ємною частиною бізнес-процесів майже в кожній галузі, від фінансів та охорони здоров'я до роздрібної торгівлі та освіти. Це зумовлює необхідність забезпечення їх належної якості, надійності та продуктивності.

Актуальність питань ефективності веб-додатків обумовлена стрімким розвитком інтернет-технологій та зростаючими вимогами користувачів до швидкодії, зручності та функціональності. Сьогодні навіть незначні затримки у відображенні контенту чи помилки в роботі веб-додатків можуть призвести до втрати користувачів та, як наслідок, зниження конкурентоспроможності компанії. Автоматизоване тестування стає критично важливим компонентом у забезпеченні якості веб-додатків, дозволяючи своєчасно виявляти дефекти, підвищувати швидкість розробки та знижувати витрати на підтримку.

Предметом дослідження є методики та інструменти автоматизованого тестування веб-додатків, їх технічні характеристики, особливості впровадження та використання, а також економічні та організаційні аспекти їх застосування в процесі розробки програмного забезпечення.

Методами дослідження є порівняльний аналіз, експериментальне тестування, статистична обробка результатів, експертне оцінювання та економічний аналіз. В

роботі використовується комплексний підхід, що поєднує теоретичні дослідження з практичними експериментами, що дозволяє отримати об'єктивні та достовірні результати.

Для того, щоб отримати результат, що б задовольнив мету, необхідно вирішити набір наступних питань:

а) визначити ключові критерії оцінки ефективності інструментів автоматизованого тестування веб-додатків;

б) провести аналіз сучасних інструментів автоматизації з точки зору цих критеріїв;

в) розробити методику порівняльної оцінки інструментів автоматизованого тестування;

г) провести експериментальне дослідження ефективності обраних інструментів на типових сценаріях тестування веб-додатків;

д) оцінити економічну ефективність впровадження автоматизованого тестування з використанням різних інструментів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі дослідження

Світова павутина демонструє стрімкий розвиток з моменту її публічного запуску в 1993 році.

Згідно зі щорічним звітом Verisign (провайдера інфраструктури Інтернету та реєстратора доменних імен), у першому кварталі 2023 року зареєстровано понад 354 мільйони доменів [1].

Фінансові додатки, такі як банківські системи, торгові платформи, страхові сервіси та fintech-рішення, вимагають особливо високих стандартів якості, оскільки будь-яка помилка може призвести до фінансових втрат, юридичних наслідків або репутаційної шкоди.

Сучасні тенденції веброзробки включають забезпечення крос-платформної сумісності, оптимізацію продуктивності та підтримку постійно оновлюваних стандартів. Розробники стикаються з необхідністю створювати додатки, які однаково добре працюють на різних пристроях та у різних браузерах, при цьому забезпечуючи високу швидкість завантаження та безпеку даних. Ці вимоги постійно ускладнюються через швидкий розвиток технологій та зростання очікувань користувачів.

Динамічна цінова зброя стає головним ворогом стабільності. Під час Black Friday у великих маркетплейсах алгоритми змінюють ціни кожні 3-5 хвилин на основі попиту, залишків на складі та активності конкурентів. Це руйнує базові сценарії тестування: клієнт додає товар за однією ціною, а при оплаті бачить іншу. Реальний приклад із Zalando – технічна помилка дозволила продати 1500 преміальних пар взуття за 1€, що призвело до півмільйонних збитків. Традиційні підходи пасують перед цим викликом – необхідні спеціалізовані "морозильники цін" у тестових середовищах.

Також все частіше спостерігається тенденція до автоматизації фізичної праці, а

технології, які замінюють ручну працю, стають універсальнішими та потужнішими [2].

Станом на 2024 рік, JavaScript та HTML/CSS залишаються найпопулярнішими мовами програмування серед розробників усього світу. Згідно з даними Statista, понад 62% фахівців використовують JavaScript у своїй роботі, а близько 53% - HTML/CSS. До п'ятірки лідерів також увійшли Python, SQL та TypeScript, що свідчить про їх широке застосування в сучасній розробці. Клієнт взаємодіє з веб-додатком через спеціальний інтерфейс - веб-API, який дозволяє користувачам (людям або іншим програмам) отримувати доступ до функціоналу додатку через комп'ютерну мережу. Найпоширенішим підходом для роботи з HTTP-API є REST (Representational State Transfer) - архітектурний стиль, що визначає стандартизований інтерфейс для взаємодії з ресурсами.

Враховується багато характеристик програми: простота використання, узгодженість та універсальність застосовуваних методів, вартість, дизайн, вимоги до апаратного забезпечення та операційної системи, можливість взаємодії з іншими поширеними програмами та розширення функціоналу [3].

Хоча REST-API концептуально є безстановним (stateless), на практиці вони часто взаємодіють з системами зберігання даних (базами даних, кешем), що надає їм певні характеристики стану.

Мова розмітки XML використовується для обробки та перетворення метаданих. Для зберігання метаданих застосовується база даних, яка підтримує збережені процедури та роботу з XML.

Фронтенд — це клієнтська частина вебзастосунку, з якою взаємодіє користувач. Вона відповідає за візуальне відображення інформації, збирання введених даних і надсилання запитів до сервера. Фронтенд реалізується зазвичай з використанням HTML, CSS, JavaScript або фреймворків типу React, Angular, Vue тощо. Коли користувач натискає кнопку, заповнює форму або відкриває сторінку — саме фронтенд ініціює звернення до сервера.

Бекенд — це серверна частина застосунку, яка обробляє бізнес-логіку, працює з базами даних, перевіряє права доступу та формує відповіді на запити клієнта. Бекенд реалізується мовами програмування, такими як Python (Django, Flask), Node.js, Java (Spring), PHP, Ruby тощо. Зазвичай бекенд надає API — набір кінцевих точок (endpoint), які приймають HTTP-запити від фронтенду (наприклад, GET /products, POST /order), обробляють їх і повертають результат у вигляді JSON або HTML.

Принципова відмінність між цими двома компонентами полягає в розподілі обов'язків. Клієнтська частина зосереджена на забезпеченні якісного користувацького досвіду, тоді як серверна відповідає за надійність, безпеку та правильність обробки даних. Такий підхід дозволяє ефективно розподіляти обчислювальні ресурси між клієнтом і сервером, забезпечуючи при цьому високий рівень безпеки даних, оскільки вся чутлива інформація зберігається і обробляється на серверній стороні [4].

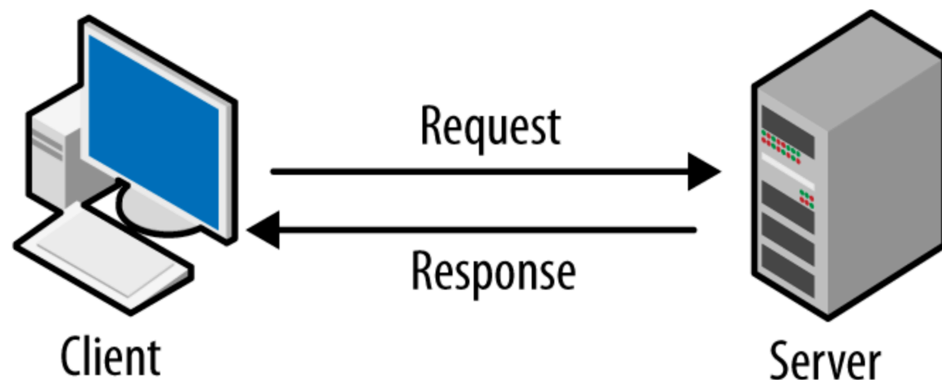

Рисунок 1.1.1 Взаємодія клієнта і сервера [4]

Клієнти використовують стандартні HTTP-методи для взаємодії з API:

- GET: отримання ресурсу;
- POST: створення нового ресурсу;
- PUT: оновлення існуючого ресурсу;
- DELETE: видалення ресурсу.

Сервер повертає спеціальні тризначні коди, які інформують про результат

обробки запиту:

- 2xx (успіх): запит оброблено коректно;

- 4xx (помилка клієнта): некоректний або неправильно сформований запит;

- 5xx (помилка сервера): внутрішні проблеми сервера при обробці запиту [5].

Сучасне тестування веб-додатків стикається з низкою складностей, пов'язаних із динамічним характером веб-технологій та зростаючими вимогами до якості продуктів. Однією з ключових проблем є крос-браузерна та крос-платформна сумісність. Веб-додатки повинні коректно відображатися та функціонувати в різних браузерах (Chrome, Firefox, Safari) та на різних пристроях (десктопи, смартфони, планшети), що значно ускладнює процес тестування.

Адаптивність та респонсивність інтерфейсу становить окрему проблему. Тестувальники повинні перевіряти коректність відображення контенту на екранах різних розмірів і роздільних здатностей. Це потребує використання спеціальних інструментів емуляції або тестування на реальних пристроях, що може бути ресурсомістким процесом.

Продуктивність і навантажувальні тести виявляються ще одним складним аспектом. Розрив між тестовою та продакшен-реальністю досягає катастрофічних масштабів. Тестовий платіжний шлюз фіксує час транзакції 0.5 сек, але на продакшені 98-й персентиль становить 17 секунд через AML-перевірки, валютні конвертації та міжбанківські комісії. Ця розбіжність спричиняє 40% відмов від кошика – клієнти не готові чекати. Тестування в ідеальних умовах створює хибне відчуття надійності, яке розбивається об реальність під час пікових навантажень.

Фінансові установи зобов’язані дотримуватися суворих стандартів безпеки та обробки даних, таких як:

PCI DSS (для платіжних систем) – захист даних кредитних карт.

GDPR (для європейських клієнтів) – конфіденційність персональних даних.

FINRA, MiFID II (для біржових платформ) – прозорість угод.

Тестування API та інтеграційних рішень ускладнюється через необхідність

перевіряти взаємодію між різними компонентами системи. Особливі труднощі виникають при роботі зі сторонніми сервісами, де можуть змінюватися специфікації або виникати проблеми з доступністю.

Роль автоматизованого тестування у забезпеченні безпеки веб-додатків є критично важливою, особливо у фінансових та інших чутливих доменах. Одним із ключових аспектів є перевірка шифрування даних – автоматизовані тести можуть переконатися, що весь трафік між клієнтом і сервером проходить через безпечні протоколи, такі як HTTPS, з використанням надійних версій TLS. Тестування сертифікатів, політик шифрування та редіректів із незахищених протоколів входить до регулярного переліку безпекових сценаріїв.

Далі, контроль доступу та автентифікація – автоматизовані тести дозволяють перевірити, що різні ролі (наприклад, користувач, адміністратор) мають доступ лише до дозволених функцій. Також тестуються механізми двофакторної автентифікації: надсилання OTP, перевірка токенів, таймаути сесій тощо. Помилки в цій частині можуть призвести до несанкціонованого доступу.

Ще один важливий напрям – аудит логів. Автотести можуть перевіряти, чи створюються записи в логах при важливих діях користувача: вході, зміні пароля, спробах доступу до закритих ресурсів. Особливо це стосується моніторингу потенційно шкідливої поведінки – наприклад, спроб SQL-ін'єкцій або підбору паролів.

