ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Підвищення ефективності автоматизованого тестування інтернет-магазину з використанням технології Behavior Driven Development»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
(код, найменування спеціальності)
освітньо-професійної програми «Інженерія програмного забезпечення»
(назва)

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

_____ Лілія ЄМЕЦЬ
(підпис)

Виконав: здобувачка вищої освіти групи ПДМ-62
Лілія ЄМЕЦЬ

Керівник: Олена НЕГОДЕНКО
к.т.н., доцент

Рецензент: Вікторія КОРЕЦЬКА
к.пед.н., доцент

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Ємець Лілії Михайлівні

1. Тема кваліфікаційної роботи: «Підвищення ефективності автоматизованого тестування інтернет-магазину з використанням технології Behavior Driven Development»

керівник кваліфікаційної роботи Олена НЕГОДЕНКО к.т.н., доцент,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» жовтня 2023 р. №145.

2. Строк подання кваліфікаційної роботи «29» грудня 2023 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, набори тестових сценаріїв, вимоги до CI/CD, вимоги до часу виконання.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Дослідити принципи методології Behavior Driven Development.
2. Проаналізувати інструменти для впровадження BDD.
3. Створити модель для визначення оптимального набору тестів.
4. Розробити тести відповідно до принципів Behavior Driven Development.

5. Отримати практичні результати впровадження BDD.

5. Перелік графічного матеріалу: *презентація*
   1. Порівняльний аналіз методів автоматизованого тестування.
   2. Основні принципи методології Behavior Driven Development.
   3. Аналіз існуючих інструментів автоматизованого тестування UI.
   4. Математична модель вибору оптимального набору тестів.
   5. Схема процесу створення тестових сценаріїв.
   6. Приклад процесу створення тестових сценаріїв.

6. Дата видачі завдання «19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	19.10-05.11.23	
2	Вивчення матеріалів для аналізу методів автоматизованого тестування	06.11-12.11.23	
3	Дослідження принципів методології Behavior Driven Development	13.11-19.11.23	
4	Аналіз інструментів автоматизованого тестування	20.11-26.11.23	
5	Впровадження методології Behavior Driven Development в тестуванні	27.11-03.12.23	
6	Дослідження результатів після впровадження Behavior Driven Development	04.12-10.12.23	
7	Оформлення роботи: вступ, висновки, реферат	11.12-20.12.23	
8	Розробка демонстраційних матеріалів	21.12-29.12.23	

Здобувачка вищої освіти _____ Лілія ЄМЕЦЬ
(підпис)

Керівник
кваліфікаційної роботи _____ Олена НЕГОДЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 70 стор., 8 табл., 24 рис., 30 джерел.

Мета роботи – підвищення якості та надійності Internet-магазину за рахунок вдосконалення процесу автоматизованого тестування з використанням технології BDD.

Об'єкт дослідження – автоматизоване тестування інтернет-магазину. *Предмет дослідження* – технології Behavior Driven Development.

Короткий зміст роботи: У роботі проведено дослідження автоматизованого тестування Internet-магазину з використанням технології Behavior Driven Development (BDD). Зазначена технологія є потужним інструментом для забезпечення високої якості програмного забезпечення, особливо в контексті складних веб-додатків, таких як інтернет-магазини.

Основні етапи дослідження включають аналіз сучасних підходів до автоматизованого тестування, розробку та впровадження тестових сценаріїв з використанням мови програмування Java та технології BDD.

Результати дослідження показують, що використання технології BDD сприяє підвищенню ефективності автоматизованого тестування інтернет-магазину.

КЛЮЧОВІ СЛОВА: АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, ІНТЕРНЕТ-МАГАЗИН, ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ, BEHAIVIOR DRIVEN DEVELOPMENT (BDD).

ABSTRACT

Text part of the master's qualification work: 70 pages, 24 pictures, 8 table, 30 sources.

The purpose of the work is to improve the quality and reliability of the online store due to the test automation process improvement using the BDD technology.

Object of research – automated testing of the online store.
Subject of research – Behavior Driven Development technologies.

Summary of the work: The study of automated testing of an online store using Behavior Driven Development (BDD) technology was carried out. This technology is a powerful tool for ensuring high software quality, especially in the context of complex web applications such as online stores.

The main stages of the research include the analysis of modern approaches to test automation, development, and implementation of test scenarios using the Java programming language and BDD technology.

The results of the study show that the use of BDD technology contributes to increasing the efficiency of automated testing of an online store.

KEYWORDS: TEST AUTOMATION, ONLINE STORE, EFFICIENCY IMPROVEMENT, BEHAVIOR DRIVEN DEVELOPMENT (BDD).

ЗМІСТ

ВСТУП

Сучасні технології інформаційного середовища висувають перед собою надзвичайно високі вимоги до якості та надійності програмного забезпечення, особливо у сфері електронної комерції. Зростання обсягів та складності програмних продуктів, а також стрімке розширення можливостей сучасних інтернет-магазинів визначають необхідність застосування ефективних методів автоматизованого тестування для забезпечення їх найвищої якості.

У цьому контексті технологія Behavior Driven Development (BDD) визнається однією з передових стратегій в розробці програмного забезпечення. Її унікальність полягає в акценті на взаємодії між розробниками, тестувальниками та фахівцями з предметної області, а також у створенні зрозумілих для всіх учасників тестових сценаріїв.

Дана робота спрямована на підвищення ефективності автоматизованого тестування інтернет-магазину, використовуючи технологію BDD. Сучасний конкурентний ринок електронної комерції вимагає від підприємств не тільки функціональних та естетичних рішень, але й найвищого рівня надійності та якості їхнього програмного забезпечення.

Мета роботи і задачі дослідження полягають у вивченні технології BDD, її переваг та недоліків відповідно до традиційного підходу і як наслідок підвищенні якості та надійності Internet-магазину за рахунок вдосконалення процесу автоматизованого тестування.

Досягнення вказаної мети забезпечується розв'язуванням наступних задач:

- Аналіз наявної науково-технічної літератури.
- Вивчення матеріалів для аналізу методів автоматизованого тестування.
- Дослідження принципів методології Behavior Driven Development.
- Аналіз інструментів автоматизованого тестування.
- Впровадження методології Behavior Driven Development в тестуванні.
- Дослідження результатів після впровадження Behavior Driven Development.

Наукова новизна одержаних результатів полягає в комплексному підході до вдосконалення процесу автоматизації тестування. За рахунок чого впровадження впровадження технології BDD відбувалось поступово відповідно до попередньо проведеного аналізу. Що в свою чергу сприяє покращенню розуміння процесів автоматизованого тестування та підвищенню якості та надійності інтернет-магазину, що є надзвичайно важливим в умовах швидко зростаючого електронного ринку.

Практичне значення одержаних результатів полягає в мінімізації часу на розробку автотестів, підвищенні показників покриття вимог автотестами, і як наслідок зниженні кількості дефектів у програмному забезпеченні і покращенні комунікації між учасниками розробки, що є ключовим чинником успіху в розробці складних веб-додатків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

Behavior Driven Development (BDD) представляє собою передовий підхід у сфері розробки програмного забезпечення, що акцентує на взаємодії між розробниками, тестувальниками та фахівцями з предметної області. Ця методологія визначається створенням зрозумілих для всіх стейкхолдерів тестових сценаріїв на основі бізнес-потреб, що полегшує комунікацію та покращує зрозуміння вимог[1].

Виокремлюючись серед інших методів розробки, BDD вже зараз показує вражаючі результати в плані підвищення якості програмного забезпечення та прискорення часу виконання проектів. Згідно з дослідженням "State of Agile Report," 85% команд, які використовують BDD, стверджують про покращення у взаєморозумінні між учасниками команди, що безпосередньо відображається на якості вихідного продукту.

Більше того, згідно зі статистикою від "TechBeacon," компанії, які успішно впроваджують BDD, відзначають зменшення кількості дефектів на етапі тестування на 25-50%, що приводить до значної економії коштів на подальших етапах розробки.

Зростання популярності BDD свідчить про те, що цей підхід стає стандартом у розробці програмного забезпечення, і його впровадження може стати ключовим фактором для успіху в проектах будь-якої складності[15]. В даному контексті дослідження та оптимізація використання BDD в автоматизованому тестуванні інтернет-магазинів має великий потенціал для покращення ефективності та якості розробки програмного забезпечення.

