МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп’ютерної інженерії

ЯНОВСЬКИЙ Олександр Михайлович

Програмний модуль автоматизації регресійного тестування із використанням засобів імітації дій користувачів / Software module for automation of the regression testing using means of simulating user actions

спеціальність: 123 – Комп’ютерна інженерія
освітньо-професійна програма – Комп’ютерна інженерія

Кваліфікаційна робота

Виконав: студент групи КІ-41
Яновський Олександр Михайлович

Науковий керівник
к.т.н. Гураль І.В.

ТЕРНОПІЛЬ - 2025

АНОТАЦІЯ

Кваліфікаційна робота на тему «Програмний модуль автоматизації регресійного тестування із використанням засобів імітації дій користувачів» зі спеціальності 123 «Комп'ютерна інженерія» освітнього ступеня «бакалавр» містить 70 сторінок пояснюючої записки, 16 рисунків, 2 таблиці та 5 додатків. Обсяг графічного матеріалу 2 аркуші формату А3.

Мета роботи полягає у розробці програмного модуля автоматизації регресійного тестування вебзастосунку з використанням інструментів емуляції дій користувачів.

Методи дослідження включають аналіз сучасних підходів до автоматизації тестування, порівняння інструментів емуляції дій користувачів (Selenium, Selenide, Playwright), розробку архітектури тестового модуля, реалізацію регресійних сценаріїв, інтеграцію з системами CI/CD (Jenkins, GitLab CI) та верифікацію роботи модуля на практичному прикладі.

У роботі розглянуто принципи реалізації регресійного тестування на основі повторюваних дій користувача, описано способи моделювання інтеракцій з вебінтерфейсом, обробки динамічних елементів, створення звітів про результати виконання тестів.

Результатом є створений програмний модуль, який дозволяє запускати тестові сценарії автоматично, фіксувати помилки, генерувати звіти та забезпечує ефективне тестове покриття критичних функцій вебзастосунку.

Ключові слова**:** РЕГРЕСІЙНЕ ТЕСТУВАННЯ, АВТОМАТИЗАЦІЯ, ЕМУЛЯЦІЯ ДІЙ КОРИСТУВАЧІВ, SELENIUM, SELENIDE, CI/CD, ВЕБІНТЕРФЕЙС.

ANNOTATION

The qualification paper titled «Software module for automation of the regression testing using means of simulating user actions», specialty 123 «Computer Engineering», Bachelor's degree, contains 70 pages of explanatory note, 16 figures, 2 tables and 5 appendices. The graphical material volume is 2 A3-format sheets.

The purpose of this work is to develop a software module for automating regression testing of a web application using user action emulation tools.

The research methods include analysis of modern approaches to automated testing, comparison of user interaction emulation tools (Selenium, Selenide, Playwright), architecture design of the test module, implementation of regression scenarios, integration with CI/CD systems (Jenkins, GitLab CI), and verification of the module’s functionality in a real testing environment.

The work examines the principles of implementing regression testing based on repetitive user actions, describes methods for simulating interactions with a web interface, handling dynamic elements, and generating test execution reports.

The result is a developed software module that enables automated execution of test scenarios, error detection, report generation, and ensures effective test coverage of critical web application functions.

Keywords**:** REGRESSION TESTING, AUTOMATION, USER ACTION SIMULATION, SELENIUM, SELENIDE, CI/CD, WEB INTERFACE.

ЗМІСТ

ВСТУП

Етап розвитку інформаційних технологій характеризується широким застосуванням вебзастосунків, хмарних платформ і інтерактивних сервісів, що потребують високої надійності та стабільності під час експлуатації. В умовах швидкого розвитку програмного забезпечення, частих оновлень і впровадження нових функціональностей важливим завданням стає забезпечення якості систем, що вже функціонують, зокрема через виявлення та запобігання регресійним помилкам. У цьому контексті особливої актуальності набуває регресійне тестування, яке дозволяє виявити порушення функціональності після внесення змін до коду.

З метою підвищення ефективності регресійного тестування у великих і складних проєктах активно впроваджуються методи автоматизації. Вони дозволяють не лише скоротити часові витрати на тестування, а й забезпечити повторюваність перевірок, зменшити вплив людського фактора та підвищити точність верифікації. Одним із найбільш поширених підходів до автоматизації є імітація дій користувача, що відтворює реальні сценарії взаємодії з інтерфейсом застосунку. Такий підхід дозволяє охопити критичні бізнес-процеси та забезпечити надійну регресійну перевірку після оновлення програмного продукту.

Об’єктом дослідження є процес автоматизованого регресійного тестування вебзастосунків.

Предмет дослідження – методи, інструменти та архітектурні рішення для побудови програмного модуля автоматизації регресійного тестування з використанням імітації дій користувача.

Метою даної роботи є розробка програмного модуля автоматизації регресійного тестування вебзастосунку з використанням інструментів емуляції дій

користувачів. Для досягнення поставленої мети у роботі необхідно виконати такі завдання:

– розглянути особливості регресійного тестування та його роль у життєвому циклі програмного забезпечення;

– провести аналіз сучасних підходів до автоматизації тестування та обґрунтувати вибір моделі, засобів і фреймворків;

– визначити переваги та недоліки підходу, заснованого на імітації дій користувача;

– розробити архітектуру програмного модуля та проєктну модель тестування;

– реалізувати програмний модуль у вигляді функціональних Java-класів;

– виконати автоматизоване регресійне тестування із прикладами сценаріїв;

– провести верифікацію роботи модуля, оцінити його ефективність та якість реалізації.

Результати дослідження та практичної реалізації були частково оприлюднені у вигляді тез на XII Всеукраїнській науково-практичній конференції молодих учених «Інформаційні технології – 2025» [1]. Світлокопії публікації наведено у додатку А.

Структура роботи відповідає поставленим завданням та складається з трьох розділів [2-6].

У першому розділі здійснено теоретичний аналіз регресійного тестування та підходів до автоматизації;

У другому розглянуто архітектурну модель розробленого модуля та описано логіку побудови тестових сценаріїв;

У третьому розділі реалізовано тестові класи, наведено приклади сценаріїв і здійснено верифікацію роботи програмного модуля.

1 ОСОБЛИВОСТІ РЕГРЕСІЙНОГО ТЕСТУВАННЯ

1.1 Призначення регресійного тестування у життєвому циклі програмного забезпечення

На сьогоднішній день стабільність і надійність функціонування програмного забезпечення (ПЗ) є не менш важливими критеріями, ніж наявність інноваційних функцій або зручність інтерфейсу. Будь-яке оновлення, покращення або виправлення помилок у коді може мати непередбачуваний вплив на вже реалізовану функціональність. Цей ефект, відомий як регресія, означає повторне виникнення помилок або появу нових збоїв у функціоналі, який раніше працював правильно [7]. Саме для запобігання таким ситуаціям в процесі розробки і підтримки програмного забезпечення активно використовується регресійне тестування, яке є одним з ключових етапів забезпечення якості ПЗ.

Регресійне тестування виконує надзвичайно важливу роль у підтримці функціональної цілісності програмного продукту. Його головна мета полягає у перевірці того, що нові зміни у коді – незалежно від їх масштабу – не впливають негативно на вже протестовані частини системи. Це дозволяє не лише мінімізувати ризики порушення функціональності, а й забезпечити сталість досвіду користувача. Більше того, застосування регресійного тестування дає можливість команді розробників упевнено впроваджувати нові функції, знаючи, що механізми перевірки оперативно виявлять порушення в інших частинах продукту.

У контексті життєвого циклу програмного забезпечення регресійне тестування займає важливе місце, що не обмежується лише етапом контролю якості [8]. На практиці воно виконується повторно після кожної суттєвої зміни або оновлення системи. Як показано на рисунку 1.1, регресійне тестування найчастіше асоціюється зі стадією тестування, проте фактично воно має зв’язок із усіма

попередніми етапами життєвого циклу [9], оскільки є засобом зворотного контролю стабільності системи після впровадження змін.

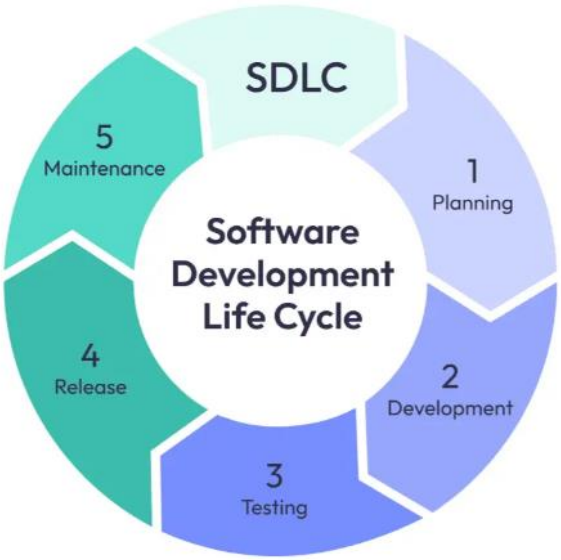

Рисунок 1.1 – Життєвий цикл програмного забезпечення

Цикл розробки програмного забезпечення включає класичні етапи: планування, аналіз вимог, проєктування, розробку, тестування та впровадження. Проте, регресійне тестування охоплює не лише момент перевірки перед випуском, але й постійно повторюється після кожного витка оновлень і змін (рисунок 1.2). Це забезпечує підтримку консистентності та передбачуваності функціональності програмного продукту [10].