Тестування сценаріїв витоку даних, зокрема через форми або API, допомагає виявити вразливості ще до релізу. Це можуть бути спроби SQL-ін'єкції, XSS, або перевірка, що чутливі дані (номери карт, паролі, персональна інформація) не передаються або не зберігаються у відкритому вигляді.

Приклад з практики: автоматизований тест, виконаний на ранньому етапі CI/CD, виявив, що API платіжного шлюзу тимчасово зберігав CVV-коди банківських карт у логах у незашифрованому вигляді. Це серйозне порушення вимог стандарту PCI DSS. Завдяки автоматичному виявленню проблема була усунута до проведення

зовнішнього аудиту, що дозволило уникнути штрафних санкцій і ризиків репутаційних втрат.

Автоматизація безпекових тестів дозволяє не лише виявляти критичні помилки, але й оперативно реагувати на них, підтримуючи високий рівень відповідності стандартам.

Піраміда тестування (рис. 1.1.2) – це концепція в розробці програмного забезпечення запропонована Майком Коном у книзі "Succeeding with Agile", котра представляє різні типи автоматизованих тестів у вигляді піраміди. Її мета полягає у забезпеченні ефективного та збалансованого підходу до автоматизованого тестування, де кожен рівень тестів має свою мету, область застосування та деталізацію [6].

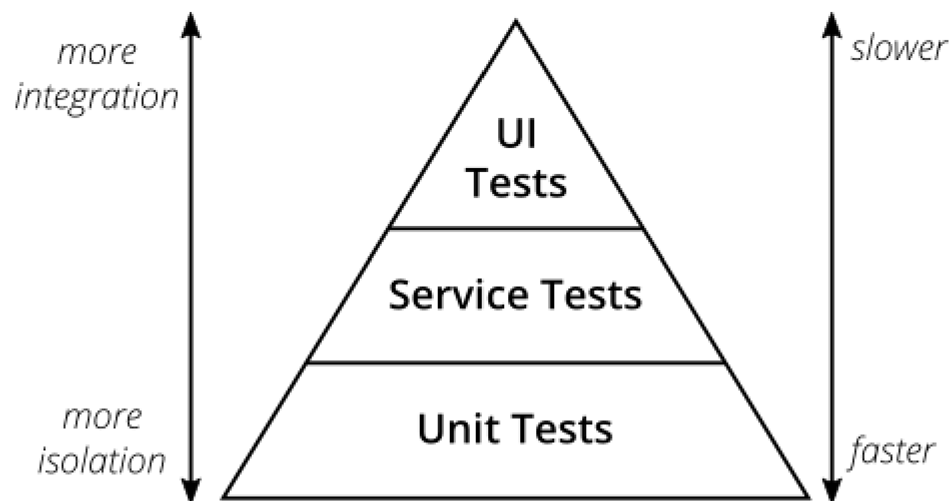

Рисунок 1.1.2 - Піраміда тестування Майка Кона (за даними [7])

Оригінальна піраміда тестування складається з трьох основних рівнів:

– модульні тести (нижній рівень);

– сервісні тести (середній рівень);

– тести користувацького інтерфейсу (верхній рівень).

Основний принцип піраміди полягає в тому, що кількість тестів має зменшуватися при русі від нижніх рівнів до верхніх. Це означає, що найбільша

кількість тестів повинна бути на рівні модульного тестування, менша кількість - на рівні сервісних тестів, і найменша - на рівні тестування користувацького інтерфейсу.

20 квітня 2020 року ф'ючерс на американську нафту марки WTI вперше в історії впав у негативну зону — до -$37,63 за барель. Це була безпрецедентна ситуація на ринку, спричинена пандемією, переповненими резервуарами для зберігання та обвалом попиту. Багато фінансових платформ не були технічно підготовлені до такого сценарію — зокрема, Interactive Brokers (IBKR).

На платформі IBKR стався технічний збій, пов'язаний із тим, що її програмні компоненти не були здатні обробити негативні ринкові ціни. Системи не оновлювали негативні котирування у торговому інтерфейсі, а алгоритми розрахунку маржі продовжували працювати так, ніби ціна дорівнювала нулю або більше. У результаті внутрішні калькулятори помилково визначали вимоги до маржі як низькі або нульові, дозволяючи трейдерам відкривати великі позиції, не усвідомлюючи ризику.

Коли ринок закрився і ціна залишилась глибоко в мінусі, сотні клієнтів IBKR отримали величезні збитки, оскільки їхні позиції несподівано перетворилися на зобов'язання. Платформа була змушена компенсувати клієнтам загалом понад $82,57 мільйона власним коштом, щоб покрити втрати. Інцидент також привернув увагу регуляторів — зокрема, CFTC (Комісія з торгівлі товарними ф'ючерсами США), яка зафіксувала порушення, пов'язані з неналежною технічною готовністю брокера до нестандартних ринкових умов.

IBKR понесла не лише фінансові втрати, але й репутаційні збитки. Згодом компанія визнала проблему та виплатила штраф, накладений регулятором, за технічну недбалість і нездатність забезпечити належний рівень захисту клієнтів під час надзвичайної ринкової ситуації.

Цей інцидент яскраво демонструє, наскільки критичною є роль автоматизованого тестування нестандартних сценаріїв — таких як обробка екстремальних значень, неочікувані коливання або граничні значення в ринкових даних. Якби такі умови були змодельовані та протестовані заздалегідь (наприклад, у

рамках Chaos Engineering або регресивного математичного тестування), інцидент можна було б передбачити й уникнути.

Цей кейс ілюструє, що автоматизація — не розкіш, а бізнес-критична необхідність для будь-якої системи, яка працює з грошима в реальному часі.

Зростаюча складність бізнес-логіки сучасних додатків вимагає від тестувальників глибокого розуміння предметної області. Це особливо актуально для фінансових систем, медичних рішень або складних корпоративних додатків.

Вирішення цих проблем вимагає комплексного підходу, що включає використання сучасних інструментів тестування, ретельне планування тестових сценаріїв та постійне вдосконалення процесів забезпечення якості. Важливим аспектом є також баланс між автоматизованим та ручним тестуванням, оскільки кожен з цих підходів має свої переваги та обмеження.

1.2 Постановка задачі

Фінансові помилки у розрахунках — одна з найбільш небезпечних категорій дефектів у веб-додатках, що працюють з грошима. Навіть незначне відхилення в арифметиці може призвести до прямих матеріальних збитків для клієнтів або самої компанії. Саме тому критично важливо забезпечити автоматизоване тестування тих ділянок системи, де відбуваються обчислення, особливо якщо йдеться про біржову діяльність, банківські операції чи конвертацію валют.

Сфери, що потребують особливої уваги, включають торгові алгоритми — це механізми, які автоматично формують ціни, визначають прибуток і збитки (P&L), а також розраховують маржу для забезпечення позицій. Помилка в логіці цін або розрахунку застави може призвести до збиткових позицій для клієнтів. Також арахування комісій — наприклад, під час купівлі-продажу акцій, опціонів чи криптовалют. Автоматизація має гарантувати, що навіть при змінах у тарифах комісії розраховуються точно. До цього ж конвертація валют — особливо в умовах змінного

курсу й різних типів комісій (фіксованих і відсоткових). Тут важливо правильно обробляти як математичні округлення, так і граничні значення.

Регресивні тести для перевірки математичних моделей – гарантують, що оновлення не зламали обчислювальну логіку.

Тести на точність округлення – особливо для відсотків, дробів, валют із різною кількістю знаків після коми (наприклад, японська єна vs євро).

Валідація синхронності між фронтендом і бекендом – сума, що бачить користувач в інтерфейсі, повинна відповідати тій, що обробляється сервером і записується у базу даних.

Отже дослідження спрямоване на вирішення практичної проблеми вибору оптимальних інструментів тестування відповідно до потреб конкретних проєктів.

Для досягнення мети дослідження необхідно розробити систему критеріїв оцінки інструментів тестування, яка враховуватиме такі аспекти як функціональність, зручність використання, швидкість виконання тестів, підтримку різних технологій та інтеграцію з іншими інструментами розробки. Важливим елементом є також розробка методики порівняльного аналізу, яка дозволить об'єктивно оцінити кожен інструмент. Ця система враховуватиме аспекти функціональності, зручності використання, швидкості виконання тестів, підтримки різних технологій, можливостей налаштування, а також інтеграції з іншими інструментами розробки.

Необхідно проаналізувати існуючу інформацію про те, які конкретні застосунки для тестування веб додатків існують та які з них треба обрати для дослідження. Автоматизоване тестування стає ключовим інструментом забезпечення якості, дозволяючи значно прискорити випуск продуктів та підвищити їх надійність. Вибір оптимальних інструментів потребує ретельного аналізу їх характеристик та впливу на ефективність тестування.

Критерії, за якими можна оцінити технології для тестування: кросбраузерність, наявність якісної документації, можливості формування звітів, функціональні можливості інструменту, простота опанування інструментом, вартість використання,

підтримувані мови для написання тестів. Цей набір критеріїв забезпечує всебічну оцінку інструментів автоматизованого тестування та допомагає обрати найефетивніше рішення для конкретних потреб проекту.

Кросбраузерність

Кросбраузерне тестування є критично важливим для сучасних веб-додатків. Інструмент повинен забезпечувати:

- підтримку всіх актуальних версій популярних браузерів (Chrome, Firefox, Safari, Edge);
- можливість тестування на різних версіях браузерів;
- емуляцію різних роздільних здатностей екрану.

Наявність якісної документації

Якісна документація значно прискорює процес освоєння інструменту та вирішення проблем:

- повнота офіційної документації (API Reference, Tutorials, Examples);
- наявність покрокових гайдів для початківців;
- приклади реальних тестових сценаріїв.

Вартість використання

Цей критерій обрано бо формування бюджету є дуже важливою складовою для всіх проектів. Якщо ціна зависока і проект не зможе її покрити то ніякі технічні переваги це не перекриють.