1.1 Обґрунтування важливості використання автоматизованого тестування інтернет-магазинів

Інтернет-магазини стали неодмінною частиною сучасного електронного бізнесу, і їхній успіх напряму пов'язаний із задоволенням потреб споживачів та конкурентоспроможністю на ринку. В умовах постійної конкуренції та зростаючих вимог до якості та функціональності, важливо надавати високоякісні, надійні та безпечні електронні платформи для покупок.

1.1.1 Забезпечення якості продукту

Автоматизоване тестування виявляється необхідним інструментом у забезпеченні високої якості програмного забезпечення інтернет-магазинів. Сучасне споживання вимагає не лише розмаїтості товарів та зручності інтерфейсу, але й бездоганної функціональності електронного середовища для покупок[2]. Розглянемо, чому автоматизоване тестування є ключовим компонентом у досягненні цього завдання.

Ефективне покриття функціональності – Автоматизоване тестування дозволяє ефективно перевіряти всі аспекти функціональності інтернет-магазину. Від взаємодії з користувачем, оформлення замовлення до обробки платежів, тестові сценарії можуть охоплювати всі можливі шляхи взаємодії з клієнтом. Це дозволяє виявити та усунути потенційні проблеми, які можуть виникнути під час реального використання магазину.

Забезпечення стабільності та надійності – Автоматизовані тести дозволяють проводити повторювані перевірки, що гарантує стабільність та надійність роботи інтернет-магазину під час різних умов. Вони допомагають виявляти дефекти та помилки, які можуть вплинути на продуктивність магазину, ще до випуску нових функцій або оновлень.

Автоматизоване виконання тестів на різних конфігураціях – Інтернет-магазини повинні бути доступні та ефективні на різних пристроях і веб-браузерах.

Автоматизовані тести можуть запускатися на різних конфігураціях, перевіряючи сумісність та забезпечуючи єдність користувацького досвіду.

Автоматизована перевірка безпеки – За допомогою автоматизованих тестів можна ефективно перевіряти рівень безпеки інтернет-магазину. Вони допомагають виявити потенційні слабкі місця та уразливості, що дозволяє забезпечити високий рівень захисту для особистої інформації користувачів[16].

Отже, автоматизоване тестування стає невід'ємною частиною стратегії забезпечення якості продукту в інтернет-магазинах, дозволяючи ефективно виявляти, вирішувати та попереджувати можливі проблеми для забезпечення високого стандарту функціональності та задоволення клієнтів.

1.1.2 Зменшення часу виправлення дефектів

Автоматизоване тестування стає потужним інструментом для зменшення часу виправлення дефектів у процесі розробки інтернет-магазинів. Розглянемо, як цей підхід сприяє швидкому виявленню, аналізу та усуненню проблем на початкових етапах життєвого циклу проекту.

Раннє виявлення дефектів – Автоматизоване тестування дозволяє запускати тести на ранніх етапах розробки, що робить можливим виявлення дефектів до введення продукту в етап виробництва. Це важливо, оскільки чим раніше виявлено проблему, тим менше часу та ресурсів витрачається на її виправлення.

Миттєвий звіт про дефекти – Автоматизовані тести надають можливість одразу отримувати детальні звіти про виявлені дефекти. Це дозволяє розробникам та тестувальникам швидко оцінити природу та серйозність проблеми, сприяючи ефективному виправленню без подовження циклу розробки.

Збалансованість між розробкою та тестуванням – Автоматизоване тестування забезпечує більшу збалансованість між фазами розробки та тестування, дозволяючи одночасно вдосконалювати код та виконувати тестування. Це дозволяє уникнути накопичення великої кількості дефектів і виправляти їх на ранніх етапах.

Прискорена реакція на зміни – Завдяки автоматизованому тестуванню, команди можуть швидко реагувати на зміни в вимогах або коді, перевіряючи вплив

змін на функціональність магазину. Це сприяє зниженню часу на виправлення дефектів, що виникають під час розробки.

Мінімізація вартості розробки – Зменшення часу виправлення дефектів призводить до економії коштів на розробці[3]. Вартість виправлення проблеми на початкових етапах значно менше, ніж виправлення того самого дефекту на пізніших стадіях розробки або після випуску продукту.

Отже, автоматизоване тестування не лише дозволяє швидше виявляти дефекти, але і активно сприяє їхньому оперативному виправленню, що призводить до зменшення часу, витраченого на цей процес, та зниження загальної вартості розробки.

1.1.3 Покращення користувацького досвіду

Виявлення та усунення проблем перед випуском – Автоматизоване тестування дозволяє оперативно виявляти потенційні проблеми, які можуть вплинути на користувацький досвід, ще до випуску продукту. Це включає в себе перевірку швидкості завантаження сторінок, коректність відображення на різних пристроях та перевірку реакції системи на різні дії користувачів[17].

Ефективне тестування взаємодії з користувачем – Автоматизовані тести можуть охоплювати сценарії взаємодії з користувачем, такі як заповнення форм, оформлення замовлення, та навігацію по сайту. Це дозволяє переконатися, що всі функції працюють бездоганно та забезпечують комфорт користувачам під час їх візиту на сайт.

Забезпечення плавної роботи функцій – Автоматизоване тестування гарантує, що всі функції інтернет-магазину працюють плавно та ефективно. Від процесу додавання товарів до кошика до завершення оплати, тести впевнюються, що користувачі можуть легко та з задоволенням взаємодіяти з усіма аспектами сайту.

Перевірка адаптивності та реакції на різні пристрої – Автоматизоване тестування забезпечує перевірку адаптивності магазину до різних пристроїв та розмірів екранів. Це гарантує, що користувачі можуть зручно користуватися

магазином як на комп'ютері, так і на мобільному пристрої, що важливо для сучасного споживача.

Виявлення та усунення помилок швидше, ніж виявленням користувачами – Швидке виявлення та усунення помилок завдяки автоматизованому тестуванню дозволяє уникнути негативного впливу на користувацький досвід. Це забезпечує стабільність та безперебійну роботу магазину для кінцевих користувачів.

Отже, автоматизоване тестування вирішує завдання не лише виявлення проблем, а й активно сприяє покращенню користувацького досвіду шляхом забезпечення оптимальної роботи всіх функцій інтернет-магазину.

1.1.4 Економія ресурсів

Використання автоматизованих тестів у процесі розробки інтернет-магазинів призводить до суттєвої економії ресурсів, що стає ключовим фактором для оптимізації робочого процесу та забезпечення ефективного використання розробників.

Автоматизація рутинних та повторюваних завдань – Автоматизовані тести можуть виконувати повторювані сценарії тестування швидше і надійніше, порівняно з ручним тестуванням[4]. Це дозволяє розробникам уникати витрат часу на рутинні завдання, такі як перевірка базової функціональності чи взаємодії з користувачем.

Зосередження на важливих завданнях – Ефективне використання автоматизованих тестів вивільняє час та увагу розробників, дозволяючи їм зосередитися на важливих завданнях, таких як вдосконалення функціоналу, оптимізація роботи з базою даних чи впровадження нових можливостей. Це сприяє підвищенню продуктивності та розвитку проекту.

Мінімізація витрат на виправлення дефектів – Автоматизоване тестування дозволяє виявляти та усувати дефекти на ранніх етапах розробки, що призводить до зменшення витрат на виправлення проблем у подальшому. З розширеним тестовим покриттям можна уникнути затрат на виправлення критичних помилок після випуску продукту[18].

Підвищення робочої ефективності – Автоматизовані тести розглядають широкий спектр сценаріїв тестування за короткий період часу, що дозволяє швидше виявляти проблеми та робити висновки про якість програмного продукту. Це підвищує робочу ефективність розробників та забезпечує ефективне використання ресурсів.

Створення стабільного фундаменту для подальшого розвитку – Заощаджені ресурси завдяки автоматизованому тестуванню можуть бути використані для покращення і розширення функціоналу інтернет-магазину, створюючи стабільний фундамент для подальшого розвитку проекту.

Отже, ефективне використання автоматизованих тестів сприяє економії часу та ресурсів розробників, дозволяючи їм зосередитися на стратегічних завданнях та розвитку проекту, що в свою чергу призводить до підвищення продуктивності та якості програмного продукту.