Однією з особливостей регресійного тестування є його повторюваність. На відміну від первинного функціонального тестування, яке виконується один раз після розробки нового функціоналу, регресійні тести запускаються кожного разу, коли виникає необхідність перевірити стабільність системи. Це вимагає системного підходу до побудови тестових сценаріїв, їх структурування та зберігання в централізованих сховищах, які можуть бути легко оновлені у разі змін в логіці програми.

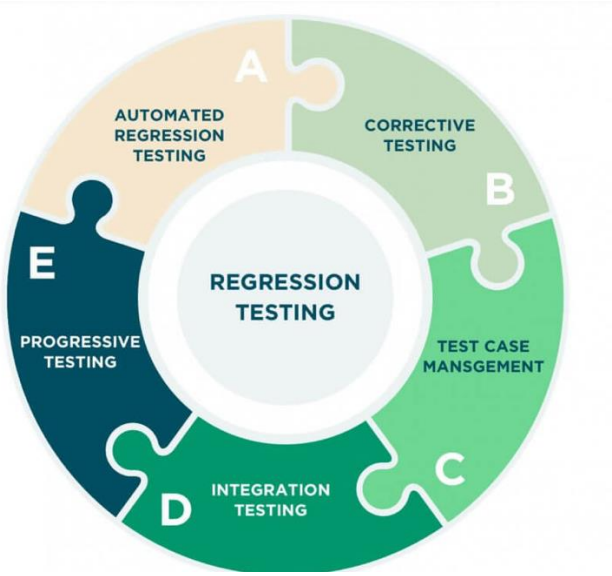

Рисунок 1.2 – Цикл регресійного тестування

Сучасні підходи до розробки програмного забезпечення, такі як Agile, Scrum або DevOps, ще більше підвищують значення регресійного тестування [11]. Часті ітерації розробки, що характерні для гнучких методологій, передбачають постійне оновлення функціоналу, що вимагає відповідного рівня перевірки. У цьому контексті без регресійного тестування команда не має змоги швидко й без ризику впроваджувати зміни, особливо коли мова йде про складні продукти з великою кількістю залежностей. Регресійні тести стають гарантом стабільності, дозволяючи за мінімальний час виявляти та локалізувати проблеми, що виникають внаслідок змін.

Ключовою особливістю регресійного тестування є його стратегічне значення у побудові довготривалого життєвого циклу ПЗ [12]. Підтримка великого проєкту передбачає тривалу експлуатацію, оновлення, рефакторинг, оптимізацію продуктивності, зміну архітектурних рішень, масштабування тощо. У кожному з цих випадків змінюється внутрішня структура або поведінка системи. Відсутність або неякісне проведення регресійного тестування може призвести до критичних помилок, особливо якщо мова йде про програмні продукти, що працюють з

фінансовими, медичними або правовими даними, де вартість помилки може бути надзвичайно високою.

Ще однією важливою характеристикою регресійного тестування є його здатність накопичувати історію перевірок [13]. Базуючись на архіві минулих запусків, команда розробників отримує можливість аналізувати тенденції появи помилок, визначати найбільш вразливі частини системи, які потребують додаткової уваги, та приймати рішення щодо їх рефакторингу або перепроектування. Крім того, систематичне проведення регресійного тестування формує культуру уважності до змін у коді, що позитивно впливає на загальну якість розробки.

Застосування регресійного тестування стає особливо важливим при роботі з мікросервісною архітектурою, де незалежні компоненти взаємодіють через API [14]. Зміни в одному мікросервісі можуть викликати небажані ефекти в іншому, навіть якщо прямої залежності не передбачено. Регресійні тести дозволяють виявити такі побічні ефекти ще до того, як продукт буде розгорнутий у продуктивному середовищі. Аналогічна ситуація спостерігається у розподілених системах, що взаємодіють з великою кількістю зовнішніх сервісів, баз даних, черг повідомлень та інших компонентів.

Не менш важливим є використання регресійного тестування в мобільній та веб-розробці, де постійні оновлення, різноманітність платформ, пристроїв, браузерів та версій ОС значно ускладнюють контроль якості [15]. Тут регресійне тестування дозволяє зберігати стабільність продукту незалежно від змін у зовнішньому середовищі або конфігурації пристрою.

Слід також відзначити, що ефективність регресійного тестування значною мірою залежить від ступеня його автоматизації [16]. Ручне виконання регресійних тестів є надзвичайно трудомістким і ресурсоємним процесом, що не підходить для великомасштабних проектів або продуктів, які оновлюються з великою частотою. Тому сьогодні практично неможливо уявити регресійне тестування без

автоматизованих інструментів. Автоматизація дає змогу виконувати сотні й тисячі перевірок за лічені хвилини, забезпечуючи швидкий зворотний зв'язок і значно знижуючи витрати [17].

Наявність автоматизованого регресійного набору тестів є необхідною умовою для впровадження безперервної інтеграції та безперервного розгортання (CI/CD) [18]. Під час кожного коміту в систему контролю версій автоматично запускається процес збірки та виконання регресійних тестів, що дозволяє миттєво отримувати інформацію про потенційні збої. Таким чином, регресійне тестування стає не просто перевіркою функціональності, а інструментом управління якістю на всіх етапах життєвого циклу продукту.

На завершення варто підкреслити, що правильне впровадження регресійного тестування у процес розробки ПЗ сприяє зниженню витрат, скороченню часу виходу на ринок, підвищенню задоволеності користувачів і конкурентоспроможності продукту. Його значення не обмежується лише технічною площиною – регресійне тестування формує довіру до продукту як з боку команди, так і з боку кінцевих користувачів, а в умовах високої конкуренції на ринку програмного забезпечення – це фактор, який часто визначає успіх або провал.

Таким чином, регресійне тестування є не просто частиною процесу забезпечення якості, а фундаментальним інструментом, що дозволяє гарантувати стабільність, цілісність та передбачуваність поведінки програмного забезпечення у довготривалій перспективі його використання. Його інтеграція у життєвий цикл програмного продукту забезпечує збереження функціональної цілісності на кожному етапі розробки, підтримки та розвитку, і є невід’ємною складовою сучасного інженерного підходу до створення програмного забезпечення.

1.2. Аналіз сучасних підходів до автоматизації регресійного тестування

У зв’язку зі зростанням складності програмних продуктів, а також пришвидшенням темпів їх розробки та оновлення, особливої актуальності набуває автоматизація процесів регресійного тестування [19]. У сучасному IT-середовищі потреба в оперативному, точному й багаторазовому запуску перевірок після внесення змін до коду стала не просто побажанням, а вимогою. Саме автоматизація дозволяє забезпечити сталість якості, підтримувати високу швидкість доставки функціоналу і при цьому контролювати витрати, пов’язані з ручним тестуванням. У цьому підрозділі розглянуто основні концепції, принципи та інструменти, що використовуються для автоматизації регресійного тестування на сучасному етапі розвитку програмної інженерії.

Автоматизоване регресійне тестування передбачає створення спеціальних сценаріїв, які виконуються за допомогою програмних засобів без втручання людини. Основна ідея полягає в тому, щоб один раз створивши автоматизований тест, мати змогу запускати його будь-яку кількість разів після кожної зміни у програмному забезпеченні. Такий підхід забезпечує не лише значне скорочення часу, потрібного на перевірку, але й підвищення надійності результатів – адже людський фактор повністю усувається з процесу виконання тестів. Крім того, автоматизація дозволяє інтегрувати тестування в процес безперервної розробки, що є необхідною умовою для Agile- і DevOps-команд [20].

Сучасні підходи до автоматизації регресійного тестування тісно пов’язані з поняттям тестової стратегії проєкту. Розробники і тестувальники вже на початкових етапах визначають, які саме частини системи потребують автоматизації, які інструменти будуть використані, як тестові сценарії будуть оновлюватися при зміні функціоналу, і як забезпечити масштабованість та стабільність тестового набору. У цьому контексті важливо розуміти, що

автоматизація – це не одноразова дія, а постійний ітеративний процес, який розвивається разом із програмним продуктом.

Одним із найпоширеніших підходів до автоматизації регресійного тестування є модульне тестування, яке фокусується на перевірці окремих частин програмного коду [21]. Такі тести пишуться зазвичай на мовах програмування, що використовуються у проєкті, та інтегруються у збірку системи через засоби безперервної інтеграції. Модульне тестування добре масштабується, швидко виконується і дозволяє локалізувати помилки на ранніх етапах. Проте воно не завжди здатне охопити помилки, що виникають на рівні взаємодії між модулями, або ті, що пов’язані з користувацькою взаємодією з інтерфейсом.

Ще одним важливим напрямом є автоматизація UI-тестів, яка охоплює поведінку системи з точки зору кінцевого користувача [22]. Сценарії такого тестування моделюють дії людини – натискання кнопок, заповнення форм, переміщення по меню, скролінг сторінки тощо. Для реалізації таких тестів широко використовуються інструменти, здатні взаємодіяти з графічним інтерфейсом – серед яких особливо популярними є Selenium, Selenide, Cypress, Playwright. Ці інструменти дозволяють створювати гнучкі сценарії, які можуть відтворювати повний цикл взаємодії користувача із системою. Такий підхід є незамінним у випадках, коли потрібно перевірити наскільки зручно, швидко і безпомилково працює інтерфейс при змінах у коді.