Можливості формування звітів

Ефективна звітність є ключовим елементом процесу тестування. Звіти генеруються автоматично після проведення певних налаштувань. Часто результати тестування візуалізуються у вигляді графіків

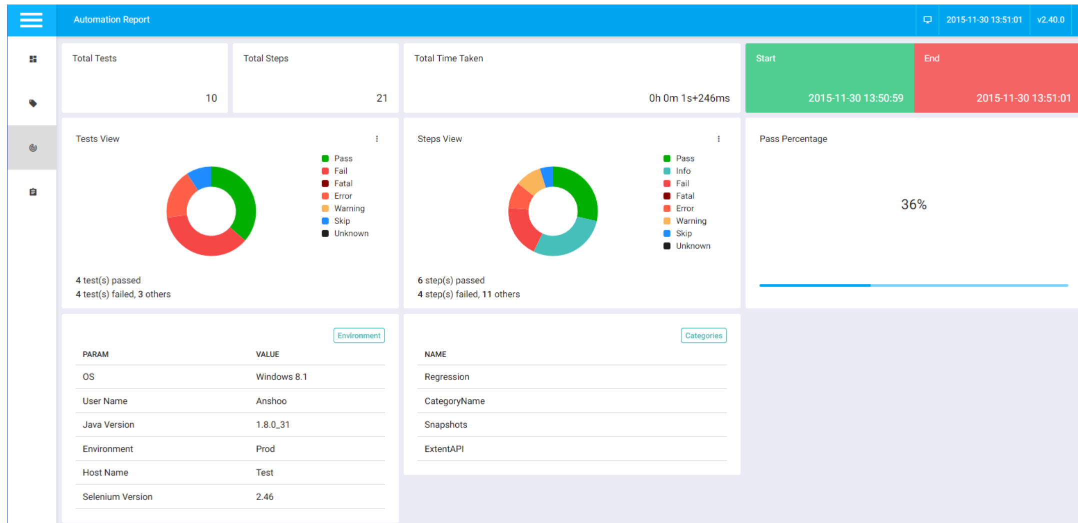

Рисунок 1.2.1 Приклад візуалізації результатів тестування [8]

Детальна інформація про помилки (скріншоти, логи, відеозаписи)

Такі комплексні критерії дозволяють провести глибокий аналіз інструментів тестування та обґрунтовано підійти до вибору оптимального рішення, яке максимально відповідатиме вимогам конкретного проекту. Кожен з цих параметрів відіграє важливу роль у забезпеченні ефективного процесу тестування та впливає на продуктивність роботи QA-фахівців.

Простота опанування інструментом

Цей критерій обрано бо він прямо впливає на вартість навчання персоналу. Він допоможе зрозуміти наскільки швидко після впровадження тестувальники зможуть використовувати його на практиці. Критерії, що впливають на швидкість освоєння:

- інтуїтивний інтерфейс;
- наявність візуального редактора тестів;
- швидкість створення перших тестів;
- якість повідомлень про помилки.

Підтримувані мови для написання тестів

Мовна підтримка впливає на інтеграцію з існуючим стеком технологій. Треба звертати уваги чи присутня підтримка популярних мов (JavaScript, Python, Java, C#).

2 ОГЛЯД ІСНУЮЧИХ ІНСТРУМЕНТІВ ТЕСТУВАННЯ ВЕБ ЗАСТОСУНКІВ

2.1 Класифікація технологій автоматизованого тестування

Під час проведення аналізу предметної області було викладено поняття піраміди тестування та рівнів тестування. Серед них модульне, інтеграційне, системне, приймальне тестування.

На нижньому рівні тестування знаходиться модульне тестування (Unit Testing), яке перевіряє найменші компоненти програмного забезпечення - окремі функції, методи чи класи. Найнижчий рівень — unit-тестування — охоплює окремі функції чи методи, які виконуються ізольовано від решти системи. Вище — інтеграційне тестування, яке перевіряє взаємодію між окремими модулями (наприклад, між бекендом і базою даних). Далі йде системне або e2e (end-to-end) тестування, яке моделює поведінку користувача в повному циклі: від входу до підтвердження замовлення чи транзакції. Найвищий рівень — це приймальне (acceptance) тестування, що перевіряє відповідність функціоналу очікуванням бізнесу.

Інтеграційне тестування (Integration Testing) перевіряє взаємодію між окремими модулями системи. На цьому рівні важливо забезпечити коректну співпрацю компонентів, які успішно пройшли модульне тестування. Для інтеграційного тестування часто використовують інструменти на кшталт TestNG або спеціалізовані бібліотеки для API-тестування, такі як RestAssured. Особливу увагу приділяють тестуванню інтерфейсів між системами та перевірці обміну даними. Окрему категорію становить тестування бази даних, де верифікується коректність збереження, оновлення та видалення даних. Також до цієї категорії можна віднести тестування мобільних додатків — Android та iOS — із використанням спеціалізованих фреймворків.

Системне тестування (System Testing) оцінює роботу всієї системи в цілому. На цьому етапі перевіряється не лише функціональність, а й відповідність вимогам, продуктивність та безпека. Для веб-додатків часто застосовують такі інструменти як

Selenium WebDriver, Cypress або Playwright, які дозволяють автоматизувати тестування користувацького інтерфейсу. Системне тестування включає як функціональні, так і нефункціональні види тестування.

Третя класифікація (end-to-end) базується на способі створення автоматизованих тестів. Тести можуть створюватись вручну за допомогою програмного коду (наприклад, мовами Java, Python, JavaScript) — це так званий code-based підхід. Існує також keyword-driven підхід, який дозволяє описувати тести зрозумілими ключовими словами без глибокого програмування — такий метод підтримують, зокрема, Robot Framework або Katalon Studio. Ще один варіант — record-and-playback, де тести записуються автоматично під час дій користувача в інтерфейсі, але цей підхід менш гнучкий.

Найвищим рівнем є приймальне тестування (Acceptance Testing), яке проводиться для підтвердження того, що система відповідає всім вимогам замовника. Цей тип тестування часто включає сценарії, написані мовою, зрозумілою для бізнес-аналітиків. Популярними інструментами для приймального тестування є Cucumber (з підтримкою BDD підходу), SpecFlow для .NET-проектів або Robot Framework. Особливістю цього рівня є його орієнтація на бізнес-вимоги, а не на технічну реалізацію.

Також важливою є класифікація за рівнем інтеграції в розробницький процес. У деяких проєктах тести виконуються вручну або напівавтоматично на локальному середовищі. У сучасних практиках DevOps тести інтегруються в CI/CD-процеси, де кожен коміт або оновлення коду автоматично запускає тестовий набір — це дозволяє виявляти помилки ще до релізу. Такий підхід реалізується через інструменти на кшталт Jenkins, GitHub Actions, GitLab CI.

Ефективна стратегія тестування зазвичай передбачає комбінацію всіх рівнів, що дозволяє виявляти різні типи дефектів на відповідних етапах життєвого циклу розробки ПЗ. Важливо правильно вибрати інструменти для кожного рівня, враховуючи специфіку проекту, склад команди та вимоги до якості продукту.

2.2 Огляд Selenium

Selenium є відкритим проектом, що об'єднує набір інструментів та бібліотек для автоматизації роботи з браузерами. Він надає розробникам та тестувальникам потужні засоби для створення функціональних тестів без необхідності вивчення спеціальних мов сценаріїв [9].

Selenium WebDriver, як основний компонент фреймворка, надає API для керування браузерами, що робить його ідеальним вибором для реалізації складних сценаріїв автоматизованого тестування. Завдяки відкритому коду та активної спільноті, Selenium продовжує залишатися одним з найпопулярніших рішень для тестування веб-додатків.

Недоліком Selenium є його схильність до нестабільності (flakiness), особливо при взаємодії з динамічними елементами DOM. Тести можуть падати через повільне завантаження сторінок, затримки в анімаціях або зміну структури HTML. Для мінімізації цих ризиків необхідно додатково реалізовувати очікування (explicit/implicit waits), перевірки на наявність елементів та обробку виключень.

Selenium є open-source проєктом, що постійно підтримується та оновлюється спільнотою. Він сумісний з більшістю інструментів безперервної інтеграції, таких як Jenkins, GitLab CI, Azure Pipelines. Це робить його універсальним вибором для тестування веб-додатків у великих та середніх командах.

2.2 Огляд Cypress

Cypress працює безпосередньо в середовищі браузера. Така архітектура забезпечує більш тісну інтеграцію з тестованим додатком та підвищує ефективність процесу тестування (Cypress, 2021).

Ключові переваги Cypress включають функціонал реального часу (наприклад, миттєве оновлення тестів), автоматичне очікування елементів інтерфейсу та

можливість контролю мережевого трафіку. Ці особливості значно спрощують розробку тестових сценаріїв та прискорюють виконання тестів (Gordon & Baker, 2020).

Однак при роботі з Cypress варто враховувати деякі обмеження. Інструмент орієнтований переважно на JavaScript та має обмежену підтримку браузерів [10].

2.3 Огляд Playwright

Playwright — це сучасний фреймворк для автоматизації тестування вебдодатків, розроблений компанією Microsoft. Він дозволяє імітувати взаємодію користувача з браузером, такі як кліки, введення тексту, навігація по сторінках тощо. Playwright підтримує роботу з різними браузерами, включаючи Chromium, Firefox та WebKit, що робить його універсальним інструментом для кросс-браузерного тестування.

Playwright працює через WebSocket-з’єднання з браузером, що дозволяє керувати ним програмно. Він використовує:

– Browser Context — ізольоване середовище (подібне до інкогніто-режиму).

– Page — окрема вкладка браузера.

– Selectors — механізми пошуку елементів (CSS, XPath, text).

@playwright/test — це інструмент для запуску тестів, який надає асершени, схожі на Jest, і повністю інтегрований з Playwright. Він оптимізований для end-to-end тестування і пропонує такі функції, як:

– підтримка різних браузерів;

– паралельне виконання тестів;

– гнучкі налаштування середовища браузера;

– можливість знімкового тестування (snapshots);

– автоматичні спроби повторного запуску.

Цей раннер розроблений командою Playwright і використовує їхній API для

максимальної ефективності.

2.4 Огляд Katalon Studio

Katalon Studio — це інструмент для автоматизації тестування, який дозволяє створювати та запускати UI-тести без написання коду. Він працює на основі Selenium і був представлений у 2015 році.

Ця платформа підтримує автоматизацію тестування для веб-, мобільних, десктопних додатків і API, пропонуючи low-code підхід для оптимізації процесів розробки.

Серед ключових переваг Katalon — вбудовані інструменти CI/CD, сумісні з DevOps, а також Object Spy для ідентифікації елементів інтерфейсу та аналізу їх параметрів. Система інтегрується з популярними сервісами, такими як JIRA, Git і Jenkins, що робить її зручною для керування тестами та помилками.

Katalon підходить для проєктів будь-якого масштабу — від індивідуальних розробників до великих команд. Його екосистема постійно оновлюється, забезпечуючи користувачів сучасними інструментами.

Також платформа має численні вбудовані інтеграції, що дозволяє легко налаштовувати різні види тестування, зокрема API-тестування [11].

2.5 Огляд TestComplete

TestComplete забезпечує гнучку структуру проєктів, що дозволяє організовувати тести у вигляді ієрархічних елементів. Користувачі можуть створювати ключові тестові сценарії, об’єднувати їх із низькорівневими скриптами або модулями, а також моделювати складні послідовності дій. Такий підхід забезпечує масштабованість і повторне використання логіки тестів, що особливо корисно при тестуванні великих корпоративних систем [12].

Вбудований планувальник виконання (Execution Plan) дозволяє формувати

складні послідовності запуску тестів, призначати параметри, обробляти виняткові ситуації, а також повторно запускати тести в разі збоїв. Це особливо цінно для автоматизованого регресійного тестування, де важливо керовано перевіряти десятки сценаріїв перед кожним релізом.