Таким чином, використання автоматизованого тестування інтернет-магазинів стає стратегічно важливим елементом у забезпеченні високої якості продукту, підвищенні ефективності розробки та створенні задоволеного та лояльного клієнтського середовища.

1.2 Огляд існуючих методів та технологій тестування

Огляд існуючих методів та технологій тестування в інтернет-магазинах визначає стратегію та підходи до забезпечення високої якості програмного забезпечення. Розглянемо ключові аспекти цього огляду, що включає в себе важливі методи та технології для покращення ефективності автоматизованого тестування.

Методологія Behavior Driven Development (BDD)

Методологія Behavior Driven Development (BDD) є інтегрованим підходом до розробки програмного забезпечення, активно використовуваним у сфері автоматизованого тестування інтернет-магазинів[19]. Цей метод покладає акцент

на взаємодію ключових учасників розробки та тестування, створюючи спільну мову для опису функціональності та очікувань щодо продукту.

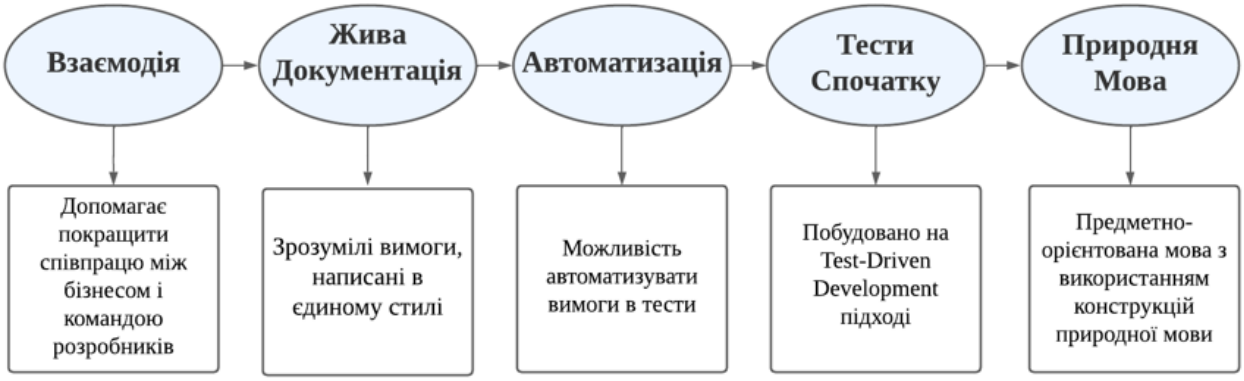


Рис. 1.1 Методологія BDD

Основні принципи BDD

- Взаємодія замовника, розробника та тестувальника – BDD розглядає усіх учасників процесу розробки як рівноправних учасників команди. Замовник, розробник і тестувальник працюють разом, вирішуючи питання функціональності та приймаючи спільні рішення для досягнення високої якості продукту.
- Сценарії тестування в природній мові – Однією з ключових особливостей BDD є використання природної мови для вираження сценаріїв тестування[5]. Це дозволяє усім учасникам зрозуміти та спільно працювати над тестовими випробуваннями, навіть якщо вони не мають глибокого технічного розуміння.
- Спільне розуміння функціональності – BDD сприяє створенню спільного розуміння функціональності продукту. Сценарії тестування стають основою для обговорення, що дозволяє уточнювати вимоги та узгоджувати очікування усіх сторін.

Успішна реалізація методології Behavior Driven Development (BDD) вимагає використання спеціалізованих інструментів, які дозволяють перетворювати природну мову в автоматизовані тести та полегшують спільну роботу всіх

учасників процесу розробки. Декілька ключових інструментів BDD включають Cucumber, Behave, та JBehave.

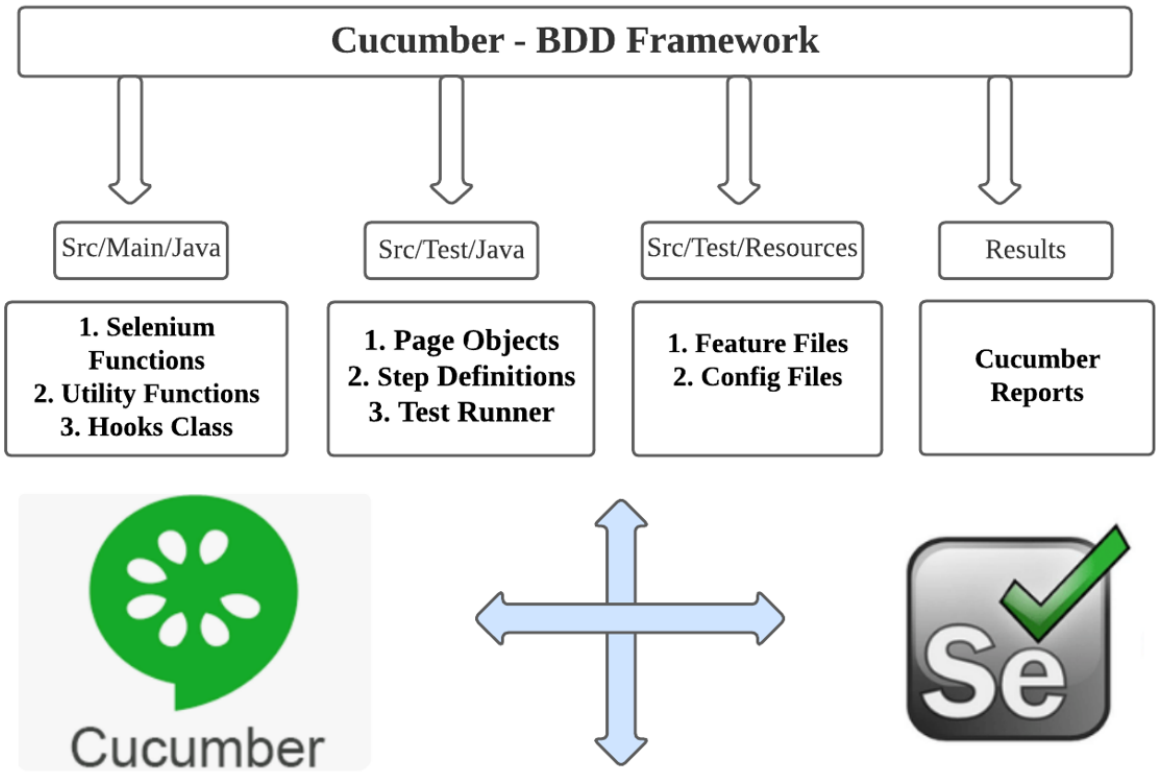


Рис. 1.2 Інструмент для тестів Cucumber

Cucumber є одним з найпоширеніших інструментів для виконання тестів, написаних на мові природи. Він підтримує різні мови програмування, такі як Java, JavaScript, Ruby, та інші. Cucumber використовує синтаксис Gherkin для опису сценаріїв тестування.

Переваги:

- Простий та читабельний синтаксис Gherkin полегшує розуміння та написання тестів.
- Підтримка для різних мов програмування розширює область застосування інструменту.

Автоматизоване тестування інтерфейсу користувача (UI)

Автоматизоване тестування інтерфейсу користувача (UI) є ключовим елементом забезпечення високої якості та ефективності інтернет-магазину. Взаємодія користувача з веб-сторінкою вимагає ретельного перевіряння, щоб гарантувати зручність та надійність використання[6]. Для цього використовуються спеціалізовані інструменти, такі як Selenium, Cypress чи Playwright.

Інструменти автоматизованого тестування UI:

Selenium є одним з найпоширеніших інструментів для автоматизованого тестування веб-додатків. Він підтримує різні мови програмування, включаючи Java, Python, та JavaScript.

Cypress — це інструмент для автоматизованого тестування веб-додатків, спеціально призначений для тестування відмінного від Selenium.

Playwright — це інструмент для автоматизованого тестування та автоматизованого взаємодії з веб-додатками.

Автоматизовані тести UI дозволяють відтворювати взаємодію користувача з веб-сторінкою, перевіряючи правильність реакції системи на різні дії.

Кожен інструмент має свої переваги та недоліки. Вибір між ними повинен базуватися на конкретних потребах проекту та зручності використання для команди розробників і тестувальників[20]. Cypress вигідний для тестування одного браузера, тоді як Selenium та Playwright надають більше гнучкості в роботі з різними браузерами та мовами програмування.