Важливим фактором ефективності автоматизованого регресійного тестування є інтеграція з системами CI/CD (Continuous Integration / Continuous Delivery) [23]. У таких середовищах кожен коміт у репозиторій коду автоматично ініціює процес збірки, запуску тестів і, за відсутності помилок, – розгортання оновленої версії програмного продукту. У цьому контексті автоматизовані регресійні тести відіграють роль «захисного бар’єру», який не дозволяє дефектному коду потрапити до користувача. Таким чином, регресійне тестування

стає невід'ємною частиною загального процесу DevOps – як на рівні контролю якості, так і як інструмент гарантування надійності змін.

Поряд із інструментами, що працюють безпосередньо з кодом або інтерфейсом, все більшої популярності набувають так звані Record & Playback системи, які дозволяють створювати автоматизовані тести шляхом запису дій користувача в браузері або додатку [24]. До таких інструментів належать, зокрема, TestProject, Katalon Recorder, Ranorex Studio, а також розширення для браузера, як-от Selenium IDE. Вони особливо корисні для створення прототипів тестів або швидкої автоматизації простих сценаріїв, однак можуть поступатися в гнучкості та масштабованості порівняно з «кодовими» фреймворками.

Не менш важливим напрямком розвитку сучасної автоматизації є застосування хмарних платформ для регресійного тестування [25]. Такі сервіси, як BrowserStack, Sauce Labs, LambdaTest надають можливість одночасного запуску тестів на великій кількості конфігурацій операційних систем, браузерів та пристроїв. Це особливо актуально у випадку кросбраузерного або мобільного тестування, де необхідно забезпечити стабільність роботи продукту в різних умовах.

Крім інструментів, важливу роль у забезпеченні якості автоматизованого регресійного тестування відіграють підходи до організації коду тестів, підтримки їх актуальності, контролю залежностей, параметризації сценаріїв тощо [26]. Використання шаблонів Page Object Model (POM), Data-Driven Testing (DDT), Behavior-Driven Development (BDD) дозволяє створювати тестовий код, який є не лише функціональним, але й зрозумілим, масштабованим і зручним у підтримці. Такі архітектурні підходи дозволяють команді не тільки швидко створювати нові сценарії, але й легко адаптувати існуючі до змін у системі.

Суттєво підвищити ефективність автоматизації допомагає також застосування метрик, що дозволяють об'єктивно оцінити якість регресійного набору тестів [27]. Серед таких метрик можна назвати покриття коду (code

coverage), кількість знайдених дефектів, середній час виконання тестів, стабільність тестів (відсоток флейків) тощо. Регулярний аналіз таких показників дає змогу виявити слабкі місця в автоматизованій перевірці й вчасно вжити заходів щодо її покращення.

Усі розглянуті підходи та інструменти мають свої переваги та обмеження, які необхідно враховувати залежно від цілей і масштабу конкретного проєкту. З позиції даної дипломної роботи, яка присвячена розробці програмного модуля саме для автоматизації регресійного тестування з використанням емуляції дій користувачів, найбільш доцільним підходом є використання інструментів, що забезпечують повну симуляцію взаємодії з інтерфейсом програми. Це дає змогу досягти максимальної відповідності реальній поведінці користувача та дозволяє виявляти помилки, які інші рівні тестування (модульні, API, інтеграційні) можуть не охоплювати.

З цією метою найбільш релевантними інструментами є Selenium, Selenide, Playwright, а також сучасні середовища типу TestCafe чи Cypress, які дозволяють автоматизувати сценарії користувацької взаємодії навіть у динамічних односторінкових застосунках. Саме цей клас інструментів найкраще відповідає вимогам до реалізації програмного модуля, який буде здатний виконувати регресійні перевірки через інтерфейс, симулюючи поведінку кінцевого користувача максимально наближено до реального сценарію.

Однак ефективність такого підходу неможливо оцінити без розуміння його обмежень. Як і будь-який інструмент, що працює на рівні UI, технології імітації дій користувача мають як суттєві переваги, так і низку викликів, пов’язаних з продуктивністю, стабільністю виконання тестів та складністю підтримки сценаріїв.

1.3. Переваги та недоліки підходу імітації дій користувачів

Імітація дій користувача – це один із найефективніших підходів до тестування поведінки програмного забезпечення, особливо коли мова йде про регресійне тестування інтерфейсів[28]. Такий підхід дозволяє виявити дефекти, які не вдається виявити на рівні логіки або API, оскільки тестування виконується в умовах, максимально наближених до реального використання продукту кінцевим користувачем. У випадку вебдодатків це передбачає взаємодію з елементами DOM, заповнення форм, натискання кнопок, перемикання між вкладками тощо. Технології імітації користувацьких дій надають змогу автоматизувати такі сценарії і значно підвищити якість кінцевого продукту.

Існує кілька класів інструментів, які дозволяють реалізовувати імітацію дій користувача: це бібліотеки для програмної взаємодії з браузером, такі як Selenium, Selenide, Puppeteer, Playwright, а також системи Record & Playback – наприклад, Katalon Recorder чи TestProject, які дають змогу створювати тести без програмування [29].

Кожен з цих інструментів має свої переваги та обмеження. Для зручного порівняння нижче у таблиці 1.1 представлено характеристики, що оцінюються ключовими аспектами використання зазначених технологій з точки зору автоматизації регресійного тестування:

Як видно з таблиці, Selenide є одним з найбільш зручних і стабільних інструментів для реалізації автоматизації регресійного тестування у Java-проєктах. На відміну від «чистого» Selenium, він забезпечує більш високий рівень абстракції**,** має вбудовану обробку очікувань**,** зручну роботу з елементами DOM**,** а також надає детальну звітність без потреби додаткових конфігурацій. Завдяки використанню Selenide, розробник зосереджується на логіці тестів, а не на обробці технічних нюансів, таких як налаштування драйверів, тайм-аутів чи стабілізація сценаріїв.

Таблиця 1.1 – Порівняльна характеристика інструментів, що реалізовують імітацію дій користувача

Критерій / Інструмент	Selenium	Selenide	Puppeteer	Playwright	Katalon / TestProject
Мови програмування	Java, Python, C#, JS та ін.	Java	JavaScript / TypeScript	JS, TS, Python, C#	Без коду або Java / Groovy
Підтримка браузерів	Chrome, Firefox, Edge, Safari, IE	Chrome, Firefox	Лише Chromium	Chromium, Firefox, WebKit	Через WebDriver
Мобільне тестування	(через Appium)	(через інтеграцію)		(обмежено)	(частково)
Гнучкість та розширюваність	Висока	Висока	Обмежена	Висока	Обмежена
Інтеграція з CI/CD	Добра	Добра	Добра	Добра	Залежить від платформи
Стабільність тестів	Залежить від реалізації	Краща стабільність	Добра	Висока	Середня
Порог входу	Середній / високий	Нижчий	Низький	Низький	Дуже низький
Швидкість виконання	Середня	Вища	Висока	Висока	Середня
Підтримка спільноти	Дуже велика	Велика	Активна	Активна	Обмежена
Простота налаштування	Потребує конфігурації	Менше налаштувань	Просте	Просте	Дуже просте
Логування / звітність	Потребує дод. налаштувань	Вбудована підтримка	Обмежена	Вбудована підтримка	Візуальні звіти

Як видно з таблиці, Selenide є одним з найбільш зручних і стабільних інструментів для реалізації автоматизації регресійного тестування у Java-проєктах. На відміну від “чистого” Selenium, він забезпечує більш високий рівень абстракції, має вбудовану обробку очікувань, зручну роботу з елементами DOM, а також надає детальну звітність без потреби додаткових конфігурацій. Завдяки використанню Selenide, розробник зосереджується на логіці тестів, а не на обробці технічних нюансів, таких як налаштування драйверів, тайм-аутів чи стабілізація сценаріїв.

Selenium, хоча і є базовим рівнем для багатьох інструментів, зокрема й Selenide, вимагає значно більше ручної конфігурації і схильний до помилок через

потребу уявно «керувати» браузером на низькому рівні. Для великого автоматизованого проєкту, де важлива не лише функціональність, а й зручність підтримки тестів, це може стати обмеженням.

У контексті даної дипломної роботи, що реалізується на мові Java і передбачає створення стабільного, повторюваного та зручного у використанні модуля, саме Selenide є найкращим вибором. Він дозволяє легко писати тести, що імітують поведінку користувача у браузері, з мінімальними витратами на технічні налаштування, та забезпечує хорошу інтеграцію з JUnit/TestNG, Allure, CI/CD-середовищами.

Таким чином, платформа Selenide обирається як основний інструмент реалізації програмного модуля автоматизованого регресійного тестування, орієнтованого на симуляцію дій користувача. Водночас для повного розуміння ефективності цього підходу необхідно дослідити не лише технічні переваги, а й потенційні виклики, які можуть виникати в процесі роботи. У наступному розділі буде детальніше розглянуто сутність підходу симуляції дій користувача, а також його переваги та недоліки, які мають бути враховані під час реалізації автоматизації тестування у рамках цієї роботи.

2 ПРОЕКТУВАННЯ ТА АРХІТЕКТУРА ПРОГРАМНОГО МОДУЛЯ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ

2.1. Обґрунтування вибору засобів реалізації програмного модуля

Автоматизація регресійного тестування на сучасному етапі розвитку інформаційних технологій є важливою складовою процесу забезпечення якості програмного забезпечення. З огляду на постійне зростання обсягів і складності функціональності програмних систем, особливого значення набуває розробка універсальних та гнучких програмних модулів, здатних виконувати повторювані перевірки після внесення змін до коду [30]. Такий підхід дозволяє своєчасно виявляти помилки, які можуть виникнути внаслідок неочікуваних побічних ефектів, спричинених оновленнями, та, відповідно, запобігати регресіям у роботі системи.