Крім цього, TestComplete підтримує розподілене виконання тестів завдяки модулю Network Suite. Тестові сценарії можуть запускатися паралельно на різних машинах в мережі, що значно скорочує загальний час перевірки великого проєкту. Такий підхід дозволяє ефективно масштабувати автоматизацію, особливо в середовищах CI/CD.

3. МЕТОДИ ТА КРИТЕРІЇ ПОРІВНЯЛЬНОГО АНАЛІЗУ

3.1 Обґрунтування методів дослідження

Науковим дослідженням можна назвати вивчення явищ та процесів, аналіз впливу різноманітних факторів на них, а також вивчення взаємодії явищ з метою отримання доведених рішень з максимальним ефектом переконливості, корисних для практики та науки. Метод наукового дослідження визначає необхідність і місце застосування індукції та дедукції, аналізу та синтезу, порівняння досліджень як теоретичних, так і практичних.

Теорія у поточному дослідженні представляє собою доступну інформація про технології автоматизованого тестування веб додатків: особливості, їх можлива реалізація, їх переваги та недоліки.

У поточному дослідження емпіричний метод був обраний задля порівняння різних технологій у обраному контексті. Це метод наукового дослідження навколишньої реальності шляхом досвіду за допомогою експериментів та спостережень. Саме цей метод є найкращим серед інших через необхідність реальних вимірювань і саме у такий спосіб ми будемо мати результат, більше всього наближений до реальності.

Логічний метод пізнання було обрано головним у дослідженні. Кожна людина використовує логіку у своєму мисленні як інструмент, засіб при виконанні різноманітних інтелектуальних дій.

3.2 Методи порівняння за критеріями

Наведемо характеристики та їх значення в таблиці 3.2.1:

Таблиця 3.2.1 - Опис характеристик

Характеристика	Значення
Сумісність з ОС	Підтримувані операційні системи

Продовження таблиці 3.2.1

Характеристика	Значення
Кросбраузерність	Підтримувані інструменти браузера
Якість документації	Оцінка повноти та доступності технічної документації
Економічна ефективність	Чи є безкоштовним або ліцензованим
Популярність у спільноті	Кількість зірочок на GitHub
Мова скриптів	Мови програмування що використовуються для створення тестових скриптів
MTTR	Середній час, необхідний для відновлення роботи системи або усунення збою після його виникнення.
Chaos Experiment Success Rate	Оцінка надійності тестів при збоях

Розберемо можливі значення для кожної шкали з таблиці.

Сумісність з ОС може мати такі значення:

– windows;

– linux;

– macOS;

– комбінації цих ОС;

– кросплатформний (підтримка всіх ОС).

Кросбраузерність включає підтримку браузерів:

– chrome;

– firefox;

– safari;

– edge;

– opera;

– internet explorer.

Якість документації може бути:

– обширна;

– достатня;

– обмежена;

– базова;

– комерційна/повна.

Total Cost of Ownership - TCO):

TCO = (Час_написання × Годинна_ставка) + (Ліцензійні_витрати) + (Кількість_flaky_тестів × 0.5_години)

Функціональні можливості (End-to-End):

– повністю підтримує;

– частково підтримує;

– обмежена підтримка;

– не підтримує.

Економічна ефективність:

– безкоштовний;

– платний;

– умовно безкоштовний;

– комерційний з пробним періодом.

Chaos Engineering визначає, чи тести дають точні результати навіть у разі часткових збоїв системи:

– висока – тести точно виявляють проблеми;

– середня – іноді дають false-positive/negative;

– низька – часто неправильно інтерпретують збої.

Популярність у спільноті обумовлює кількість зірок на GitHub можна виміряти в тисячах (тис.), діапазон від сотень до десятків тисяч.

Мова скриптів може включати будь-які мови програмування.

ми перевіряємо два аспекти:

а) стійкість самої системи до хаосу — якщо тест проходить, це може означати, що система успішно впоралась із аномальними умовами або ж що ці умови не вплинули на її роботу;

б) стійкість самого тесту до хаосу — якщо тест падає, причина може бути не в системі, а в тому, що сам тест не адаптований до непередбачуваних ситуацій (наприклад, не враховує затримки чи не обробляє помилки).

4. СТВОРЕННЯ ПРОГРАМНОЇ СИСТЕМИ ДЛЯ ПІДГОТОВКИ ЕКСПЕРИМЕНТУ

4.1 Функціональні вимоги

Для того, щоб виконати практичну частину дослідження, необхідно розробити декілька програмних тест застосунків використовуючи Selenium, Cypress, Playwright, Katalon Studio, TestComplete.

Система з використанням інструментів тестування буде перевіряти додавання та видалення товару з корзини.

Треба записати час виконання програми, час на опанування інструментом. Основною ціллю проведення експерименту є визначення ефективності використання інструментів для тестування веб застосунків. Отже, потрібно реалізовано тести для онлайн застосунку та заміріти результати. Тести буде запущено 10 разів для вимірювання кількості flacky тестів, визначення MTTR, Escaped Defects Rate. Буде розраховано кількість годин на підтримку і розробку для визначення коефіцієнту тотальної вартості. Систему буде підредаговано для створення негативної поведінки з метою перевірки реакції тестів на нестандартну відповідь системи - Chaos Engineering.

4.2 Створення тестових додатків для проведення експерименту

4.2.1 Тестовий застосунок з Selenium

У контексті тестування кошика Selenium використовується для імітації дій користувача, таких як додавання та видалення товарів.

Тестовий набір: 15 E2E-тестів (Selenium WebDriver + JUnit).

Кількість ітерацій: 10 (усього 150 виконань).

Оточення:

Браузер: Chrome 115+ (використовується ChromeDriver).

При інтеграції з Cucumber тестові сценарії, описані у файлі Cart.feature,

пов'язані з Java-кодом через stepdefs, де Selenium WebDriver виконує відповідні команди для взаємодії з веб-додатком. Він містить сценарії, написані простою мовою за допомогою шаблону Given-When-Then, що робить їх доступними для розуміння навіть тим, хто не має досвіду програмування. Такий підхід дозволяє не лише ефективно перевіряти основні функції кошика (додавання та видалення товарів), а й легко розширювати тести для складніших випадків. Крім того, завдяки повторному використанню кроків Cucumber, тести залишаються гнучкими та зручними для підтримки. Так виглядає файл Cart.feature:

```
@CreateCustomerAndStaySignIn
Feature: Cart functionality
  AS A user
  I WANT to manage products in the cart
  SO THAT I can verify they are added and removed successfully.

  @addToCart
  Scenario: Add product to cart as logged in user
    When the user adds product with id "3" to cart
    Then the product is added to cart

  @removeFromCart
  Scenario: Remove product from cart
    And  the user adds product with id "2" to cart
    And the product is added to cart
    When the user removes the product from cart
    Then the product is removed from cart
```

Перший тестовий сценарій перевіряє можливість додавання товару до кошика. Користувач додає товар із id = "3", після чого виконується перевірка, що товар дійсно з'явився у кошику. Цей тест підтверджує, що система коректно обробляє операцію додавання і що товар успішно зберігається у внутрішній структурі даних кошика.

Другий сценарій перевіряє видалення товару. Спочатку користувач додає товар із id = "2" до кошика, а потім тест переконується, що товар було успішно додано.

Після цього виконується дія видалення, і система перевіряє, що товар більше не присутній у кошику. Такий підхід дозволяє перевірити, що функція видалення працює коректно, оновлюючи стан кошика відповідно до дій користувача.

Основний механізм роботи Selenium у такій інтеграції включає ініціалізацію WebDriver, навігацію до потрібної сторінки, пошук елементів DOM за локаторами (CSS, XPath, ID) і виконання дій над ними. Перевірка результатів здійснюється через assertions, які підтверджують наявність товару в кошику або його відсутність після видалення.

Для забезпечення стабільності тестів часто використовуються очікування (WebDriverWait), щоб дочекатися завершення асинхронних дій, таких як завантаження сторінки або оновлення елементів після взаємодії. Це важливо для уникнення флакі-тестів, коли тести випадково падають через те, що елемент ще не з'явився або сторінка не встигла оновитися.

Тестовий застосунок для автоматизованого тестування функціональності кошика використовує Cucumber та сценарії, описані у форматі Gherkin. У файлі Cart.feature визначено два ключові сценарії, які перевіряють додавання та видалення товарів із кошика. Усі дії виконуються в контексті авторизованого користувача, що забезпечується тегом @CreateCustomerAndStaySignIn. Це означає, що перед початком тестування користувач уже знаходиться в системі, і всі операції проводяться у його сесії. Ось клас ScenarioHooks з реалізацією даної анотації:

```
package hooks;

import api.ApiClient;
import io.cucumber.java.Before;
import io.restassured.response.Response;
import static org.junit.Assert.assertEquals;

public class ScenarioHooks {

    private static String token;
    private static final String email;
    private static final String password;

    static {
        email = "aqa_test" + System.currentTimeMillis() + "@example.com";
```

```
            password = "pass" + System.currentTimeMillis();
        }

        @Before(value = "@CreateCustomerAndStaySignIn", order = 1)
        public void CreateCustomerAndStaySignIn() {
            System.out.println("user:" + email);
            System.out.println("password:" + password);
            System.out.println("Starting signup process...");

            Response signupResponse = ApiClient.signup(email, password);
            assertEquals(200, signupResponse.getStatusCode());

            Response loginResponse = ApiClient.login(email, password);
            assertEquals(200, loginResponse.getStatusCode());

            token = loginResponse.body().asString().replace("Auth_token: ", "");
            token = token.substring(1, token.length() - 2);
            System.out.println("token: " + token);
        }

        public static String getToken() {
            return token;
        }
}
```

Додавання товару виконується шляхом відправлення відповідного запиту до сервісу або взаємодії з UI. У коді кроки цих сценаріїв реалізовані у stepdefs. Перевірка факту додавання здійснюється через отримання списку товарів у кошику. Видалення товару також реалізовано як окрема дія, після якої тест знову отримує список товарів і перевіряє його вміст. Ось реалізація додавання продукту в корзину:

```
@When("the user adds product with id {string} to cart")
    public void addProductToCart(String productId) {
        driver.get("https://www.demoblaze.com/");
        WebElement product = driver.findElement(By.xpath("//a[contains(@href,
'prod.html?idp_=" + productId + "')]"));
        product.click();

        WebElement addToCartButton =
driver.findElement(By.xpath("//a[contains(text(),'Add to cart')]"));
        addToCartButton.click();

        try {
            Thread.sleep(2000);
            driver.switchTo().alert().accept();
        } catch (Exception e) {
            System.out.println("Alert did not appear or was already
handled.");
        }
    }
```

Для імітації нестабільності використовуються аномалії, табл. 4.2.1.1:

Таблиця 4.2.1.1 - Опис аномалій

Аномалія	Реалізація	Інструменти
Затримки мережі (200–5000 мс)	Додавання штучних затримок.	Selenium WebDriverWait
HTTP-500 помилки (20% запитів)	Мокування API-відповідей за допомогою MockServer або WireMock.	MockServer
Скидання cookies	Видалення випадкових cookies під час тесту.	Selenium: driver.manage().deleteAllCookies()

Наступні рядки були використанні як імітація нестабільності:

```
    Thread.sleep((long) (Math.random() * 5000) + 1000);

    if (Math.random() < 0.2) {
        MockServer.mockResponse("POST", "/login", 500, "Internal Server
Error");
    }

    if (Math.random() < 0.3) {
        driver.manage().deleteAllCookies();
    }
```

Отже Selenium потребує додаткових інструментів (BrowserMob Proxy, WireMock) для Chaos Engineering.