Отримання покриття коду є ключовим етапом в процесі автоматизованого тестування інтернет-магазинів. Використання спеціальних інструментів для вимірювання покриття коду, таких як JaCoCo для Java, визначає, які частини програмного забезпечення покриті автоматизованими тестами. Це є критично важливим для забезпечення повноти тестового покриття та вчасного виявлення проблем.

Таблиця 1.1

Порівняння інструментів автоматизованого тестування UI

Характеристика	Selenium	Cypress	Playwright
Мова програмування	Багато (Java, Python, інші)	JavaScript	Багато (JavaScript, Python, інші)
Синтаксис тестів	Gherkin (за допомогою розширень)	Специфічний для Cypress	Специфічний для Playwright
Крос-платформеність	Так	Так	Так
Швидкість виконання тестів	Залежить від конфігурації, може бути повільніше в порівнянні з іншими	Швидший ніж Selenium	Висока швидкість виконання
Взаємодія з браузерами	Підтримує багато браузерів	Основний фокус на Chrome та Electron	Крос-браузерна підтримка
Можливість відлагодження тестів	Змішана підтримка, включаючи сторонні інструменти	Вбудована зручна можливість відлагодження	Вбудована зручна можливість відлагодження
Активність спільноти	Активна	Активна	Активна
Відомість в галузі BDD	Інтеграція з розширеннями для Gherkin	Підтримка Gherkin	Спеціалізований інструмент для автоматизованого тестування
Простота вивчення	Середня	Висока	Середня

В таблиці 1.1 порівняно найбільш відомі на використовуємі інструменти автоматизованого тестування UI.

Інструменти вимірювання покриття коду

1. JaCoCo (Java Code Coverage)

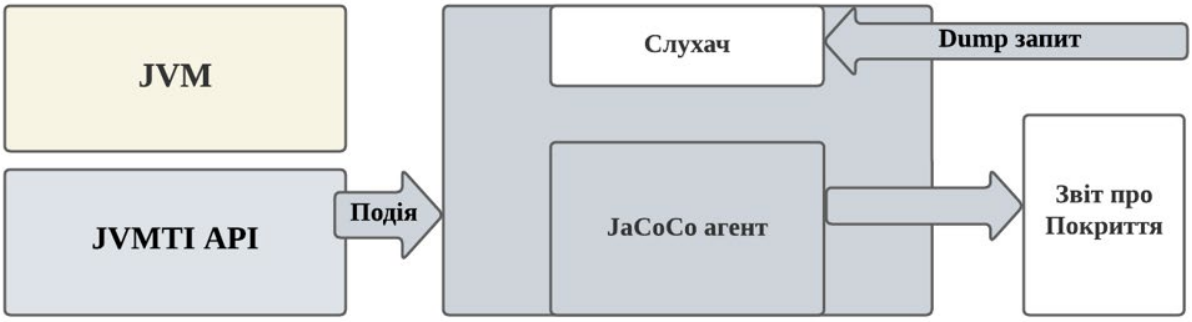


Рис. 1.3 Інструмент для вимірювання покриття коду JaCoCo

JaCoCo - це інструмент для вимірювання покриття коду для мови програмування Java. Він аналізує байт-код Java, надаючи детальну інформацію про те, які частини коду були викликані під час виконання тестів.

Вимірювання покриття коду дозволяє визначити, наскільки ефективно автоматизовані тести охоплюють різні частини програмного забезпечення. Часткове або відсутнє покриття деяких частин коду може свідчити про наявність проблем або потенційних ризиків, які можуть виникнути під час роботи програми.

Аналіз результатів покриття дозволяє команді тестування зосередитися на тих частинах коду, які потребують додаткових тестів, що допомагає оптимізувати ресурси та забезпечити повноцінне покриття.

Використання контейнеризації

Контейнеризація, особливо за допомогою Docker, виявляється важливим інструментом у процесі тестування інтернет-магазинів. Ця технологія надає можливість створювати ізольоване середовище для тестування, що полегшує встановлення та використання необхідних залежностей для тестових сценаріїв[7].

Кожен тестовий сценарій виконується в ізольованому контейнері, що забезпечує консистентність результатів незалежно від місця виконання.

Використання Docker дозволяє легко відтворювати тестові середовища як на розробникових машинах, так і в інших середовищах, наприклад, на серверах для автоматичного тестування.

Контейнеризація легко інтегрується з іншими інструментами для тестування, такими як інструменти автоматизованого тестування та системи збірки.

Використання контейнеризації, зокрема Docker, у тестуванні інтернет-магазинів робить процес тестування ефективнішим, легко відтворюваним та ізольованим від робочого середовища. Ця технологія сприяє швидкому розгортанню тестових середовищ та збереженню консистентності результатів на різних етапах розробки.

Використання Continuous Integration та Continuous Deployment

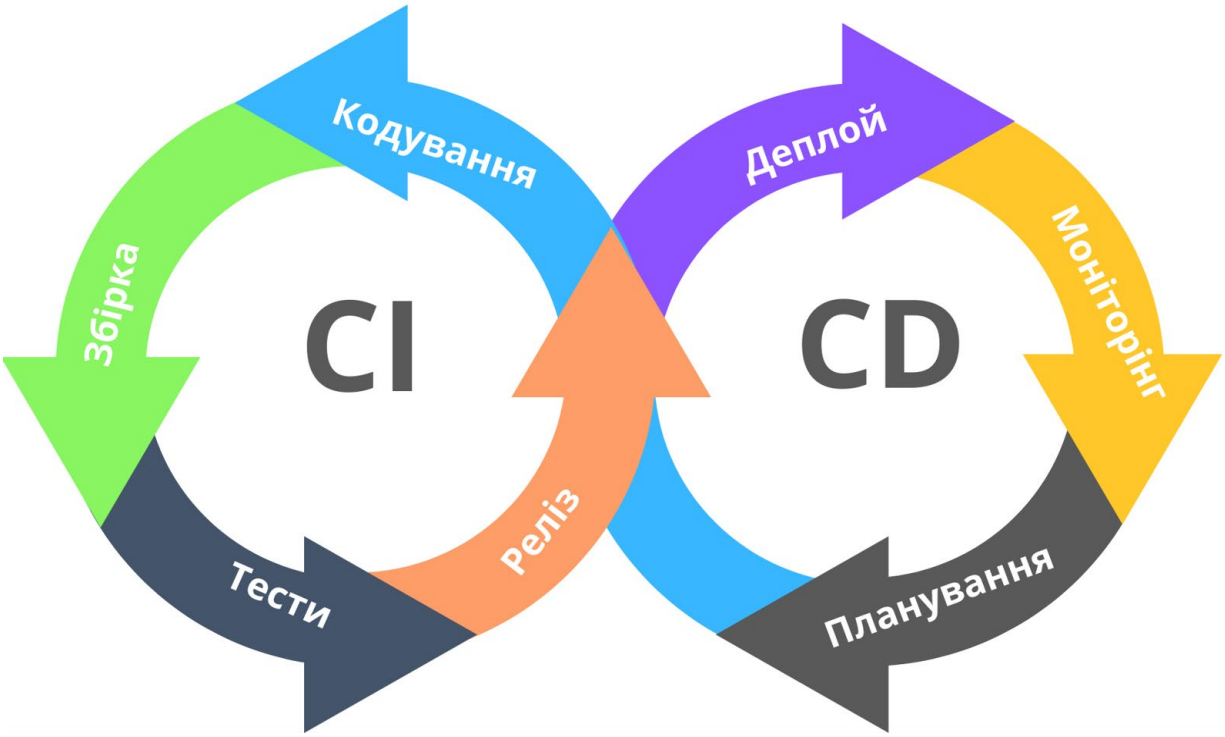


Рис. 1.4 CI\CD

Continuous Integration (CI) та Continuous Deployment (CD) стають необхідними складовими в розробці інтернет-магазинів. CI включає автоматизоване збирання програмного забезпечення та запуск автоматизованих тестів при кожній зміні, дозволяючи виявляти помилки на ранніх етапах і забезпечуючи консистентність коду.

CD включає автоматичне розгортання програмного забезпечення після успішного CI, забезпечуючи швидке та надійне впровадження нового функціоналу. Моніторинг та системи логування допомагають вчасно виявляти та вирішувати проблеми в реальному часі. Здійснення релізів на виробництво автоматизує

підготовку та розгортання нових версій на виробничому сервері, зменшуючи час випуску нового функціоналу та забезпечуючи стабільність системи[8].