У межах даного дослідження передбачено створення програмного модуля автоматизації регресійного тестування із використанням засобів імітації дій користувачів, що дозволяє здійснювати перевірку поведінки вебінтерфейсу програмного продукту в умовах, максимально наближених до реальної взаємодії кінцевого користувача із системою. Основною метою є підвищення ефективності процесу тестування за рахунок автоматизації типових сценаріїв, що повторюються після кожної ітерації розробки або випуску оновлень.

Таким чином необхідно реалізувати програмний модуль, що забезпечить:

– автоматизоване виконання тестових сценаріїв у веббраузері з імітацією дій користувача;

– підтримку багаторазового повторного запуску тестів після змін у програмному коді;

– інтеграцію з CI/CD пайплайнами для забезпечення безперервної перевірки;

– виведення детальної звітності про результати тестування;

– легкість адаптації, розширення та повторного використання.

Узагальнена схема запропонованого підходу подана в додатку В.

З урахуванням специфіки задачі, яка передбачає реалізацію модулю у середовищі Java, а також потреби у стабільності, простоті налаштування та підтримці сучасних вебтехнологій, було обрано відповідний набір технологічних засобів реалізації.

У якості основної бібліотеки для реалізації симуляції дій користувача обрано Selenide, що є високорівневим обгортанням над фреймворком Selenium WebDriver. Такий вибір обумовлений низкою переваг: автоматичне управління очікуваннями елементів, зручний синтаксис, мінімальна потреба в конфігурації, стабільна обробка помилок та інтеграція з популярними фреймворками для тестування.

Додатково для організації структури тестів і їх запуску буде використано JUnit 5 – сучасний фреймворк модульного тестування, який забезпечує гнучку систему анотацій, параметризації, а також підтримує розширення. Для генерації звітів про результати виконання тестів передбачено інтеграцію з Allure Report, що дозволяє створювати візуалізовані багаторівневі звіти із зазначенням статусів, часу виконання та трасування помилок.

Управління проєктом, підключення залежностей, конфігурація середовища тестування та виконання тестів здійснюватиметься за допомогою системи автоматизації збірки Maven. Для зберігання коду, контролю версій і реалізації безперервної інтеграції буде використано Git у поєднанні з CI/CD-середовищем, наприклад GitLab CI.

Архітектура обраного інструментарію зображена на рисунку 2.1.

Побудова рішення базується на багаторівневому технологічному стеку, який представлено у вигляді узагальненої моделі на рисунку 2.2. Такий підхід дозволяє чітко виокремити логіку виконання тестів, інфраструктурні компоненти та засоби звітності.

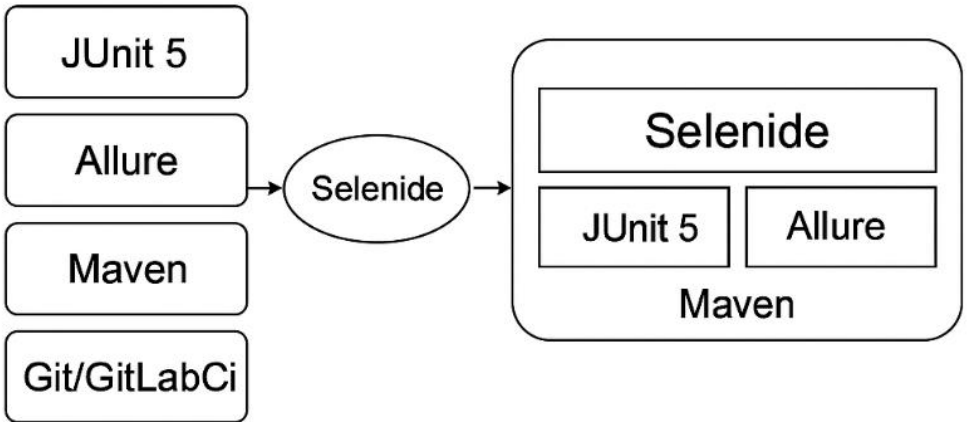

Рисунок 2.1 – Архітектура використаних інструментів у модулі

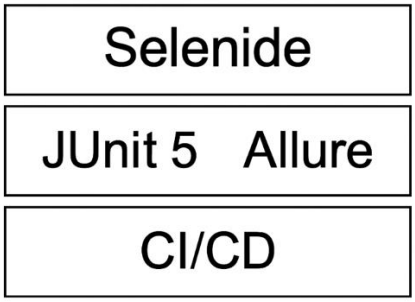

Рисунок 2.2 – Технологічний стек реалізації програмного модуля

Таким чином, на основі аналізу вимог до програмного модуля, оцінки можливостей сучасних інструментів та з урахуванням обмежень середовища розробки, було сформовано обґрунтований вибір стеку технологій, що забезпечить ефективну реалізацію поставленої задачі.

2.2. Архітектурна модель програмного модуля автоматизації регресійного тестування

Побудова архітектури програмного модуля автоматизації регресійного тестування є важливим етапом, що визначає подальшу ефективність його

функціонування, масштабованість, гнучкість та можливість інтеграції в існуючі процеси розробки програмного забезпечення. Створення надійної архітектурної основи дає змогу реалізувати тестовий модуль таким чином, щоб він був легко супроводжуваним, розширюваним і придатним для багаторазового використання. Враховуючи особливості предметної галузі та вибраний підхід до реалізації (імітація дій користувача через браузер з використанням фреймворку Selenide), було розроблено багаторівневу архітектурну модель, яка охоплює ключові компоненти та відображає логіку їх взаємодії.

На рисунку 2.3 подано узагальнену архітектурну модель програмного модуля, яка поділена на чотири логічні рівні: рівень взаємодії з користувачем (UI level), логічний рівень (Core logic), інфраструктурний рівень (Infrastructure level) та рівень CI/CD.

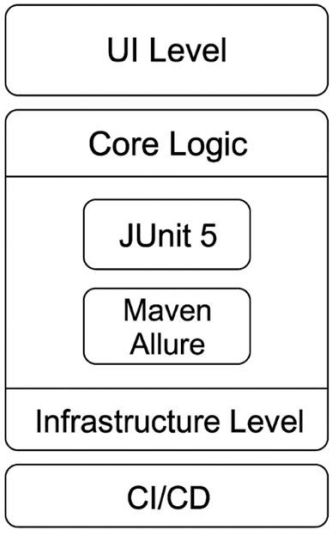

Рисунок 2.3 – Архітектурна модель програмного модуля

Розглянемо рівень взаємодії з користувачем (UI level). Цей рівень відповідає за опис сценаріїв тестування в термінах поведінки користувача. Застосовується патерн Page Object, що дозволяє структуровано організувати код із виділенням окремих класів для кожної вебсторінки або компонента інтерфейсу. Такий підхід

сприяє повторному використанню коду, зниженню дублювання і полегшенню супроводу тестів. У рамках цього рівня реалізуються класи, що описують елементи сторінок, методи взаємодії з ними (натискання кнопок, введення тексту, вибір значень тощо), а також логіка переходів між сторінками.

У центрі логічного рівня (Core logic) перебуває безпосереднє визначення тестових сценаріїв і перевірок. Саме тут виконується зв'язування кроків користувача в логічну послідовність та формулювання очікуваних результатів (асертацій). У даному модулі реалізація тестів здійснюється з використанням фреймворку JUnit 5, що забезпечує структурування тестів, зручну систему анотацій та параметризацію. Важливо зазначити, що Selenide повністю інтегрується з JUnit, що спрощує реалізацію тестів, дозволяючи уникнути явної обробки очікувань, об'єктів WebDriver та інших технічних аспектів.

На цьому рівні відбувається активна взаємодія з Page Object, реалізованими на попередньому рівні. Зокрема, в рамках одного тесту можуть використовуватись декілька сторінок, а тестова логіка полягає у відтворенні реального сценарію дій користувача в браузері з верифікацією очікуваних результатів на кожному кроці.

Інфраструктурний рівень (Infrastructure level) забезпечує підключення зовнішніх інструментів, що необхідні для налаштування середовища тестування, управління залежностями, запуску тестів та формування звітності. У рамках цієї реалізації було використано систему керування проєктом Maven, яка дозволяє ефективно керувати залежностями, конфігураціями середовища, профілями запуску тощо.

Для формування звітів про результати тестування застосовується Allure Report – візуальний фреймворк, що забезпечує генерацію інформативних звітів у форматі HTML з відображенням структури тестового набору, часу виконання, статусів кожного тесту та трасуванням у разі помилок. Завдяки повній інтеграції з JUnit і Maven, Allure дозволяє реалізувати безперервний збір інформації про стан тестування без додаткового втручання користувача.

На найнижчому рівні архітектури розташований процес безперервної інтеграції та доставки (CI/CD), який забезпечує автоматичний запуск тестів після кожного оновлення кодової бази. В реалізації дипломного проєкту використовується GitLab CI, що дозволяє створити pipeline із кількох етапів (наприклад: build, test, report), де тестовий модуль відіграє роль критичного компонента, відповідального за контроль якості змін.

CI/CD-рівень забезпечує не лише регулярне тестування, але й підключення результатів до загального процесу контролю релізів, забезпечуючи своєчасне виявлення помилок і блокування релізу в разі їх виявлення.