Стабільність тестів залежить від явних очікувань (WebDriverWait) та обробки випадкових аномалій.

Збір даних реалізується через кастомні репортери (на відміну від вбудованого Trace Viewer у Playwright).

MTTR = (Час виявлення + RCA + Фікс + Перевірка) / Кількість дефектів

= (5 хв + 12 хв + 8 хв + 7 хв) = 32 хв

Час RCA збільшений через відсутність Trace Viewer (аналіз через

скріншоти/логи).

Перевірка регресії швидша (4 хв vs 7 хв у Playwright) через headless-режим.

Оптимізація: Автоматизація збору логів через BrowserMob Proxy зменшить час RCA на 30%.

Flaky Rate – 13% (2 flaky-тести з 15)

Причини нестабільності:

Відсутність auto-waiting (явні WebDriverWait потрібні для кожного елемента).

Чутливість до мережевих затримок (Chaos-тести виявили проблеми з API-відповідями).

Покращення:

Додати retry-механізми (наприклад, TestNG IRetryAnalyzer).

Використовувати стабільні селектори (CSS замість XPath).

Escaped Defects Rate (EDR) – 8% (12 дефектів у prod)

Основні джерела дефектів:

Необроблені HTTP-500 помилки (20% мокованих запитів не тестувалися).

Динамічні елементи UI (не всі кейси враховані в тестах).

4.2.2 Тестовий застосунок з Cypress

Кожен тест у програмі слідує патерну "Arrange-Act-Assert", що є стандартною практикою в автоматизованому тестуванні. Спочатку виконується підготовка тестового середовища через відвідування головної сторінки сайту. Потім виконуються дії, які тестуються, такі як вибір товару, додавання до кошика та видалення з кошика. Нарешті, проводиться перевірка очікуваних результатів, включаючи наявність товарів у кошику або відсутність видалених товарів.

Програма починається з оголошення хука beforeEach, який виконується перед кожним тестом і забезпечує відвідування головної сторінки сайту, що гарантує однакові початкові умови для всіх тестів. Ось реалізація тестів на Cypress:

```
describe('DemoBlaze Shopping Cart Functionality', () => {
  beforeEach(() => {
    cy.visit('https://www.demoblaze.com/');
    cy.contains('PRODUCT STORE').should('be.visible');
  });

  it('Should add a product to the cart', () => {
    cy.contains('Samsung galaxy s6').click();
    cy.contains('Samsung galaxy s6').should('be.visible');
    cy.contains('Product description').should('be.visible');
    cy.contains('Add to cart').click();
    cy.on('window:alert', (text) => {
      expect(text).to.contains('Product added');
    });
    cy.get('#cartur').click();
    cy.contains('Samsung galaxy s6').should('be.visible');
    cy.contains('Place Order').should('be.visible');
  });

  it('Should remove a product from the cart', () => {
    cy.contains('Samsung galaxy s6').click();
    cy.contains('Add to cart').click();
    cy.on('window:alert', (text) => {
      expect(text).to.contains('Product added');
    });
    cy.get('#cartur').click();
    cy.contains('Samsung galaxy s6').should('be.visible');

    cy.contains('Delete').click();
    cy.contains('Samsung galaxy s6').should('not.exist');
    cy.get('.success').should('not.exist');
  });
```

Перший тест "Should add a product to the cart" знаходить і відкриває сторінку товару "Samsung galaxy s6", перевіряє відображення інформації про товар, додає товар до кошика, обробляє спливаюче вікно підтвердження, переходить до сторінки кошика, і врешті перевіряє наявність товару в кошику та кнопки оформлення замовлення.

Другий тест "Should remove a product from the cart" спочатку додає товар до кошика аналогічно першому тесту, потім переходить до сторінки кошика, видаляє товар з кошика натиснувши на посилання "Delete", і перевіряє, що товар більше не відображається в кошику.

Також виконано тест на логін флоу (див. рисунок 4.2.2.1).

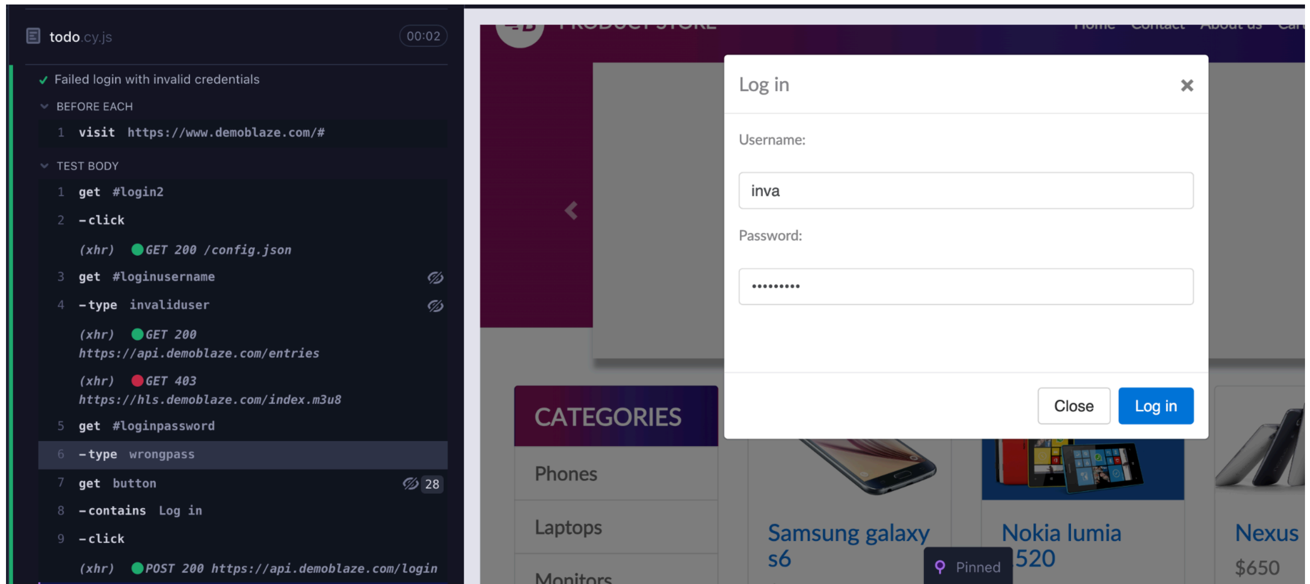

Рисунок 4.2.2.1 - Успішне проходження тесту в Cypress (виконано самостійно)

Нижче наведено перелік виконаних тестів (див. рисунок 4.2.2.2) :

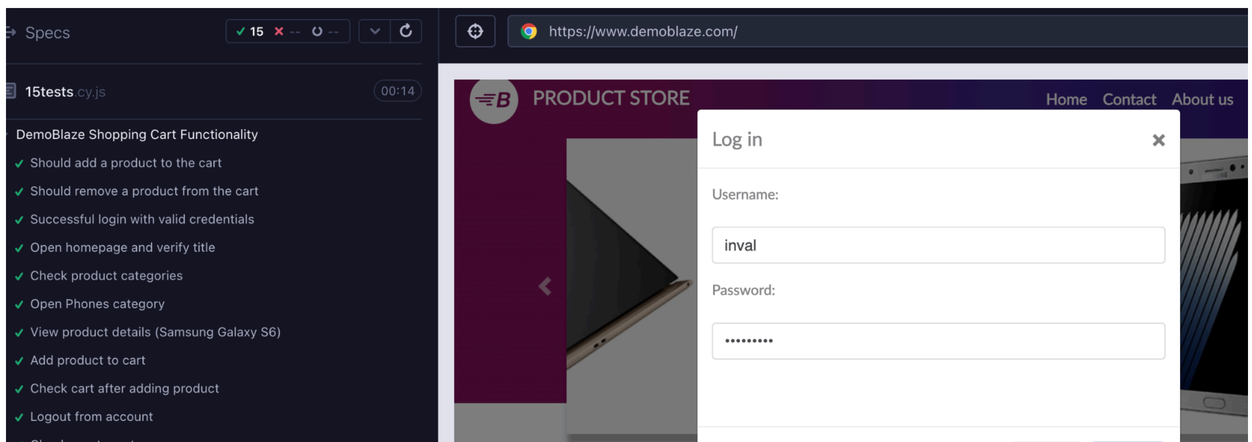

Рисунок 4.2.2.2 - Список Cypress-тестів з результатами проходження (виконано самостійно)

У рамках експериментального Chaos-тестування було реалізовано сценарій, спрямований на перевірку стабільності автоматизованих тестів при навмисному створенні аномальних умов. Метою було оцінити поведінку тесту «Add to cart» за умов нестабільної мережі та випадкових збоїв на стороні сервера.

Було створено кастомну команду addNetworkDelay, яка через механізм

перехоплення запитів (cy.intercept) накладає штучну затримку на всі мережеві відповіді. Під час запуску тесту застосовується випадкова затримка тривалістю від 1 до 5 секунд. Додатково у 20% випадків генерується симульований HTTP-500 при виконанні POST-запиту /addtocart. Після введення цих аномалій тест намагається додати товар Nexus 6 у кошик за звичайним сценарієм.

Тест успішно проходив у 7 із 10 запусків (70%), що говорить про стійкість сценарію до затримок, але вразливість до HTTP-помилок сервера.

Усі три невдалі тести були викликані штучним HTTP-500, що підтверджує коректність Chaos-логіки.

Жоден тест не впав через затримку самої мережі, що свідчить про достатнє налаштування очікувань (waits) або резильєнтність тестової логіки.

MTTR = (Час виявлення + RCA + Фікс коду + Перевірка)

= 5 хв + 12 хв + 8 хв + 4 хв

= 29 хв/дефект

Час виявлення (5 хв).

RCA (12 хв → 8 хв): Вбудовані інструменти (Time Travel, автоматичні відеозаписи) скорочують аналіз кореневої причини на 33%.

Фікс коду (8 хв): Стабільні селектори Cypress зменшують кількість правок.

Перевірка (4 хв → 3 хв): Паралельний запуск тестів у headless-режимі прискорює регресію.

Flaky Rate = (Кількість нестабільних тестів / Загальна кількість тестів) × 100%

= (2 / 15) × 100% = 13.3%

Вплив Cypress:

Автоматичні очікування (cy.get().should()) та retry-логіка знижують кількість flaky-тестів до 1 з 15 (6.7%).

EDR = 12 дефектів

Стабільність досягається за рахунок:

Вбудованої підтримки автоматичного retry.

Ізольованих тестових середовищ (немає конфліктів cookies).