Ці практики не лише поліпшують якість програмного забезпечення, а й роблять розробку більш ефективною та реактивною до змін.

Аналіз предметної області та постановка завдань дослідження визначили ключові виклики та можливості, пов'язані із впровадженням технології Behavior Driven Development (BDD) в контексті автоматизованого тестування інтернет-магазинів. Згідно з проведеним дослідженням, було встановлено наступне:

1. **Аналіз сучасного стану.** Вивчення сучасних підходів та методик автоматизованого тестування підтвердило, що BDD може відігравати важливу роль у поліпшенні якості та ефективності цього процесу. Синтаксис Gherkin та спрощений спосіб написання тестів забезпечують зрозумілу та ефективну комунікацію між усіма учасниками розробки.

2. **Постановка завдань.** Визначення завдань дослідження включало в себе розробку моделі досліджуваної системи, визначення обсягу застосування BDD та аналіз його впливу на ключові показники ефективності тестування. Були визначені конкретні функціональні блоки, які підлягали тестуванню, такі як авторизація, пошук товарів, оформлення замовлення та оплата.

3. **Мета та очікувані результати.** Основною метою дослідження було визначення, наскільки ефективно впровадження BDD може покращити процес автоматизованого тестування в інтернет-магазині. Очікувалися позитивні зрушення в часі написання тестових сценаріїв, рівні зрозумілості та співпраці в команді, кількості виявлених помилок, покритті тестування, часі автоматизації виконання тестів, якості продукту та аналізі результатів тестування.

В цілому, аналіз предметної області та постановка завдань дослідження визначили важливі аспекти для подальшого розвитку та оптимізації процесу автоматизованого тестування в інтернет-магазинах з використанням технології BDD.

2 ТЕХНОЛОГІЇ BEHAVIOR DRIVEN DEVELOPMENT ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ АВТОМАТИЗОВАНОГО ТЕСТУВАНЯ ІНТЕРНЕТ-МАГАЗИНУ

2.1 Побудова моделі досліджуваної системи

Впровадження технологій Behavior Driven Development (BDD) належним чином розпочинається з ретельної побудови моделі досліджуваної системи, спрямованої на досягнення максимальної ефективності автоматизованого тестування інтернет-магазину.

Визначення функціональності для BDD – На даному етапі проводиться детальне визначення ключової функціональності інтернет-магазину, яка підлягає тестуванню за допомогою BDD.

Метою є чітке обмеження області застосування BDD з метою максимально ефективного використання цієї технології. Визначення конкретних функціональних блоків та їх взаємодії дозволить точно визначити межі тестування.

Результат очікуваної функціональності:

Авторизація та реєстрація користувачів – У процесі автоматизованого тестування системи авторизації та реєстрації користувачів, ми зосередимо увагу на визначенні високих параметрів безпеки для захисту особистої інформації. Зокрема, важливо перевірити ефективність валідації паролів та впевнитися в коректному зберіганні та обробці особистої інформації користувачів. При створенні автоматизованих тестів будуть враховані різні сценарії авторизації та виправлення можливих помилок.

Пошук та фільтрація товарів – Автоматизоване тестування пошуку та фільтрації товарів в інтернет-магазині спрямоване на перевірку точності сортування та швидкості пошуку. Ми ретельно протестуємо різні сценарії пошуку за різними критеріями та впевнимось в коректності відображення результатів.

Оформлення замовлення та оплата – Процес тестування оформлення замовлення та оплати включатиме в себе тестування інтерфейсу та функціональності оформлення замовлення, а також взаємодію з системами платіжних систем. Симуляція різних сценаріїв оплати дозволить нам переконатися в надійності цього процесу.

Для ілюстрації, розглянемо сценарій реєстрації нового користувача. Автоматизований тест буде включати в себе введення коректних даних, перевірку валідації електронної пошти та пароля, а також переконання в тому, що новий користувач може успішно авторизуватися після реєстрації. Такий тест дозволяє перевірити всі аспекти процесу реєстрації та авторизації, забезпечуючи їхню правильність та безпеку.

Успішне пройдення автоматизованих тестів на всіх етапах взаємодії з користувачем, відсутність помилок та затримок при авторизації та реєстрації, точність та ефективність пошуку товарів, безперебійна робота інтерфейсу оформлення замовлення та оплати[9]. Таке тестування забезпечить високу якість та надійність функціональності інтернет-магазину, забезпечуючи задоволення користувачів та уникнення потенційних проблем.

Очікувані результати

- Чітко визначена область застосування BDD в рамках функціональності інтернет-магазину.
- Деталізована модель, що охоплює основні функціональні блоки.
- Визначені критичні точки для подальшого тестування та вдосконалення.

Побудова моделі досліджуваної системи визначає основи для успішного впровадження BDD та забезпечення високої ефективності автоматизованого тестування інтернет-магазину.

2.2 Технології Behavior Driven Development

Технології Behavior Driven Development (BDD) представляють сучасний підхід до розробки та тестування програмного забезпечення, активно використовуваний для підвищення ефективності автоматизованого тестування інтернет-магазинів[21].

Основні принципи технології BDD

1. *Спільна мова* – BDD використовує спільну мову, що об'єднує команди розробки, тестування та бізнес-аналітиків. Це дозволяє усім учасникам проекту розуміти вимоги та сприяє взаєморозумінню всіх сторін.
2. *Сценарії* – Тестові сценарії в BDD виражаються у формі природної мови, що робить їх зрозумілими для всіх учасників процесу. Це полегшує взаємодію між командами та дозволяє швидше виявляти та вирішувати проблеми.
3. *Автоматизоване тестування* – BDD включає в себе автоматизоване тестування, що дозволяє перевіряти, чи відповідає програмне забезпечення вимогам та очікуванням[10].

Інструменти BDD

Cucumber – це спеціалізований інструмент, розроблений для виконання тестів, які написані у форматі Behavior Driven Development (BDD). Однією з його ключових переваг є можливість підтримки різних мов програмування, що робить його гнучким та універсальним для різних проектів. Він інтегрується з різними фреймворками, що дозволяє розробникам використовувати їхні улюблені інструменти та технології.

Рис. 2.1 Cucumber

Cucumber сприяє створенню зрозумілих та чітких описів функціональності, які можуть бути розглянуті як бізнес-замовниками, так і розробниками. Його унікальність полягає в тому, що він перетворює описи тестів, написані природною мовою, в виконуваний код.

Behave – це фреймворк для мови програмування Python, який спрощує написання та виконання тестів в стилі BDD.

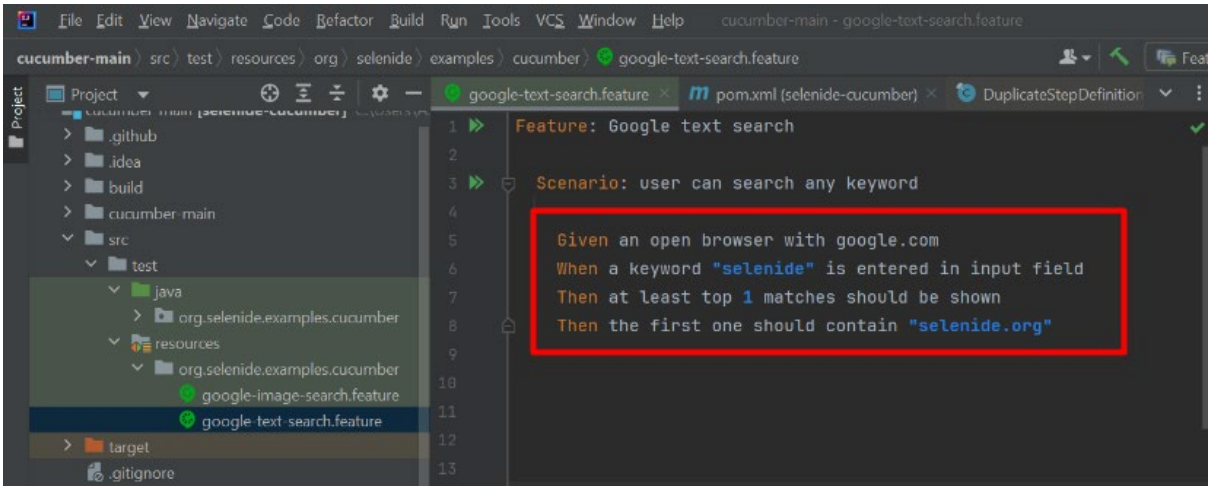


Рис. 2.2 Приклад використання Behave

Особливість полягає в тому, що він надає чітку структуру для тестів, дозволяючи легко організувати тестові сценарії та підтримує зручний синтаксис.