На рисунку 2.4 подано послідовність взаємодії компонентів модулю в рамках одного запуску тесту. Діаграма послідовності демонструє, як тестовий клас викликає методи Selenide, які відкривають сторінки, взаємодіють з елементами інтерфейсу та здійснюють перевірку результатів. Цей процес є частиною пайплайну, що виконується GitLab CI.

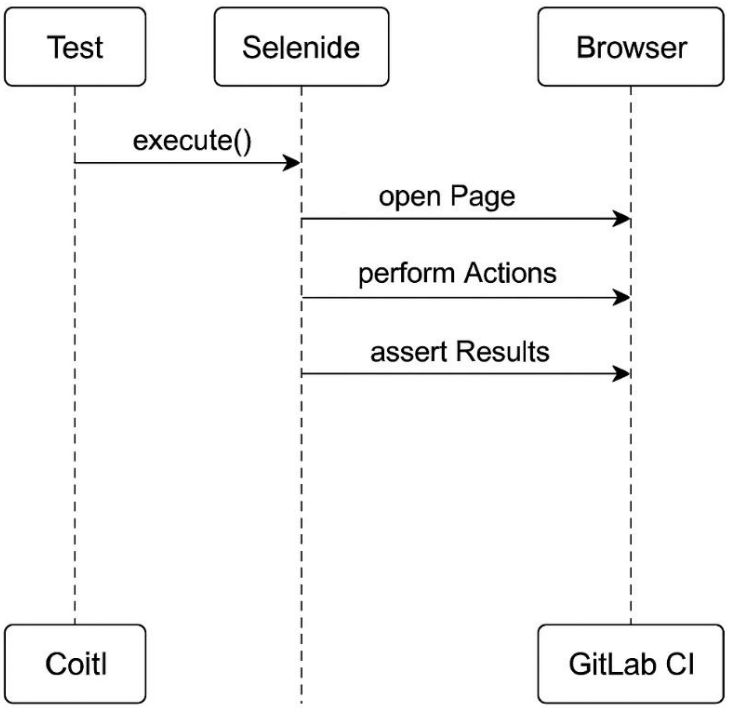

Рисунок 2.4 – Послідовність взаємодії компонентів під час виконання тесту

Рисунок 2.5 візуалізує внутрішню структуру модулю на основі патерну Page Object. Структурна UML-діаграма показує, як клас Page успадковується Page Object, який містить методи взаємодії з Page Element. Клас Test викликає методи Page Object, ініціюючи сценарій користувача.

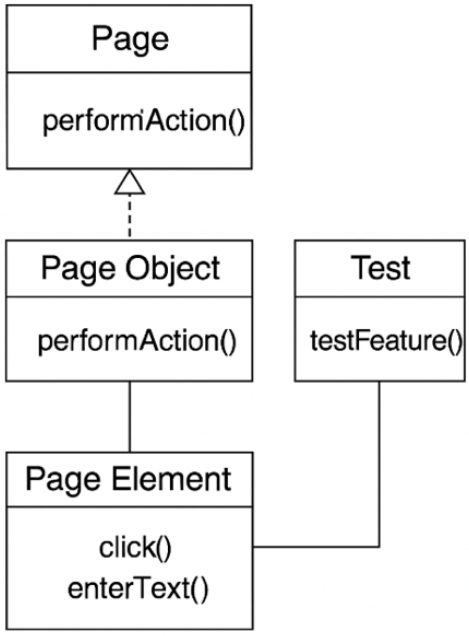

Рисунок 2.5 – Структура модуля з використанням патерну Page Object

Загальна архітектура програмного модуля відповідає принципам модульності, повторного використання та інкапсуляції. Кожен рівень чітко відокремлений від інших, що спрощує модифікацію і розширення окремих частин системи без ризику порушення її цілісності. Такий підхід дозволяє забезпечити високу надійність модуля, його стабільність при частих оновленнях та сумісність із сучасними інструментами безперервної інтеграції.

2.3. Проектування сценаріїв взаємодії з інтерфейсом користувача

Проєктування сценаріїв взаємодії з інтерфейсом користувача є ключовим етапом побудови ефективної системи автоматизованого регресійного тестування. У контексті тестування вебзастосунку Google Cloud Pricing Calculator особливе значення має деталізоване моделювання дій користувача, що дозволяє забезпечити перевірку як основного функціоналу, так і граничних ситуацій. Основне завдання на цьому етапі – чітко визначити, які саме елементи інтерфейсу потребують перевірки, які дії і в якій послідовності мають бути змодельовані, а також які очікувані результати свідчитимуть про правильність роботи системи.

Для реалізації сценаріїв взаємодії використовується шаблон Page Object, який дозволяє відокремити логіку представлення інтерфейсу від логіки самих тестів. Кожна функціональна сторінка або підкомпонент інтерфейсу реалізується у вигляді окремого Java-класу, що містить локатори необхідних елементів та методи взаємодії з ними. Тестові класи, у свою чергу, оперують тільки методами відповідних Page-класів, що дозволяє підтримувати стабільність тестів навіть у разі змін у DOM-структурі.

Розглянемо два приклади Java-класів для взаємодії зі сторінками Google Cloud Calculator: GoogleCloudPage та GoogleCloudCalculatorPage.

Основний клас – GoogleCloudPage – відповідає за відкриття головної сторінки Google Cloud і пошук калькулятора вартості (рисунок 2.6). Він містить метод openPricingCalculator(), який виконує послідовність дій для відкриття необхідного розділу.

```java
public class GoogleCloudPage {
    private SelenideElement searchButton = $("button[aria-label='Open search']
    private SelenideElement searchField = $("input[aria-label='Search']");
    private SelenideElement firstResult = $(".gs-title");

    public void openPricingCalculator() {
        open("https://cloud.google.com/");
        searchButton.click();
        searchField.setValue("Google Cloud Pricing Calculator").pressEnter();
        firstResult.click();
    }
}
```

Рисунок 2.6 – Java-клас GoogleCloudPage для переходу до калькулятора вартості Google Cloud

На основі цих Page Object-класу реалізуються тестові класи (рисунок 2.7), які викликають методи взаємодії із UI, не залежачи від реалізації самої сторінки. Це дозволяє дотримуватись принципів інкапсуляції, модульності й розділення обов'язків.

```java
public class GoogleCloudTest {
    GoogleCloudPage googleCloudPage = new GoogleCloudPage();
    GoogleCloudCalculatorPage calculatorPage = new GoogleCloudCalculatorPage()

    @Test
    public void estimateCostTest() {
        googleCloudPage.openPricingCalculator();
        calculatorPage.fillRequiredFields();
        calculatorPage.addToEstimate();
    }
}
```

Рисунок 2.7 – Java-клас GoogleCloudTest для моделювання користувацького сценарію оцінки вартості

Нарешті, для перевірки взаємодії між класами GoogleCloudPage та GoogleCloudCalculatorPage створюється тестовий клас GoogleCloudTest, який демонструє типову послідовність дій користувача: відкриття сайту, пошук калькулятора, заповнення конфігурації та додавання розрахунку до кошика.

Для узагальнення структури зв’язків між компонентами застосовано UML-діаграму, яка подана у додатку Г. Діаграма демонструє взаємозв’язок між тестовими класами й класами сторінок – кожен тест викликає один або кілька Page-класів, кожен з яких інкапсулює частину UI-логіки.

Проєктування сценаріїв на основі Page Object робить автоматизовані тести стійкими до змін у зовнішньому представленні інтерфейсу, підвищує читабельність та надійність тестового коду, що є необхідним для довготривалої підтримки регресійного тестування у рамках великих програмних систем.

3 РЕАЛІЗАЦІЯ, ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО МОДУЛЯ

3.1 Реалізація програмного модуля автоматизації регресійного тестування і структура коду

Етап реалізації програмного модуля є ключовим для досягнення цілей автоматизації регресійного тестування, зокрема з огляду на практичну реалізацію обраної архітектури, формалізованих сценаріїв і технічних рішень. У даній роботі реалізація виконана з використанням мови Java, фреймворку JUnit 5, бібліотеки Selenide, системи керування залежностями Maven і генератора звітів Allure. Програмний модуль орієнтовано на стабільність, масштабованість і простоту підтримки.

Загальна структура проєкту відповідає типовій організації автоматизованих тестів у середовищі Maven і наведена на рисунку 3.1.

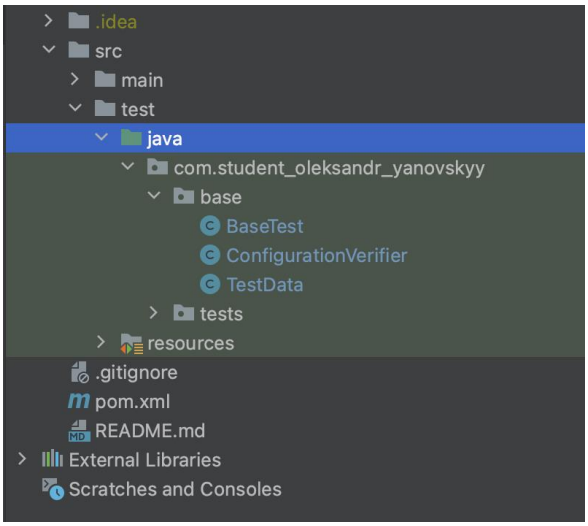

Рисунок 3.1 – Структура проєкту автоматизованих тестів

Вона включає основні каталоги .allure, build, src/test/java для зберігання тестів, src/test/resources для ресурсів, target для результатів збірки та файл pom.xml

з описом залежностей і плагінів. Окрім того, конфігураційна директорія lama-at-config використовується для параметризації середовища тестування, наприклад, зберігання файлу налаштувань логіну, шляхи до середовищ або URL.