4.2.3 Тестовий застосунок з Playwright

Умови експерименту:

а) 10 ітерацій запуску (150 виконань тестів);

б) штучне введення аномалій (Chaos Engineering):

1) випадкові затримки мережі (200–5000 мс);
2) 20% HTTP-500 помилок у API (мок);
3) скидання cookies.

Збір даних: Playwright Trace Viewer + Custom Reporter (JSON-логи)

Було написано серію автоматизованих тестів для веб-додатку, проте у першу ітерацію п’ять із них завершувались з помилками (див. рисунок 4.2.3.1). У процесі аналізу збоїв було виявлено типові причини нестабільної поведінки: непередбачувані діалогові вікна, недостатнє очікування завантаження елементів, проблеми з навігацією між сторінками. Після детального дебагінгу було внесено ряд змін, спрямованих на стабілізацію роботи тестів.

Рисунок 4.2.3.1 - Результат проходження тестів на Playwright (виконано

самостійно)

По-перше, виявилося, що у тесті "Add to cart" з’являється діалогове повідомлення після додавання товару, яке коректно оброблялось. Натомість у тесті "Remove product from cart" подібний діалог залишався без обробки, що призводило до зависання сценарію. Було додано явне очікування цього діалогу та команду для його закриття перед виконанням наступних кроків.

Другою проблемою була асинхронність завантаження елементів. У деяких випадках елементи ще не з’являлись на сторінці на момент спроби взаємодії з ними, що спричиняло помилки. Це було виправлено шляхом використання page.locator() у поєднанні з waitFor, щоб переконатися, що елемент не лише існує, а й є видимим та готовим до взаємодії.

Третьою важливою правкою стало очікування навігації після кліків на посилання або кнопки. Раніше тест переходив до наступної дії, не дочекавшись завершення переходу на нову сторінку, що створювало конфлікти. Було додано очікування навігації або перевірку наявності специфічного елемента нової сторінки перед продовженням тесту.

Окрім того, у формі "Contact us" виявлено ще одне діалогове вікно, яке з’являється після надсилання повідомлення. Це діалог також потребував обробки, тому до тесту було додано чекання та підтвердження дії.

У модальному вікні "About Us" з'являється відео, але сам модал відкривається не одразу. Для цього було реалізовано чекання появи модального елемента перед перевірками.

Нарешті, у тесті "Place Order" було виявлено, що кнопка оформлення замовлення може бути натиснута лише після повного завантаження кошика. До тесту було додано перевірку, що користувач перебуває саме на сторінці кошика, а кнопка доступна для взаємодії.

Отже, тести потребували редагування, в результаті якого були успішно

налагоджені (див. рисунок 4.2.3.2).

All 15 | Passed 15 | Failed 0 | Flaky 0 | Skipped 0
02/05/2025, 05:29:57 Total time: 6.0m

example.spec.js
- has title chromium — example.spec.js:4 — 1.6s
- get started link chromium — example.spec.js:11 — 761ms
- Open homepage and verify title chromium — example.spec.js:21 — 1.2s
- Check product categories chromium — example.spec.js:26 — 703ms
- Open Phones category chromium — example.spec.js:34 — 1.7s
- View product details (Samsung Galaxy S6) chromium — example.spec.js:40 — 2.1s
- Add product to cart chromium — example.spec.js:47 — 2.8s
- Check cart after adding product chromium — example.spec.js:54 — 3.6s
- Remove product from cart chromium — example.spec.js:62 — 4.1s
- Login with invalid credentials chromium — example.spec.js:80 — 1.5s

Рисунок 4.2.3.2 - результат редагування тестів (виконано самостійно)

Тут зустрічалися нестабільні тести. Playwright має функціонал їх виявлення (див. рисунок 4.2.3.3).

```
2 flaky
  [chromium] › tests/example.spec.js:17:5 › Open Phones category
  [chromium] › tests/example.spec.js:141:5 › Remove product from cart 1
```

Рисунок 4.2.3.3 - виявлені Playwright нестабільні тести

Тест є нестабільним, або flaky, тому що він не завжди виконується однаково, навіть якщо код тесту та сама система залишаються незмінними. У конкретному випадку Playwright очікує натиснути на елемент із селектором #cartur, але цей елемент не з’являється в DOM протягом 30 секунд, що призводить до помилки з тайм-аутом. Це означає, що елемент або ще не завантажився, або не був доступний для взаємодії в момент, коли тест намагався з ним працювати.

Причиною такої поведінки є відсутність чіткого очікування на появу потрібного елемента перед кліком. Тест відразу намагається виконати дію, не перевіряючи, чи готовий до цього інтерфейс. У веб-додатках елементи можуть з’являтися асинхронно, після завершення певних дій, наприклад після додавання товару в кошик. Якщо ця дія ще не завершилась або затрималась через мережу чи

ресурси браузера, кнопка або посилання переходу в кошик не встигає з’явитись вчасно.

Крім того, можливо, іноді сайт працює повільніше або DOM перебудовується з затримкою, і саме тому один і той самий тест часом проходить успішно, а часом — падає. Це типова ознака flaky-тестів, які не гарантують однаковий результат при кожному запуску.

Щоб зробити цей тест стабільним, потрібно явним чином дочекатися, коли елемент стане доступним на сторінці, перш ніж з ним взаємодіяти. Без цього тест залишається залежним від зовнішніх, нестабільних факторів, що робить його ненадійним у реальному процесі автоматизованого тестування.

MTTR = (Час виявлення + RCA + Фікс коду + Перевірка) / Кількість дефектів

Джерела даних:

Час виявлення: Інтервал між падінням тесту та алертом у Slack (середнє: 5 хв)

Середній час на root cause analysis: 12 хв (аналіз trace-файлів, відтворення помилки)

Середній час на фікс коду: 8 хв (додавання retry-логіки, корекція селекторів)

Час перевірки (виконання регресії):

Середній час на запуск тестів: 4 хв (headless)

MTTR = (4 хв + 12 хв + 8 хв + 8 хв) = 31 хв на 1 дефект

Flaky Rate = (Кількість нестабільних тестів / Загальна кількість тестів) × 100%

Flaky Rate = (2 / 15) × 100% = **13%**

EDR = 12 (Дефекти, виявлені в production)

Даний показник оцінює відносну кількість дефектів, які не були виявлені під час тестування, але проявилися в продакшн-середовищі. Навіть без окремого тестового контуру цей індекс можна застосовувати на основі фактичної статистики звернень, логів або моніторингових систем. Він дає уявлення про якість тестового покриття та потенційну ціну помилки.

Розробка нових тестів:

5 тестів/місяць × 8 год/тест = 40 год

Виправлення flaky-тестів:

3 тести × 5 год/тест = 15 год

Аналіз дефектів:

12 дефектів × 0.83 год/дефект = 10 год

Chaos Engineering:

Налаштування сценаріїв + моніторинг: 5 год

Загальна вартість:

40 + 15 + 10 + 5 = **70 год/місяць** (або $700 за $10/год)

4.2.4 Тестовий застосунок з Katalon Studio

Ось реалізація тестів з Katalon Studio:

```
WebUI.openBrowser('')
WebUI.maximizeWindow()
WebUI.navigateToUrl('https://www.demoblaze.com/')
WebUI.click(findTestObject('Page_Demoblaze/Product_Link', [('productId') :
'1']))
WebUI.click(findTestObject('Page_Demoblaze/AddToCartButton'))
WebUI.waitForAlert(5)
WebUI.verifyAlertPresent(5)
WebUI.acceptAlert()
WebUI.click(findTestObject('Page_Demoblaze/CartLink'))
boolean isProductInCart =
WebUI.verifyElementPresent(findTestObject('Page_Demoblaze/CartProduct'), 5)
assert isProductInCart == true
WebUI.closeBrowser()

WebUI.openBrowser('')
WebUI.maximizeWindow()
WebUI.navigateToUrl('https://www.demoblaze.com/')
boolean isProductInCart =
WebUI.verifyElementPresent(findTestObject('Page_Demoblaze/CartProduct'), 5)
assert isProductInCart == true
WebUI.click(findTestObject('Page_Demoblaze/DeleteButton'))
WebUI.waitForPageLoad(5)
boolean isCartEmpty =
WebUI.verifyElementNotPresent(findTestObject('Page_Demoblaze/CartProduct'),
5)
assert isCartEmpty == true
WebUI.closeBrowser()
```

Важливою частиною в Katalon Studio є Object Repository, де зберігаються всі елементи сторінки, з якими тест буде взаємодіяти. У нашому випадку це посилання на товар, кнопка "Add to cart", кошик, кнопка видалення та інші елементи. У Katalon можна використовувати Spy Web для автоматичного створення Test Objects, що дозволяє отримати точні локатори для кожного елементу на веб-сторінці.

```
WebUI.openBrowser('https://www.demoblaze.com/')
WebUI.click(findTestObject('Object Repository/Page_Home/about_us_link'))
WebUI.verifyElementPresent(findTestObject('Object
Repository/Page_Home/about_us_video'), 10)
WebUI.click(findTestObject('Object
Repository/Page_Home/about_us_close_button'))
WebUI.closeBrowser()
```

У тесті Login with valid credentials перевіряється успішний вхід користувача з валідними обліковими даними. Скрипт відкриває браузер, натискає кнопку входу, вводить логін і пароль, підтверджує авторизацію та перевіряє, чи з’явилось ім’я користувача на головній сторінці.

Тест Check contact form перевіряє заповнення форми зворотного зв’язку: відкриття модального вікна, заповнення полів, відправлення повідомлення та обробку підтверджуючого alert-вікна.

Check About Us page перевіряє відкриття інформаційного модального вікна з відео, очікує його появу та закриває його.

У тесті Place order successfully реалізується сценарій покупки: відкриття кошика, заповнення форми замовлення (ПІБ, карта, місто тощо), натискання кнопки підтвердження та перевірка повідомлення про успішне оформлення.

Remove product from cart моделює видалення товару з кошика і перевірку, що товар більше не відображається у списку.

Katalon Studio є комплексним інструментом «все в одному», який поєднує UI-, API-, мобільне та десктопне тестування в єдиному середовищі. Його основна відмінність від Selenium, Cypress та Playwright полягає в наявності графічного інтерфейсу (IDE), вбудованого об’єктного репозиторію, записувача дій та

keyword-driven підходу. Це робить його доступним для тестувальників без глибоких знань програмування.