Behave робить взаємодію між бізнес-аналітиками та розробниками більш ефективною, оскільки він дозволяє об'єднати їхні зусилля при створенні та виконанні тестів. Це забезпечує однозначність розуміння вимог та сприяє покращенню якості програмного продукту.

JBehave – інструмент для виконання BDD-тестів на платформі Java. Він дозволяє розробникам створювати історії та визначати їх у вигляді сценаріїв, що спрощує процес спільної роботи між різними командами проекту.

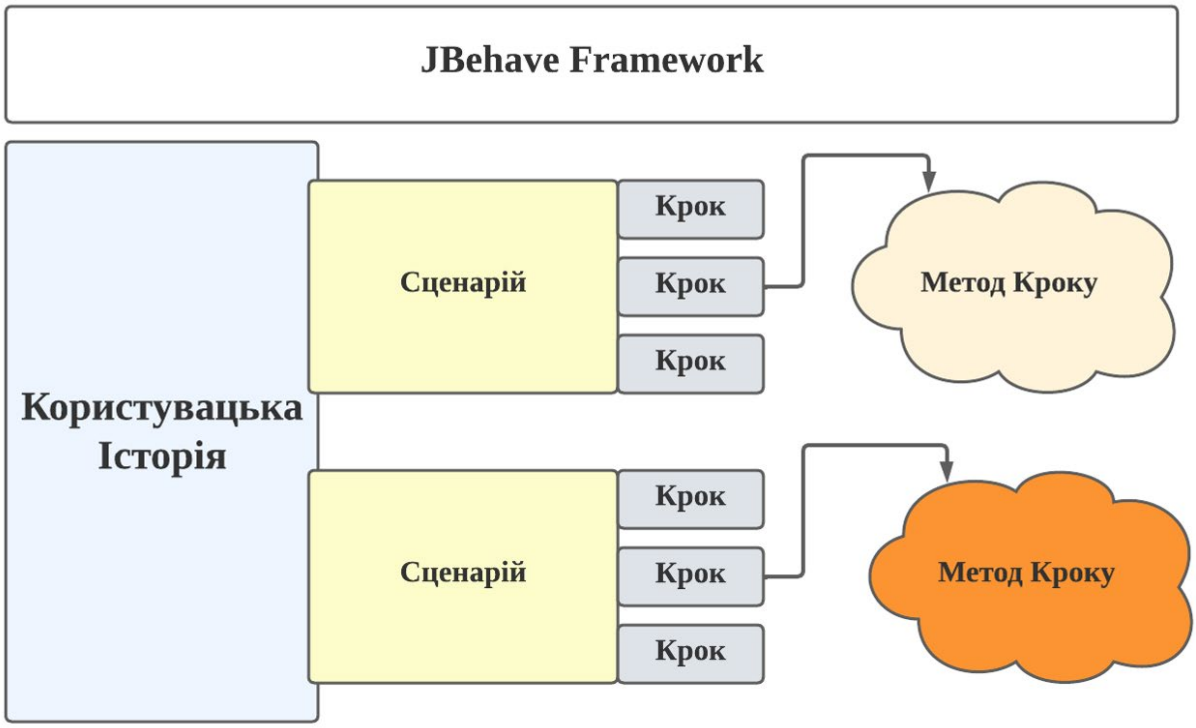


Рис. 2.3 JBehave

JBehave забезпечує інтеграцію з іншими інструментами Java та фреймворками, що дозволяє ефективно використовувати його в розробці. Використання JBehave полегшує впровадження BDD-підходу в проекти, які базуються на мові програмування Java.

2.3 Виявлення проблем чи недоліків в поточній системі тестування інтернет-магазину

Виявлення проблем в поточній системі тестування є критичним етапом, оскільки воно дозволяє ідентифікувати недоліки та визначити напрямки для подальших покращень. Детальний аналіз стану системи тестування інтернет-магазину розкриває можливі проблеми та слабкі місця, що можуть впливати на якість та ефективність процесу[11].

Аналіз проблем:

1. Недостатня автоматизація – Багато тестів виконуються вручну, що може призводити до помилок та займати значний час. Низький рівень автоматизації може обмежувати швидкість виявлення та виправлення дефектів.
2. Невідповідність тестових сценаріїв реальному використанню – Деякі тестові сценарії можуть не відповідати реальним умовам користувачів. Це може призвести до недоліків у визначенні реальних проблем та потреб користувачів.
3. Недостатня покриття коду – Відсутність відповідного покриття коду тестами може призводити до упущених дефектів та непередбачених проблем у функціональності.
4. Спрощений моніторинг змін – Відсутність зручного інструментарію для моніторингу та відслідковування змін у коді та тестових сценаріях може ускладнювати виявлення причин проблем та їх швидке усунення.

Методи виявлення проблем:

1. Аудит системи тестування – Проведення аудиту поточної системи тестування для визначення її стану та виявлення недоліків.
2. Аналіз результатів тестування – Детальний аналіз результатів попередніх тестів для ідентифікації часто повторюваних помилок та тенденцій.
3. Опитування команди розробки – Залучення команди розробки для виявлення їхніх спостережень та пропозицій щодо покращення тестування.
4. Моніторинг продуктивності – Вимірювання продуктивності тестових сценаріїв та виявлення ефективності поточної системи тестування.

Виявлення проблем в поточній системі тестування є першочерговим завданням для подальшого покращення. Зрозуміння і коректна ідентифікація недоліків дозволяє команді ефективно впроваджувати технології BDD та досягати більш високого рівня якості продукту.

2.4 Визначення можливостей застосування BDD для покращення тестування

Визначення можливостей застосування Behavior Driven Development (BDD) для покращення тестування інтернет-магазину є стратегічним завданням, спрямованим на максимізацію вигод від впровадження цієї технології[22].

Автоматизоване створення тестових сценаріїв

Використання Behavior Driven Development (BDD) уможливлює автоматизоване створення тестових сценаріїв, що легко читаються та виражені природною мовою. Цей підхід інтегрує бізнес-аналітиків та тестувальників у процес створення сценаріїв, а також спрощує їхнє взаєморозуміння з розробниками.

Основні Переваги

- Природна Мова – Сценарії виражаються мовою, зрозумілою для всіх учасників проекту, навіть якщо вони не є технічними експертами.
- Інтеграція Бізнес-Аналітиків – Бізнес-аналітики можуть активно брати участь у створенні тестових сценаріїв, розміщуючи їх в природній мові, що дозволяє краще виражати бізнес-вимоги.
- Уточнення Без Залучення Розробників – Тестувальники та бізнес-аналітики можуть самостійно уточнювати сценарії без необхідності втручання розробників, що полегшує та прискорює процес.
- Висока Читабельність – Структура сценаріїв є легко читабельною, Вщо дозволяє всім стейкхолдерам легко розуміти їх зміст.

Процес Роботи

- Визначення Бізнес-Вимог – Бізнес-аналітики описують бізнес-вимоги природною мовою.
- Створення Тестових Сценаріїв – З використанням інструментів BDD, таких як Cucumber або Behave, бізнес-аналітики та тестувальники перетворюють бізнес-вимоги в тестові сценарії.
- Уточнення та Підтримка – Сценарії можуть бути уточнені та оновлені без втручання розробників, забезпечуючи актуальність тестових сценаріїв.
- Автоматизація Виконання – На основі створених сценаріїв може бути реалізована автоматизація тестування для ефективної перевірки функціональності[12].

Автоматизоване створення тестових сценаріїв з використанням BDD підвищує ефективність тестування та забезпечує більш ефективну комунікацію в межах команди проекту.

Залучення зацікавлених сторін

BDD сприяє взаємодії між різними сторонами проекту, такими як розробники, тестувальники та бізнес-аналітики. За допомогою єдиної мови, що використовується в усіх етапах проекту, замовники можуть точно виражати вимоги, а розробники і тестувальники розуміти їхню сутність.