Файл pom.xml містить необхідні залежності для реалізації модуля, включаючи бібліотеки для роботи з браузером, написання тестів, генерації звітів та інтеграції зі збіркою. Серед основних бібліотек – com.codeborne:selenide, org.junit.jupiter:junit-jupiter, io.qameta.allure:allure-junit5. Також у файлі зазначено плагіни, які відповідають за збірку, запуск тестів і генерацію звітів. У додатку Б подано фрагмент конфігураційного файлу pom.xml, що ілюструє основні секції залежностей і плагінів.

Файл pom.xml є критичним для коректного виконання тестів, оскільки саме він формує середовище виконання: обирає версії бібліотек, плагінів, формує репозиторії, що використовуються для отримання залежностей, і задає конфігурації тестового запуску.

Для централізованого керування конфігурацією браузера і життєвим циклом тестів створено базовий клас BaseTest. У ньому реалізовано ініціалізацію драйвера браузера перед виконанням кожного тесту та його закриття після завершення. Завдяки цьому інші тестові класи можуть наслідувати цей клас і уникати дублювання коду. Це відповідає принципам успадкування, повторного використання коду та забезпечує контроль над середовищем запуску.

Клас BaseTest забезпечує єдність конфігурацій у всіх тестах. Його використання дозволяє зосередитися на логіці перевірки, оскільки підготовка і очищення середовища виконується автоматично. Код класу представлено нижче на рисунку 3.2.

Така структура реалізації забезпечує модульність і незалежність кожного тесту, а також полегшує інтеграцію модуля в пайплайн безперервної інтеграції. Саме завдяки цьому модуль може бути безпосередньо включений у CI/CD

середовище (наприклад, GitLab CI) і виконувати перевірки автоматично після кожного оновлення коду.

```
public class BaseTest

    protected WebDriver driver;
    protected BasePage basePage;
    protected GoogleCloudCalculatorPage
    protected GoogleCloudPage
    protected CostEstimateSummaryPage

    @BeforeEach
    public void setUp() {
        driver = WebDriverManager.getDriver
                 (BrowserType.CHROME);
        basePage = new BasePage(driver);
        googleCloudCalculator = new
        GoogleCloudCalculatorPage(dL;
        googleCloudPage = new GoogleCloudPage
        costEstimateSummaryPage = new
        CostEstimateSummaryPage driverr);
    }

    @AfterEach
    public void tearDown() {
        if (driver != null) {
            WebDriverManager.closeDriver();
    }
}
```

Рисунок 3.2 – Клас BaseTest

У загальному випадку, реалізація модулю враховує всі принципи чистого коду (Clean Code), об'єктно-орієнтованого підходу (ООР), інкапсуляції та автоматизованого управління середовищем. Вона легко масштабується, дозволяє додавати нові тести, без ризику порушити існуючі сценарії, і є зручною для колективного розроблення та підтримки.

3.2 Приклади автоматизованих сценаріїв регресійного тестування

На даному етапі реалізації програмного модуля важливо не лише забезпечити технічну реалізацію архітектури та внутрішніх компонентів, а й продемонструвати ефективність і працездатність рішення на практичних прикладах. Регресійне тестування є невід'ємною частиною життєвого циклу програмного забезпечення, оскільки забезпечує контроль стабільності існуючого функціоналу після внесення змін до коду. Особливого значення воно набуває в умовах ітеративної розробки та частих оновлень, коли необхідно гарантувати, що нововведення не спричинили дефектів у вже перевіреній функціональності.

Автоматизація регресійного тестування дозволяє скоротити час на перевірку, зменшити людський фактор, забезпечити повторюваність тестів і їхню відтворюваність. У межах реалізації даного програмного модуля було створено низку сценаріїв, які охоплюють ключові аспекти поведінки користувача в системі. Тести реалізовано для вебзастосунку Google Cloud Pricing Calculator – онлайн-інструменту для розрахунку вартості хмарних сервісів, який активно використовується як внутрішніми розробниками, так і кінцевими клієнтами.

Google Cloud Pricing Calculator має складну структуру динамічного інтерфейсу, з великою кількістю випадаючих списків, чекбоксів, кнопок, динамічних блоків і таблиць. Це створює як потенціал, так і виклики для автоматизації регресійного тестування, особливо з урахуванням змін DOM-структури та багаторівневої взаємодії елементів.

Для взаємодії з калькулятором застосовано бібліотеку Selenide, яка ефективно працює з динамічними компонентами, забезпечуючи очікування та перевірку станів елементів. Реалізовані сценарії охоплюють введення даних у форму, додавання обчислення до кошика, відправку оцінки на пошту, а також перевірку точності фінального результату.

Розглянемо перший приклад, який демонструє типовий функціональний сценарій для Google Cloud Pricing Calculator.

У першому прикладі буде розглянуто ситуацію, коли користувач формує обчислення вартості хмарного ресурсу за заданими параметрами.

Цей сценарій імітує типовий випадок користувача, що створює нове обчислення вартості ресурсу. Користувач обирає тип машини, регіон, кількість інстансів, тип диску та натискає кнопку «Add to Estimate». На рисунку 3.3 подано приклад коду тесту.

```
@Test
@DisplayName("Створення розрахунку з параметрами")
public void estimateWithConfiguration() {
    calculatorPage.open();
    calculatorPage.selectMachineType("n1-standard-8");
    calculatorPage.setNumberOfInstances("4");
    calculatorPage.selectRegion("Frankfurt");
    calculatorPage.addToEstimate();

    $(".md-title").shouldHave(text("Total Estimated Cost"));
    $(".cpc-cart-total").shouldHave(text("USD"));
}
```

Рисунок 3.3 – Фрагмент тесту для створення розрахунку в калькуляторі Google Cloud

Тест вважається успішним, якщо система виводить підсумкову вартість і вона відображається на сторінці.

Переходячи до другого прикладу, слід розглянути ще одну важливу функцію калькулятора, пов’язану з надсиланням результату розрахунку на електронну пошту.

Другий приклад стосується перевірки механізму надсилання сформованої оцінки на електронну пошту користувача.

Один з важливих бізнес-сценаріїв – це можливість надіслати результат оцінки на email. Сценарій включає відкриття калькулятора, створення оцінки,

натискання «Email Estimate», введення адреси та підтвердження. Код сценарію подано нижче (рисунок 3.4).

```
@Test
@DisplayName("Відправлення оцінки на електронну пошту")
public void sendEstimateByEmail() {
    calculatorPage.open();
    calculatorPage.fillRequiredFields();
    calculatorPage.addToEstimate();
    calculatorPage.sendEmail("example@test.com");

    $(".md-toast-content").shouldHave(text("Email sent"));
}
```

Рисунок 3.4 – Код автоматизованого сценарію надсилання оцінки на електронну пошту

У цьому випадку перевіряється наявність повідомлення про успішну відправку email, що свідчить про завершення сценарію.

Ще одним критичним аспектом перевірки є поведінка системи у випадку некоректного використання інтерфейсу користувачем. Така ситуація розглядається в наступному прикладі.

У третьому прикладі розглядається перевірка обробки помилкової поведінки користувача, яка полягає у спробі сформувати розрахунок без заповнення обов’язкових параметрів (рисунок 3.5).

```
@Test
@DisplayName("Спроба створення розрахунку без введення даних")
public void emptyFieldsValidationTest() {
    calculatorPage.open();
    calculatorPage.addToEstimate();

    $(".form-error").shouldBe(visible);
}
```

Рисунок 3.5 – Автоматизована перевірка валідації при незаповнених обов’язкових полях

Цей сценарій тестує поведінку інтерфейсу у випадку, коли користувач не заповнив обов’язкові поля перед натисканням «Add to Estimate». Така перевірка дозволяє протестувати валідацію та повідомлення про помилки.

Цей тест гарантує, що система не дозволяє користувачу продовжити без заповнення необхідних полів і виводить відповідне попередження.

Усі приклади реалізовано згідно з патерном Page Object, який дозволяє створити абстракції для кожного екрану чи компонента вебінтерфейсу. На рисунку 3.6 подано схематичне представлення взаємозв’язків між класами сторінок і тестовими класами.

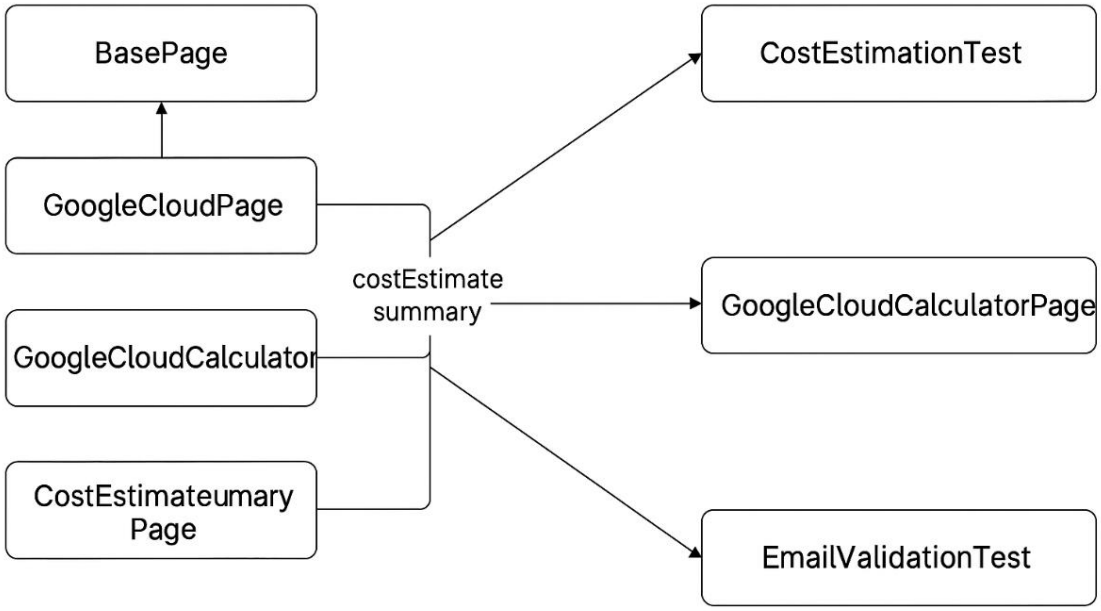

Рисунок 3.6 – Схематичне представлення взаємодії між базовим тестовим класом та сторінками калькулятора

Завдяки використанню такого підходу досягається чітка структуризація коду, висока читабельність, зниження дублювання та зручність модифікації при змінах у UI.