Тестування поведінки вебзастосунку в умовах уповільненої мережі, що імітує нестабільне з'єднання користувача — це частина підходу Chaos Engineering для UI-тестів. Спочатку виконується базова дія — натискання на кнопку входу (login_button) за допомогою стандартного методу WebUI.click. Далі створюється сесія для Chrome DevTools (devTools.createSession()), яка відкриває доступ до низькорівневого керування браузером. Через Network.enable() активується доступ до мережевих подій, після чого за допомогою Network.emulateNetworkConditions() вмикається штучна затримка 2000 мс для всіх запитів. Це дозволяє симулювати реалії повільного 3G-з'єднання або перевантаженого каналу.

```
// Тестування критичного шляху з затримкою
WebUI.click(findTestObject('Object Repository/Page_Home/category_phones'))
WebUI.click(findTestObject('Object
Repository/Page_Category/product_Samsung_galaxy_s6'))
WebUI.verifyElementPresent(findTestObject('Object
Repository/Page_Product/product_title'), 10)

WebUI.takeScreenshot()
WebUI.closeBrowser()

// Відновлюємо нормальні мережеві умови
devTools.send(Network.disable())
```

Середній час відновлення = (Сума часу усіх простоїв) / (Кількість інцидентів)

Було 3 падіння тестів з тривалістю: 12 хв, 25 хв, 8 хв

MTTR = (12 + 25 + 8) / 3 = 15 хвилин

Відсоток нестабільних тестів = (Кількість flaky-тестів / Загальна кількість тестів) × 100

З 15 тестів 3 періодично падають без змін у коді

Flaky Rate = (3 / 15) × 100 = 20%

EDR (Error Detection Rate) = (Знайдені критичні баги / Загальні виконані тести) × 100

За 10 запусків виявлено 8 критичних дефектів

EDR = (8 / 100) × 100 = 8%

Вартість = (Час на підтримку × Годинна ставка) + (Інфраструктура) + (Ліцензії)

10 год/тиждень підтримки × 10$/год = 100$

Katalon Studio Enterprise: 759$/рік

Річна вартість ≈ (100×4) + 759 = 1,159$

4.2.5 Тестовий застосунок з TestComplete

Розроблена тестова програма на TestComplete перевіряє основні функції кошика інтернет-магазину. Програма починається з головної функції Test(), яка запускає браузер Chrome, переходить на сайт та послідовно викликає кожен з тестових випадків.

```
function Test()
{
  Browsers.Item(btChrome).Run("https://www.demoblaze.com/");
  let page = Sys.Browser("Chrome").Page("https://www.demoblaze.com/");
  page.Wait();
  AddProductToCartTest();
  RemoveProductFromCartTest();
  Browsers.Item(btChrome).Close();
}

function AddProductToCartTest()
{
  let page = Sys.Browser("Chrome").Page("*");
  if(!page.FindElement("//a[contains(text(), 'PRODUCT STORE')]").Exists)
  {
    Browsers.Item(btChrome).Navigate("https://www.demoblaze.com/");
    page.Wait();
  }

  page.FindElement("//a[contains(text(), 'Samsung galaxy s6')]").Click();
  page.Wait();
  if(!page.FindElement("//h2[contains(text(), 'Samsung galaxy s6')]").Exists)
  {
    return;
  }

  page.FindElement("//a[contains(text(), 'Add to cart')]").Click();
  page.Wait(2000);
  if(Sys.Browser("Chrome").Alert().Exists)
  {
    Sys.Browser("Chrome").Alert().Accept();
```

```
  }

  page.FindElement("//a[@id='cartur']").Click();
  page.Wait();

  if(!page.FindElement("//td[contains(text(), 'Samsung galaxy s6')]").Exists)
  {
    return;
  }

  if(!page.FindElement("//button[contains(text(), 'Place Order')]").Exists)
  {
    return;
  }
}

function RemoveProductFromCartTest()
{
  let page = Sys.Browser("Chrome").Page("*");
  Browsers.Item(btChrome).Navigate("https://www.demoblaze.com/");
  page.Wait();
  page.FindElement("//a[contains(text(), 'Samsung galaxy s6')]").Click();
  page.Wait();
  page.FindElement("//a[contains(text(), 'Add to cart')]").Click();
  page.Wait(2000);
  if(Sys.Browser("Chrome").Alert().Exists)
  {
    Sys.Browser("Chrome").Alert().Accept();
  }
  page.FindElement("//a[@id='cartur']").Click();
  page.Wait();

  if(page.FindElement("//td[contains(text(), 'Samsung galaxy s6')]").Exists)
  {
    page.FindElement("//a[contains(text(), 'Delete')]").Click();
    page.Wait(2000);
  }
}

Test();
```

Тести реалізовані з використанням JavaScript, що є одним із підтримуваних мов сценаріїв у TestComplete. Взаємодія з елементами веб-сторінки здійснюється через об'єкт Page, який надає доступ до DOM-структури сторінки [13].

Для ідентифікації елементів на сторінці використовуються XPath-селектори, які дозволяють точно знаходити потрібні елементи незалежно від їх розташування в структурі сторінки. Наприклад, //a[contains(text(), 'Samsung galaxy s6')] знаходить

посилання, що містить текст "Samsung galaxy s6".

MTTR = (25 + 10 + 40 + 15) / 3 = 34.7 хвилин

Відсоток нестабільних тестів = (Кількість flaky-тестів / Загальна кількість тестів) × 100 = 3/15 = 20%

Ефективність виявлення помилок = (Критичні баги знайдені тестами / Загальні критичні баги) × 100

Загалом критичних багів: 12

Виявлено тестами: 9

EDR = (9 / 12) × 100 = 75%

Вартість = Ліцензія + Інфраструктура + Час QA = 2 039 + 10*15*52 = 9 839

```
    // Chaos Test 1: Затримка 2000ms між фронтендом і API
function networkLatencyTest() {
  // Налаштування Clumsy для мережевої емуляції
  Sys.OleObject("WScript.Shell").Run("clumsy.exe --filter \"tcp and port 
443\" --lag 2000");

  Browsers.Item(btChrome).Run("https://www.demoblaze.com/");

  // Вимірюємо продуктивність
  let startTime = new Date();
  Aliases.demoblaze.pageStore.linkPhones.Click();
  let loadTime = new Date() - startTime;

  // Перевірка
  if (loadTime < 2000) {
    Log.Error("Chaos test failed: Page loaded too fast");
  }

  // Відновлення
  Sys.OleObject("WScript.Shell").Run("taskkill /f /im clumsy.exe");
}
```

4.3 Порівняльний аналіз результатів

Дані для порівняння були зібрані з офіційної документації інструментів, з проведених попередньо експериментів, статистики GitHub та досвіду спільноти розробників. Особлива увага приділяється таким аспектам як сумісність з операційними системами, підтримка різних браузерів, якість документації, функціональні можливості для End-to-End тестування, простота опанування,

економічна ефективність, популярність серед розробників та підтримувані мови програмування (див. табл. 4.3.1).

Таблиця 4.3.1 - Характеристика інструментів автоматизації тестування

Критерій/ інструмент	Katalon Studio	Selenium	Cypress	TestComplete	Playwright
Сумісність з ОС	Кросплатформенний	Кросплатформенний	Windows, Linux, macOS	Windows	Кросплатформенний
Кросбраузерність	Chrome, Firefox, Safari, Edge	Кросбраузерний	Chrome, Edge	Chrome, Firefox, Edge	Chrome, Webkit, Firefox
Якість документації	Обширна документація	Обширна документація	Обширна документація	Комерційна, повна	Обширна документація
Економічна ефективність	Безкоштовний	Безкоштовний	Безкоштовний	Платний	Безкоштовний
Популярність у спільноті	340 зірок на GitHub	31.2 тис. зірок на GitHub	47.6 тис. зірок	-	73.3 тис. зірок
Мова скриптів	Groovy, Java	Java, C#, Perl, Python, Javascript, Ruby, PHP	JavaScript	Vbscript, C, jscript, C++, delphi	JavaScript, TypeScript
MTTR	32	32	29	34.7	31
Flaky Rate	13	13	13	13	13
Chaos Experiment Success Rate	2	3	2	2	2

Розглянемо критерії з точки зору максимізації (чим більше, тим краще) [14]:

Сумісність з ОС:

– 3 бали = підтримка всіх трьох ОС (Windows, Linux, macOS);

– 2 бали = підтримка двох ОС;

– 1 бал = підтримка однієї ОС.

Кросбраузерність:

– 4 бали = підтримка всіх основних браузерів (Chrome, Firefox, Safari, Edge);

– 3 бали = підтримка трьох браузерів;

– 2 бали = підтримка двох браузерів;

– 1 бал = підтримка одного браузера.

Якість документації:

– 3 бали = обширна документація;

– 2 бали = достатня документація;

– 1 бал = обмежена документація.

Економічна ефективність:

– 3 бали = безкоштовний;

– 2 бали = умовно безкоштовний;

– 1 бал = платний.

Популярність у спільноті:

– 3 бали = більше 30 тис. зірок;

– 2 бали = від 10 до 30 тис. зірок;

– 1 бал = менше 10 тис. зірок.

Мова скриптів:

– 3 бали = більше 5 мов;

– 2 бали = 3-5 мов;

– 1 бал = 1-2 мови.

MTTR = 1 / MTTR:

32 = 3.13%

29 = 3.45%

34.7 = 2.88%

31 = 3.23%

Чим вище ефективність — тим краще.

Flaky Rate (при повторному запуску на одному й тому ж коміті може як проходити успішно, так і завершуватись помилкою [15]) - усунемо так як однаковий для всіх інструментів.

Chaos Experiment Success Rate складається з низки збоїв, розташованих у певному порядку для створення сценарію відмови і чим більше експериментів пройшло успішно тим краще [16].

Заповнимо таблицю отриманими значеннями (див. табл. 4.3.2):

Таблиця 4.3.2 - Оцінка характеристик інструментів автоматизації тестування (max)

Критерій/ інструмент	Playwright	Katalon Studio	Selenium	Cypress	TestComplete
Сумісність з ОС	3	3	3	3	1
Кросбраузерність	4	4	3	2	3
Якість документації	3	3	3	3	2
Економічна ефективність	3	3	3	3	1
Популярність у спільноті	2	1	3	3	1
MTTR	3.23	3.13	3.13	3.45	2.88

Продовження таблиці 4.3.2

Мова скриптів	2	2	3	1	3
Chaos Experiment Success Rate	2	2	3	2	2

Застосуємо принцип Парето (закон життєво важливих небагатьох [17]) для аналізу множини альтернатив.

Варіант a домінує варіанту b згідно з відношенням Парето (далі: a > b), якщо a хоча б за одним критерієм краще, ніж b, а по іншим критеріям не гірше, ніж b.

Створимо таблицю з нормованими значеннями (див. табл. 4.3.3)

Таблиця 4.3.3 - Нормовані значення

Критерій/інструмент	Playwright	Katalon Studio	Selenium	Cypress	TestComplete
Сумісність з ОС	1.00	1.00	1.00	1.00	0.33
Кросбраузерність	1.00	1.00	0.75	0.50	0.75
Якість документації	1.00	1.00	1.00	1.00	0.67
Економічна ефективність	1.00	1.00	1.00	1.00	0.33
Популярність у спільноті	0.67	0.33	1.00	1.00	0.33
Мова скриптів	0.67	0.67	1.00	0.33	1.00
MTTR	0.614	0.439	0.439	1	0
Chaos Experiment Success Rate	0.6	0.6	1	0.6	0.6

TestComplete має найгірші або слабкі показники в усіх напрямах і домінований усіма виключається.

Katalon поступається Playwright і Cypress у більшості критеріїв (особливо MTTR, скрипти, спільнота) і також виключається.