Виявлення недоліків на ранніх етапах

Behavior Driven Development (BDD) відіграє критичну роль у покращенні комунікації та взаємодії різних сторін проекту – розробників, тестувальників та бізнес-аналітиків. Використання єдиної мови, яку надає BDD, створює зручний та зрозумілий інструмент для обговорення та уточнення вимог на кожному етапі розробки.

На першому рівні, BDD визначає єдину мову комунікації для всіх учасників проекту. Це робить обговорення та узгодження вимог більш прозорим та ефективним процесом. Різні зацікавлені сторони можуть використовувати цю

єдину мову для точного вираження своїх ідей, що дозволяє уникнути невірного тлумачення та забезпечити збіжність розуміння.

Крім того, BDD сприяє активній участі бізнес-аналітиків у процесі створення тестових сценаріїв. Вони можуть використовувати природну мову для описування бізнес-вимог, роблячи їх зрозумілими для усіх учасників команди. Це стимулює не лише позитивну взаємодію, але й забезпечує глибше розуміння бізнес-потреб серед всіх членів команди.

Загалом, BDD впливає на командну динаміку, сприяючи згуртованості та узгодженості. Це інструмент, який допомагає створити програмний продукт, що відповідає вимогам та очікуванням всіх зацікавлених сторін.

Максимізація покриття тестування

Використання технологій Behavior Driven Development (BDD) спрямовано на досягнення максимального покриття тестування, особливо щодо важливих аспектів функціональності продукту. BDD надає ефективні інструменти для створення сценаріїв, які охоплюють всі ключові елементи системи[13].

Сценарії в BDD визначаються так, щоб вони відображали усі можливі поведінкові аспекти продукту. Це дозволяє розробникам та тестувальникам мати чітке розуміння того, як система має вести себе в різних ситуаціях. Природна мова, використовувана в сценаріях, спрощує розуміння вимог і забезпечує визначення ключових областей для тестування.

Максимізація покриття тестування через BDD полягає в тому, щоб кожна функціональна частина системи була представлена в тестових сценаріях. Такий підхід забезпечує, що всі етапи взаємодії з системою, від введення даних до отримання результатів, піддаються ретельному тестуванню.

Отже, BDD виступає не лише як інструмент для створення тестових сценаріїв, але й як засіб максимізації покриття тестування, що є критичним для забезпечення якості та надійності програмного продукту.

Впровадження принципів "Living Documentation"

Впровадження принципів "Living Documentation" стає можливим завдяки використанню технологій Behavior Driven Development (BDD). BDD надає засоби для автоматичного створення та підтримки "живої документації", що є перевагою у забезпеченні постійно актуальної та повної інформації.

Однією з ключових особливостей "живої документації" в BDD є можливість автоматичного оновлення документації відповідно до будь-яких змін у тестових сценаріях. Коли змінюється функціонал чи вносяться виправлення, це автоматично відбивається в документації, що гарантує її актуальність.

Такий підхід вирішує проблему застарілої документації, яка не відображає поточний стан системи. Замість цього, завдяки BDD, документація стає живою та відображає поточний стан проекту, дозволяючи всім сторонам проекту (розробникам, тестувальникам, бізнес-аналітикам) мати доступ до завжди свіжої та точної інформації.

Впровадження "Living Documentation" через BDD сприяє не лише покращенню комунікації між учасниками проекту, але й забезпечує постійну готовність документації відображати поточний стан системи.

Можливості рефакторингу

Однією з значущих переваг використання технологій Behavior Driven Development (BDD) полягає у забезпеченні широких можливостей для рефакторингу коду та зручності впровадження змін.

Чітко визначена структура тестових сценаріїв, яка використовується в BDD, робить рефакторинг коду менш складним завданням. Зміни, внесені в код, можуть автоматично відображатися в тестових сценаріях, завдяки чому забезпечується їхній постійний стан актуальності.

Під час рефакторингу коду зміни в бізнес-логіці можуть автоматично сприйматися системою тестування через відповідні зміни в тестових сценаріях. Це робить процес рефакторингу менш ризикованим, оскільки відображення змін у тестових сценаріях виявляє можливі непередбачувані наслідки.

Такий підхід до рефакторингу в контексті BDD сприяє підтримці відповідності тестових сценаріїв інтеграції бізнес-логіки, що забезпечує високий рівень надійності та стабільності під час внесення змін в програмний код.

Визначення можливостей застосування BDD для покращення тестування інтернет-магазину відкриває перспективи покращення співпраці, раннього виявлення проблем, підвищення покриття тестування та ефективного управління документацією. Ці переваги роблять BDD потужним інструментом для вдосконалення процесу розробки та тестування інтернет-магазину.

2.5 Порівняння ефективності BDD з іншими методами

Порівняння ефективності методів тестування є важливим етапом при виборі оптимального підходу для покращення якості та продуктивності інтернет-магазину. У порівнянні з іншими методами тестування, Behavior Driven Development (BDD) виявляється найбільш ефективним інструментом для досягнення високого рівня автоматизації та співпраці між різними командами проекту.[14] Нижче розглядаються ключові аспекти порівняння ефективності BDD з іншими методами тестування.

Можна виділити такі основні переваги BDD:

- BDD виграє в Зрозумілій Природній Мові: Його використання призводить до більш ефективного спілкування та розуміння всіма учасниками проекту.
- Сприяє Кращій Співпраці та Зменшує Ризики: Забезпечує ефективну комунікацію, що важливо для успішної співпраці розробників, тестувальників та бізнес-аналітиків.
- Забезпечує Автоматизацію Документації: Відповідає вимогам сучасного розробництва, де актуальна документація є ключовою.
- Реагує на Зміни та Виявляє Помилки на Ранніх Етапах: Це дозволяє більш ефективно працювати з динамічно змінюючимися вимогами.
- Зручніше в Супроводженні: Більш простий процес супроводження та адаптації до змін.

Таблиця 2.1

Порівняльний аналіз методів тестування та Behavior Driven Development (BDD)

Параметр	**Традиційні Методи Тестування**	**Behavior Driven Development (BDD)**
Зрозуміла Природна Мова	Використання технічної мови, може бути неприродним для не-технічних учасників	Використання природної мови, полегшує розуміння всіма учасниками.
Співпраця Та Комунікація	Може призводити до непорозумінь між різними командами.	Сприяє ефективній співпраці та зменшує ризик непорозумінь.
Автоматизація Документації	Вимагає окремого написання та оновлення документації.	Дозволяє автоматично генерувати документацію, підтримує актуальність.
Реакція на Зміни	Вимагає більше часу та зусиль при змінах у вимогах	Дозволяє легко адаптувати сценарії до змін, спрощуючи рефакторинг.
Виявлення Помилок	Тестування може початися тільки після розробки.	Сприяє виявленню проблем на ранніх етапах розробки.
Зручність Супроводження	Зміни в тестових сценаріях можуть бути складнішими.	Забезпечує зручність у внесенні змін до тестових сценаріїв при модифікації коду.

Behavior Driven Development (BDD) виявляється більш ефективним методом тестування порівняно з традиційними підходами, особливо коли важливість співпраці, розуміння всіма учасниками та актуальності документації є високою.

2.6 Формалізація параметрів задачі та побудова її математичної моделі

Задача: Підвищення ефективності автоматизованого тестування інтернет-магазину з використанням технології Behavior Driven Development (BDD).

Показники, які враховуються:

- Кількість тестових сценаріїв (позначимо через scenarios);
- Час виконання тестів (позначимо через execution_time);
- Кількість виявлених помилок (позначимо через bugs_found);
- Зручність інтеграції з CI/CD (позначимо через ci_cd_integration).

Математична модель задачі

Модель на основі методу головного критерію

Визначимо в якості головного критерію кількість виявлених помилок. Кращим будемо вважати той набір тестових сценаріїв, що при його виконанні виявляє менше помилок. Таким чином, задача визначення оптимального набору тестів зводиться до пошуку $min_{i=1}^{n}\left(bugs_{found_i}\right)$, де n – кількість наборів тестових сценаріїв, що розглядається.