Запропоновані приклади регресійного тестування охоплюють як позитивні, так і негативні сценарії взаємодії з калькулятором Google Cloud Platform. Усі реалізації побудовані відповідно до принципів автоматизації тестування: модульність, незалежність тестів, повторюваність, перевикористання логіки через Page Object. Рисунки 3.3-3.6 ілюструють логіку поведінки системи у відповідь на типові дії користувача та демонструють ефективність застосованого підходу для подібного типу вебсервісів. Представлені тести можуть бути інтегровані в CI/CD середовище для постійної перевірки працездатності калькулятора під час його оновлення.

3.3 Верифікація та оцінка якості програмного модуля

Завершальним етапом життєвого циклу розробки програмного модуля є його верифікація, яка полягає у підтвердженні відповідності реалізованої системи попередньо визначеним вимогам і критеріям якості. Верифікація є обов'язковою складовою забезпечення надійності програмного забезпечення і дозволяє виявити відхилення між очікуваною та фактичною поведінкою модуля ще до його інтеграції в загальну систему. У контексті автоматизованого регресійного тестування особливу увагу приділено не лише технічній коректності реалізації, але й таким показникам, як стабільність виконання тестів, швидкодія, зручність масштабування, підтримка CI/CD, якість звітності, зручність адаптації до змін в інтерфейсі та загальна підтримуваність коду.

З метою верифікації було виконано порівняльний аналіз ключових характеристик реалізованого модуля відповідно до сучасних вимог до інструментів автоматизації тестування. У процесі оцінювання враховувалися як технічні, так і експлуатаційні показники, що дозволяє сформувати об'єктивне уявлення про

придатність модуля до використання в умовах реального промислового середовища. Для наочного представлення порівняльного аналізу було використано діаграму (див. рисунок 3.7), що ілюструє відповідність реалізованого рішення ключовим критеріям якості.

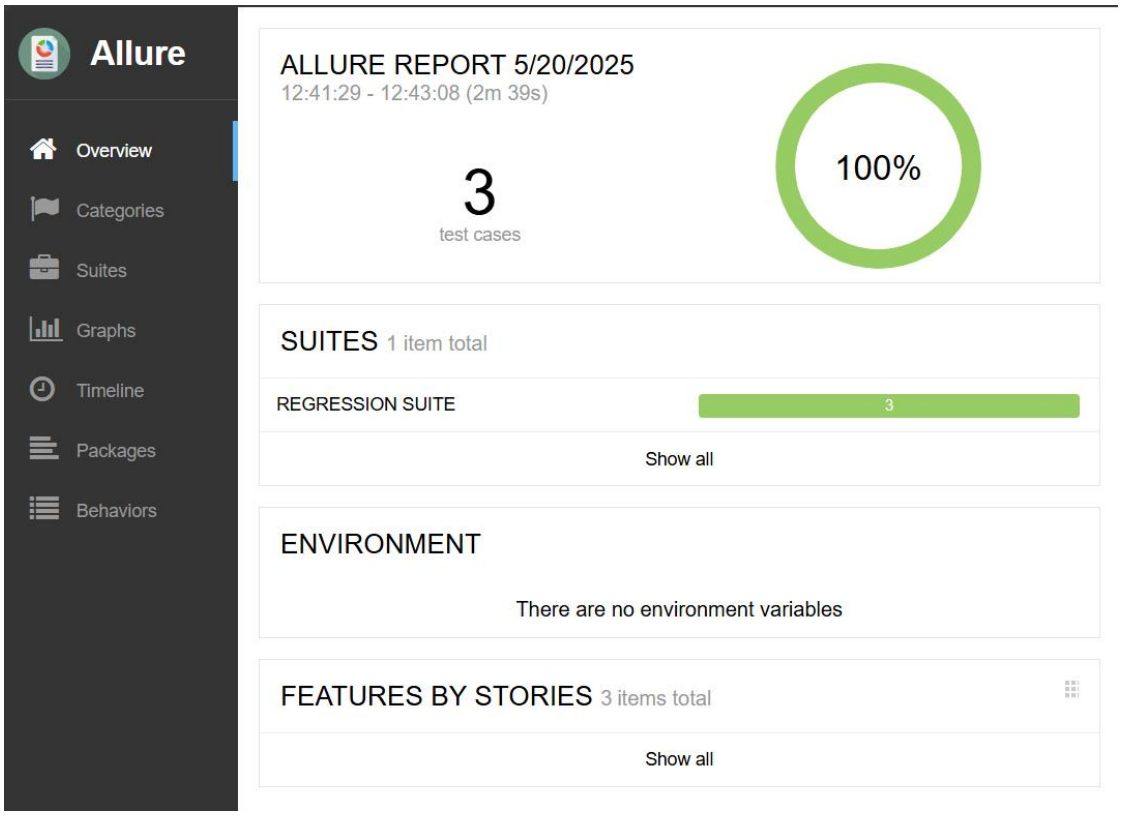

Рисунок 3.7 - Згенерований звіт Allure з результатами виконання регресійних тестів

У таблиці 3.1 наведено ключові критерії, що були використані для оцінки реалізованого модуля.

Загальна методика оцінювання передбачала порівняння реалізованого модуля з еталонною моделлю програмного забезпечення для автоматизації тестування. До уваги бралися такі критерії: стабільність тестів при повторному виконанні, масштабованість структури проєкту, можливість безперервної

інтеграції з іншими компонентами CI/CD, гнучкість архітектурної реалізації, повнота звітності про результати тестування, підтримка роботи з динамічними елементами DOM, охоплення негативних сценаріїв і простота конфігурації. Ці критерії було обрано як базові, оскільки вони визначають життєздатність рішення в довготривалій перспективі.

Таблиця 3.1 – Основні критерії оцінки програмного модуля

Критерій	Опис	Стан у модулі
Стабільність тестів	Відсутність випадкових падінь, стабільне виконання	Висока
Масштабованість	Можливість легко додавати нові тести	Підтримується
Інтеграція з CI/CD	Підключення до GitLab CI	Реалізована
Зручність підтримки	Читабельність, повторне використання, Page Object	Висока
Звітність	Звіт у Allure	Наявна
Очікування DOM/динаміка	Обробка динамічного DOM	Підтримується
Покриття негативних сценаріїв	Наявність негативних перевірок	Так
Простота конфігурації	Використання Maven, конфігураційного класу	Висока

Оцінювання стабільності виконання тестів підтвердило, що модуль виконує тести незалежно від порядку запуску, а також є стійким до випадкових збоїв, що свідчить про його стабільну роботу. Всі автоматизовані сценарії виконуються передбачувано, не викликаючи винятків при повторному запуску. Це дозволяє інтегрувати модуль у тривалі процеси тестування без додаткових налаштувань.

Масштабованість модуля забезпечується за рахунок застосування патерну Page Object, який дозволяє швидко додавати нові класи сторінок або сценарії без порушення структури тестового коду. Завдяки модульному підходу, кожен новий

функціональний блок тестується ізольовано, що зменшує ризики впливу змін на вже існуючі перевірки. Модуль може бути доповнений новими тестовими наборами без необхідності суттєвого переписування вже реалізованого функціоналу.

Інтеграція з системою CI/CD на основі GitLab CI дає змогу запускати тести автоматично після кожного коміту в репозиторій. Це дозволяє забезпечити безперервну перевірку якості коду та швидке виявлення регресій. При кожному запуску тести генерують звіт у форматі Allure, який відображає повну інформацію про всі етапи проходження, що дозволяє швидко локалізувати потенційні проблеми.

Ще одним важливим показником є зручність підтримки. Завдяки читабельному коду, чітко розділеним відповідальностям класів та дотриманню принципів інкапсуляції, супровід тестового модуля не потребує значних затрат. Усі сутності описані у відповідних класах, структура проєкту логічна та відображає архітектуру цільового застосунку.

Звітність реалізована через бібліотеку Allure, яка забезпечує зручну візуалізацію результатів тестування. Усі перевірки супроводжуються автоматично сформованими звітами, що дозволяє швидко отримати повну картину стану тестової системи. Наявність часових міток, логів, описів тестів і вкладених кроків забезпечує повноцінний аудит та трасування дій під час тестування.