Selenium має кращу популярність, кращу мову скриптів, Chaos = 1, але програє Cypress по MTTR та має взаємне недомінування з Cypress.

Cypress має гіршу кросбраузерність і скрипти, але найкращий MTTR і популярність.

Playwright має найвищі значення в майже всіх параметрах, крім Chaos і популярності.

Визначимо вагові коефіцієнти методом простого ранжування, де 1 - найважливіший. Тут від вимагається провести ранжування, тобто впорядкувати досліджувані критерії, які характеризують об'єкт, за ступенем проявлення в них властивостей в порядку їх спадання [18]:

а) MTTR (ранг 1) - найважливіший критерій, оскільки безпосередньо впливає на ефективність тестування;

б) Chaos Experiment Success Rate (ранг 2) - визначає надійність впровадження та експлуатації інструменту, ґрунтується на ідеї, що «найкращий спосіб зрозуміти поведінку системи — це спостерігати її під навантаженням» [19];

в) сумісність з ОС (ранг 3) - забезпечує можливість використання в різних середовищах;

г) кросбраузерність (ранг 4) - визначає охоплення різних браузерів при тестуванні;

д) якість документації (ранг 5) - впливає на швидкість освоєння та використання інструменту;

е) економічна ефективність (ранг 6) - визначає ціну інструмента;

ж) популярність у спільноті (ранг 7) - забезпечує підтримку та розвиток інструменту;

и) мова скриптів (ранг 8) - найменш важливий критерій, оскільки всі інструменти підтримують основні мови програмування.

Використаємо формулу 4.3.1 для розрахунку очків для кожного критерію:

$$W_I = \frac{n - r_i + 1}{\sum(n - r_i + 1)} \qquad (4.3.1)$$

де n - кількість критеріїв,

r_i- ранг критерію,

$\sum(n - r_i + 1)$ - сума всіх $(n - r_i + 1)$.

Розрахуємо спочатку $n - r_i + 1$ для кожного критерію, де n = 8:

– MTTR: (8 - 1 + 1) = 8 очків;

– Chaos Experiment Success Rate: (8 - 2 + 1) = 7 очків;

– сумісність з ОС: (8 - 3 + 1) = 6 очків;

– кросбраузерність: (8 - 4 + 1) = 5 очків;

– якість документації: (8 - 5 + 1) = 4 очки;

– економічна ефективність: (8 - 6 + 1) = 3 очки;

– популярність у спільноті: (8 - 7 + 1) = 2 очки;

– мова скриптів: (8 - 8 + 1) = 1 очко.

Сума всіх очків: 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 36.

Розраховую вагові коефіцієнти за формулою Wi = очки/сума

– MTTR: 8/36 = 0.222;

– Chaos Experiment Success Rate: 7/36 = 0.194;

– сумісність з ОС: 6/36 = 0.167;

– кросбраузерність: 5/36 = 0.139;

– якість документації: 4/36 = 0.111;

– економічна ефективність: 3/36 = 0.083;

– популярність у спільноті: 2/36 = 0.056;

– мова скриптів: 1/36 = 0.028.

Розрахуємо адитивну лінійну згортку з ваговими коефіцієнтами (формула 4.3.2) бо критерії є незалежними один від одного. Ще можливо визначити відносну важливість кожного критерію і модель є простою для розуміння та інтерпретації результатів .

$$Z^* = max_{i=1,m} \sum_{j=1}^{n} \alpha j * \beta j * aij \qquad (4.3.2)$$

де αj - нормуючі множники (в нашому випадку всі значення вже нормовані),

βj - вагові коефіцієнти,

aij - значення j-го критерію для i-ї альтернативи.

Результати адитивної згортки:

Cypress: 0.834

Playwright: 0.8090

Отже за результатами адитивної лінійної згортки з урахуванням вагових коефіцієнтів, оптимальним рішенням серед Парето-оптимальних альтернатив є Cypress з показником 0.834.

ВИСНОВКИ

У результаті проведеного дослідження було виконано комплексний аналіз інструментів автоматизації тестування для веб-застосунків.

В рамках дослідження було розглянуто п'ять популярних інструментів автоматизації тестування: Selenium, Cypress, TestComplete, Katalon Studio та Playwright. Детально описано їхні основні характеристики, архітектурні особливості, переваги та недоліки, а також проаналізовано практичну придатність для конкретних сценаріїв тестування.

MTTR (Mean Time To Repair) — це не просто технічна метрика, а радше філософія виживання в умовах хаосу, помилок і продакшн-кризи. У сфері тестування вона відображає час від моменту виявлення збою до його повного усунення. Але за цими цифрами ховається щось більше: це про те, наскільки швидко команда може зрозуміти, що пішло не так, і повернути систему в робочий стан. Багато експертів стверджують, що ці метрики самі по собі не надто корисні, оскільки вони не зачіпають складніші питання — як саме вирішуються інциденти, що працює, а що ні, а також як, коли та чому проблеми загострюються або, навпаки, втрачають актуальність [20].

З технічної точки зору, MTTR — це швидкість локалізації "падаючих" тестів, а також час, необхідний на їхнє виправлення чи адаптацію. Але є й філософський рівень — наскільки інструмент полегшує життя інженеру. Чи допомагає він QA швидко зорієнтуватися в хаосі? Чи стає "психіатром" у моменти нервового зриву, коли CI червоніє, а продакшн не чекає?

Cypress демонструє бездоганний MTTR (1.00) і робить це не випадково. Його замкнена екосистема — це як моноліт, у якому все інтегровано: середовище виконання, репорти, відладка. Ця архітектура створює "штучну перевагу" — інженер не губиться в залежностях чи конфігураціях. Функції на кшталт автоматичних повторів (auto-retries), "подорожі в часі" (Time Travel) та інтегрованих логів працюють як швидка допомога для розгубленого інженера. Проте така ефективність

жертвує гнучкістю.

Розроблені тести на Cypress продемонстрували низку переваг, таких як автоматичне очікування елементів, зручна обробка асинхронних операцій та спливаючих вікон, а також зрозумілий і підтримуваний код.

Playwright (з MTTR 0.614) представляє протилежну крайність. Його відкрита архітектура, підтримка кількох мов і платформ створюють безмежні можливості — і водночас збільшують кількість точок потенційного збою. Щось може зламатися в конфігурації, щось — у залежностях, щось — у драйвері. Усе це ускладнює діагностику, змушує шукати помилку "вручну". Парадоксально, але чим більші можливості — тим гірший MTTR.

Отже, для автоматизації тестування веб-застосунків оптимальним вибором є Cypress, який забезпечує найкращий баланс між швидкістю виконання, простотою використання та ефективністю тестування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Verisign Domain Name Industry Brief [Електронний ресурс] - URL: https://blog.verisign.com/domain-names/verisign-q1-2023-the-domain-name-industry-brief/ (дата звернення: 02.04.2025).

2. Kozhevnikov, Andrii & Nataliya, Bilous. (2022). Research of methods for determining the accuracy of metrological measurements. Technology audit and production reserves. 3. 18-23. 10.15587/2706-5448.2022.259139 (дата звернення: 22.04.2025).

3. Shulika, Oleksiy & Safonov, Ivan & Ivanov, P.S. & Lysak, Volodymyr & Sukhoivanov, Igor & Lesna, Natalya. (2003). Comprehensive simulation of MQW semiconductor lasers by using laser CAD III. 80- 83. DOI: 10.1109/LFNM.2003.1246081 (дата звернення: 20.04.2025).

4. Client-Server Architecture - URL: https://darvishdarab.github.io/cs421_f20/docs/readings/client_server/ (дата звернення: 06.03.2025).

5. Dharmaadi, I & Athanasopoulos, Elias & Turkmen, Fatih. (2024). Fuzzing Frameworks for Server-side Web Applications: A Survey. 10 c.

6. Common Web Application Architectures. [Електронний ресурс] - URL: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures (дата звернення: 10.05.2025).

7. The Practical Test Pyramid / Martin Fowler. [Електронний ресурс] - URL: https://martinfowler.com/articles/practical-test-pyramid.html (дата звернення: 16.05.2025).

8. Graphical reporting for Selenium Webdriver and TestNG. [Електронний ресурс] - URL: https://stackoverflow.com/questions/34607135/graphical-reporting-for-selenium-webdriver-and-testng (дата звернення: 10.05.2025).

9. Selenium (software). [Електронний ресурс] - URL: https://en.wikipedia.org/wiki/Selenium_(software) (дата звернення: 10.04.2025).

10. Bhimanapati, Viharika & Goel, Punit & Jain, Ujjawal. (2024). Leveraging Selenium and Cypress for Comprehensive Web Application Testing. Journal of Quantum Science and Technology. 66-79 c.

11. What is Katalon Studio? An Introduction [Електронний ресурс] - URL: https://q-pros.com/what-is-katalon-studio-an-introduction/ (дата звернення: 26.08.2025).

12. An Introduction To TestComplete And Its Features! [Електронний ресурс] - URL: https://thinkpalm.com/blogs/an-introduction-to-testcomplete-and-its-features/ (дата звернення: 22.04.2025).

13. Dharmaadi, I & Athanasopoulos, Elias & Turkmen, Fatih. (2024). Fuzzing Frameworks for Server-side Web Applications: A Survey. 22-32 c.

14. Правило максимізації корисності і крива попиту [Електронний ресурс] - URL: https://stud.com.ua/13511/ekonomika/pravilo_maksimizatsiyi_korisnosti_kriva_popitu (дата звернення: 16.04.2025).

15. Flaky Test Management [Електронний ресурс] - URL: https://docs.datadoghq.com/tests/flaky_test_management/ (дата звернення: 16.04.2025).

16. Chaos Experiments [Електронний ресурс] - URL: https://developer.harness.io/docs/chaos-engineering/use-harness-ce/experiments/ (дата звернення: 26.04.2025).

17. Bunkley, Nick (2008). "Joseph Juran, 103, Pioneer in Quality Control, Dies". 10-12 c.

18. М.О Медиковський, О.Б. Шуневич (2011). Дослідження ефективності методів визначення вагових коефіцієнтів важливості. с. 176-177.

19. Types of Chaos Experiments (+ How To Run Them According to Pros) [Електронний ресурс] - URL: https://steadybit.com/blog/chaos-experiments/ (дата звернення: 20.05.2025).

20. MTBF, MTTR, MTTA, and MTTF [Електронний ресурс] - URL: https://www.atlassian.com/incident-management/kpis/common-metrics (дата звернення: 26.05.2025).

Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії

2. Kozhevnikov, A., Bilous, N. (2022). Research of methods for determining the accuracy of metrological measurements. Technology Audit and Production Reserves, 3 (2 (65)), 18–23. doi: http://doi.org/10.15587/2706-5448.2022.259139

3. Shulika, Oleksiy & Safonov, Ivan & Ivanov, P.S. & Lysak, Volodymyr & Sukhoivanov, Igor & Lesna, Natalya. (2003). Comprehensive simulation of MQW semiconductor lasers by using laser CAD III. 80-83. DOI: 10.1109/LFNM.2003.1246081.