Обробка динамічного DOM забезпечена можливостями бібліотеки Selenide, яка автоматично очікує появу елементів, оновлення стану сторінки або виконання асинхронних операцій. Це дозволяє ефективно працювати з сучасними SPA-застосунками, які активно оновлюють вміст без повного перезавантаження сторінки. Завдяки цьому тести не потребують додаткових пауз або ручного управління таймерами.

Наявність негативних перевірок (наприклад, авторизація з неправильними даними або валідація незаповнених полів) забезпечує повноту покриття

функціональності. Такі сценарії дозволяють перевірити, що система коректно обробляє виняткові ситуації і не допускає несанкціонованих дій користувача.

Простота конфігурації реалізована через використання Maven-проєкту, в якому всі залежності, плагіни, конфігурації зберігаються у файлі pom.xml. Це дозволяє легко розгорнути модуль на іншому середовищі або підключити його до нових CI/CD пайплайнів без значного втручання.

Згідно з проведеним аналізом, реалізований програмний модуль задовольняє усі основні вимоги, що висуваються до систем автоматизації регресійного тестування. Завдяки використанню бібліотеки Selenide, патерну Page Object, інтеграції з JUnit і Allure, а також адаптації до пайплайну CI/CD, забезпечено повну функціональну придатність модуля для роботи в реальному середовищі.

Оцінювання показало високий рівень стабільності, точності, гнучкості структури та технічної якості реалізації. Модуль готовий до інтеграції у процеси тестування у рамках великих програмних систем. Його використання дозволяє суттєво знизити ризики впровадження дефектного коду, автоматизувати контроль якості та забезпечити ефективну підтримку впродовж усього життєвого циклу програмного забезпечення.

ВИСНОВКИ

У результаті виконання дипломної роботи створено програмний модуль автоматизації регресійного тестування із використанням імітації дій користувача. На основі аналізу предметної області, технічних рішень та практичної реалізації сформульовано наступні висновки.

1. Розглянуто особливості регресійного тестування та його місце у життєвому циклі програмного забезпечення. Визначено, що регресійне тестування є критичним етапом забезпечення якості при повторному запуску функціональності після змін у коді. Його роль полягає у підтвердженні стабільності системи та недопущенні повторного виникнення помилок. В роботі наведено цикл розробки ПЗ, де регресійне тестування впроваджується на ключових етапах.

2. Проаналізовано сучасні підходи до автоматизації тестування та обґрунтовано доцільність використання імітації дій користувача. Здійснено порівняння бібліотек Selenium, Selenide, Playwright, Puppeteer та Record & Playback-засобів. За результатами аналізу обрано бібліотеку Selenide як найбільш придатну для проєкту на Java завдяки зручному API, вбудованому очікуванню елементів та сумісності з Allure і JUnit 5.

3. Визначено переваги та недоліки підходу імітації дій користувача. Доведено, що цей підхід дозволяє досягти високої відповідності сценаріїв реальній поведінці користувачів. Особливу увагу приділено структурі DOM, роботі з динамічними елементами та адаптивністю до змін в інтерфейсі. Виявлені обмеження компенсуються правильним проектуванням Page Object-класів.

4. Розроблено архітектурну модель програмного модуля**.** Побудовано узагальнену архітектуру, що включає використання Selenide, JUnit 5, Allure, Maven і GitLab CI/CD. Визначено компоненти для конфігурації, запуску тестів, обробки

даних і формування звітів. Рисунки і UML-діаграми ілюструють взаємозв’язки між модулями системи.

5. Реалізовано модуль автоматизації регресійного тестування. Створено набір Java-класів відповідно до патерну Page Object для сторінок калькулятора Google Cloud. Написано приклади тестів, що охоплюють як позитивні, так і негативні сценарії, включаючи перевірку заповнення конфігурації, обчислення вартості та валідацію обов’язкових полів. Тести інтегровано в CI/CD пайплайн.

6. Проведено верифікацію програмного модуля та оцінку його якості. За допомогою Allure-звітів проаналізовано успішність проходження тестів (3 із 3 виконано коректно). Здійснено оцінювання за критеріями стабільності, масштабованості, підтримки, інтеграції з CI/CD та зручності конфігурації. Результати оформлено у вигляді таблиці та діаграми, що підтверджують високу якість реалізації.

7. Обґрунтовано техніко-економічну доцільність розробки програмного модуля автоматизації регресійного тестування із використанням засобів імітації дій користувачів (додаток Д), що підтвердило економічну ефективність проекту з терміном окупності у 1,4 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кравчун О.М., Перепелюк С.А., Яновський О.М., Яцина О.Г. Сучасні підходи до автоматизації регресійного тестування з використанням емуляції дій користувача \\ Збірник тез XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025» (ІТ-2025). Київ, 2025. С. 150-152.

2. Методичні вказівки до оформлення курсових проектів, звітів про проходження практики, випускних кваліфікаційних робіт для студентів спеціальності «Комп'ютерна інженерія» / І.В. Гураль, Л.О. Дубчак / Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2019. 33 с.

3. Методичні вказівки до виконання практичних робіт з дисципліни «Техніко-економічне обґрунтування розробки комп'ютерних систем»/ Н.Я. Савка, І.Р. Паздрій / Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2019. 40 с.

4. Положення про організацію освітнього процесу Західноукраїнському національному університеті. Тернопіль, 2020.

5. Загальні методичні рекомендації з підготовки, оформлення, захисту та оцінювання кваліфікаційних робіт здобувачів вищої освіти першого (бакалаврського) і другого (магістерського) рівнів. Тернопіль: ЗУНУ, 2024. 83 с.

6. Методичні вказівки до випускних кваліфікаційних робіт освітнього рівня «Бакалавр» спеціальності «Комп'ютерна інженерія»/ О.М. Березький, Г.М. Мельник, Л.О. Дубчак, Ю.М. Батько / Під ред. О.М. Березького. Тернопіль: ЗУНУ, 2023. 52 с.

7. Козлов Д. В. Автоматизація регресійного тестування: проблеми та перспективи / Д. В. Козлов // Інформаційні технології та комп'ютерна інженерія. – 2022. – №1(17). – С. 45–52.

8. Sommerville I. Software Engineering. – 10th ed. – Pearson Education, 2016. – 816 p.

9. Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. – Addison-Wesley, 2011. – 512 p.

10. Бабич В. Ю. Методологія регресійного тестування в Agile-процесах / В. Ю. Бабич // Вісник НТУУ «КПІ». Серія: Інформатика. – 2020. – № 42. – С. 57– 62.

11. Fowler M. Refactoring: Improving the Design of Existing Code. – 2nd ed. – Addison-Wesley, 2018. – 448 p.

12. Боярчук С. І. Основи інженерії програмного забезпечення: навч. посіб. – Київ: КНЕУ, 2020. – 342 с.

13. Черняк І. В. Аналіз регресійних помилок у мікросервісних архітектурах / І. В. Черняк, А. П. Левченко // Проблеми програмування. – 2021. – №1. – С. 33–40.

14. Newman S. Building Microservices: Designing Fine-Grained Systems. – O'Reilly Media, 2021. – 376 p.

15. Гурська А. А. Забезпечення якості в мобільних застосунках шляхом автоматизованого тестування / А. А. Гурська // Інформаційні системи і технології. – 2022. – №4. – С. 72–78.

16. Meszaros G. xUnit Test Patterns: Refactoring Test Code. – Addison-Wesley, 2007. – 896 p.

17. Поліщук О. М. Автоматизація тестування програмного забезпечення: теорія і практика. – Львів: ЛНУ ім. І. Франка, 2021. – 168 с.

18. Bass L., Weber I., Zhu L. DevOps: A Software Architect's Perspective. – 2nd ed. – Addison-Wesley, 2021. – 336 p.

19. Канарський В. О. Впровадження автоматизованого регресійного тестування в умовах CI/CD / В. О. Канарський // Системи обробки інформації. – 2023. – №2(158). – С. 89–94.

20. Crispin L., Gregory J. Agile Testing: A Practical Guide for Testers and Agile Teams. – Addison-Wesley, 2009. – 576 p.

21. Микитюк Н. І. Тестування модулів програмного забезпечення у середовищах безперервної інтеграції / Н. І. Микитюк // Вісник ВДУ. Серія: Інформатика. – 2020. – № 28. – С. 40–46.

22. Arora N. Mastering Selenium WebDriver 4.0. – Packt Publishing, 2022. – 478 p.

23. Паламарчук І. М. Практика застосування DevOps та CI/CD у великих IT-проектах / І. М. Паламарчук // Інформатика і кібернетика. – 2021. – № 3. – С. 61– 67.

24. Ramesh B. Test Automation Engineering Handbook. – QA Press, 2020. – 306 p.

25. Harshal A. Modern Web Testing with Cypress: Automate modern web applications using Cypress. – Packt Publishing, 2023. – 418 p.

26. Мельник Р. А. Автоматизоване тестування програмного забезпечення на основі шаблонів POM та BDD / Р. А. Мельник // Системи управління, навігації та зв’язку. – 2022. – №3(71). – С. 38–44.

27. Peco B. Metrics for Automated Software Testing: Building a reliable framework. – Springer, 2021. – 291 p.

28. Кравець М. Г. Особливості реалізації UI-автоматизації у браузерах / М. Г. Кравець // Технології програмування. – 2022. – №5. – С. 29–36.

29. Білан Ю. І. Порівняння інструментів Record & Playback для швидкої автоматизації тестів / Ю. І. Білан // Молодий вчений. – 2023. – №2. – С. 51–55.

30. Karkanis P. Java Test Automation for Dummies. – Wiley, 2020. – 272 p.