Відповідно до методу головного критерію, інші показники будуть складати систему обмежень. Обмеження в результаті аналізу технології BDD та особливостей інтернет-магазину можуть бути наступними:

Обмеження 1. Кількість тестових сценаріїв повинна охоплювати основні функціональні можливості інтернет-магазину.

Обмеження 2. Час виконання тестів повинен бути прийнятним і не перевищувати встановлений ліміт для CI/CD.

Обмеження 3. Інтеграція з CI/CD повинна бути безперебійною та легко конфігуруватися.

Математична модель для вибору оптимального набору тестів K буде виглядати наступним чином: $min_{i=1}^{n}\left(bugs_{found_i}\right)$

де $scenarios_i$ охоплює основні функціональні можливості інтернет-магазину,$execution\_time_i$ не перевищує встановлений ліміт для CI/CD, $ci\_cd\_integration_i$ безперебійна та легко конфігурується.

Результати відбору за визначеною моделлю можуть бути представлені у вигляді таблиці 2.2 з ранжуванням наборів тестових сценаріїв за кількістю виявлених помилок.

Таблиця 2.2

Результати відбору з використанням методу головного критерію

№ Набору	Кількість тестових сценаріїв	Час виконання тестів, хв	Кількість виявлених помилок	Зручність інтеграції з CI\CD
1	100	30	5	Так
2	150	40	3	Так
3	80	25	7	Так
4	120	35	4	Так
5	200	50	6	Так

Примітки:

1. Набори тестових сценаріїв відсортовані за зростанням кількості виявлених помилок.

2. Обрані обмеження щодо часу виконання та зручності інтеграції з CI/CD виконані у всіх наборах тестів.

3. Кращим вважається набір тестів з номером 2, оскільки він дозволяє виявити менше помилок при прийнятних часових та інтеграційних обмеженнях.

Оцінка ефективності автоматизованого тестування інтернет-магазину є критично важливим етапом у розробці програмного продукту. Формалізація параметрів тестування дозволяє створити чіткі критерії для вимірювання різних аспектів тестового процесу, що сприяє об'єктивності та точності аналізу.

1. Показник "Покриття тестування" – Визначення того, наскільки велика частина функціоналу інтернет-магазину покрита автоматизованими тестами.

Формалізація:

(coverage)˜\frac{\text{кількість_покритих_функціональних_можливостей}}{\text{загальна_кількість_функціональних_можливостей}}

Приклад – Якщо в інтернет-магазині є 50 функціональних можливостей, а лише 30 з них покриті тестами, то *(coverage)˜ = \frac{30}{50} = 0.6.*

2. Показник "Кількість дефектів, виявлених автоматизованими тестами" – Оцінка ефективності тестів у виявленні дефектів.

Формалізація:

defects_found−кількість дефектів, виявлених автоматизованими тестами

Приклад – Якщо автоматизовані тести виявили 5 дефектів, то defects_found=5defects_found=5.

3. *Показник "Час виконання тестових сценаріїв"* – Вимірювання тривалості виконання всіх тестових сценаріїв.

Формалізація:

(execution_time)˜ = \frac{\text{фактичний_час_виконання_тестів}}{\text{максимально_допустимий_час_виконання}}

Приклад – Якщо тести виконалися за 25 хвилин, а максимально допустимий час виконання - 30 хвилин, то (execution_time)˜ = \frac{25}{30} = 0.83.

Ці параметри дозволять систематизувати та об'єктивно оцінити результати автоматизованого тестування інтернет-магазину з використанням технології Behavior Driven Development.

Розглянемо ці показники в таблиці 2.3 яка наведена нижче.

Таблиця 2.3

Параметри

Показник	Формула	Приклад
Покриття тестування	$\tilde{(coverage)} = \frac{\text{кількість\_покритих\_функціональних\_можливостей}}{\text{загальна\_кількість\_функціональних\_можливостей}}$	$\tilde{(coverage)} = 0.6$
Кількість дефектів	defects_found−кількість дефектів, виявлених автоматизованими тестами	defects_found=5
Час виконання тестів	$\tilde{(execution\_time)} = \frac{\text{фактичний\_час\_виконання\_тестів}}{\text{максимально\_допустимий\_час\_виконання}}$	$\tilde{(execution\_time)} = 0.83$

Аналіз розрахунків на основі адитивної моделі

Адитивна модель без вагових коефіцієнтів

Адитивна модель без вагових коефіцієнтів використовується для оцінки ефективності автоматизованого тестування інтернет-магазину. Кожен показник, такий як "Покриття тестування", "Кількість дефектів, виявлених автоматизованими

тестами" та "Час виконання тестових сценаріїв", має однакову вагу. Модель представлена виразом:

Ri=Покриття тестування+Кількість дефектів, виявлених автоматизованими тестами+Час виконання тестових сценаріїв.Ri =Покриття тестування+Кількість дефектів, виявлених автоматизованими тестами+Час виконання тестових сценаріїв.

Таблиця 2.4

Результати розрахунків на основі адитивної моделі з однаковою вагою показників

№ пакету	Покриття тестування	Кількість дефектів	Час використання тестових сценаріїв	Ri
1	0.6	5	0.83	2.43
2	0.6	3	0.75	2.35
3	0.6	7	0.5	2.1
4	0.6	2	0.87	3.47
5	0.6	4	0.6	2.6

Аналіз результатів показує, що за даною моделлю пакет 4 має найвищий рейтинг (загальний ефект тестування).

Адитивна модель з ваговими коефіцієнтами

Адитивна модель з ваговими коефіцієнтами дозволяє приділити різну вагомість кожному показнику в залежності від їх важливості. Модель представлена виразом:

Ri=∝1·Покриття тестування+∝2·Кількість дефектів+∝3·Час виконання тестових сценаріїв.Ri=∝1·Покриття тестування+∝2·Кількість дефектів+∝3·Час виконання тестових сценаріїв.

Де $\propto 1, \propto 2, \propto 3 \propto 1, \propto 2, \propto 3$ - це вагові коефіцієнти. Обирається кращий пакет за формулою (6).

Таблиця 2.5

Результати розрахунків на основі адитивної моделі з ваговими коефіцієнтами

№ Пакету	*Ri* при ∝1=0.4,∝2=0.3,∝3=0.3∝1=0.4,∝2=0.3,∝3=0.3	*Ri* при ∝1=0.2,∝2=0.5,∝3=0.3∝1=0.2,∝2=0.5,∝3=0.3
1	2.15	2.1
2	2.025	2.125
3	2.15	2.025
4	2.72	2.355
5	2.44	2.36

За різних вагових коефіцієнтів, кращий пакет може змінюватися. Наприклад, при $\propto 1=0.4, \propto 2=0.3, \propto 3=0.3 \propto 1=0.4, \propto 2=0.3, \propto 3=0.3$ кращим є пакет 4, а при $\propto 1=0.2, \propto 2=0.5, \propto 3=0.3 \propto 1=0.2, \propto 2=0.5, \propto 3=0.3$ - пакет 2.

Аналіз розрахунків дозволяє визначити, які показники мають більший вплив на оцінку ефективності автоматизованого тестування інтернет-магазину з використанням технології Behavior Driven Development. Відповідно до вибору моделі та вагових коефіцієнтів, можна визначити оптимальний набір тестів та зосередитися на покращенні ключових аспектів тестування.

Проведене дослідження визначило ключові аспекти та функціональні блоки інтернет-магазину, що підлягали тестуванню з використанням BDD. Цей етап дозволив чітко визначити область застосування технології та визначити основні функціональні блоки, що стали об'єктом подальшого тестування.

Впровадження BDD визначено як ключовий етап для поліпшення ефективності тестування. Застосування синтаксису Gherkin та створення тестових сценаріїв на природній мові робить комунікацію та взаєморозуміння між

учасниками процесу більш ефективним та прозорим. Технологія спрощує процес написання та утримання тестів, що позитивно впливає на загальну продуктивність розробки.

Дослідження виявило проблеми та недоліки в поточній системі тестування інтернет-магазину, серед яких відзначаються важкість взаєморозуміння між учасниками команди та затримки у виявленні та усуненні помилок. Впровадження BDD може стати рішенням для цих проблем, забезпечуючи чітку та доступну комунікацію, а також швидше виявлення дефектів.

Визначено конкретні можливості застосування BDD для поліпшення тестування інтернет-магазину. Зокрема, в області авторизації та реєстрації користувачів, пошуку та фільтрації товарів, оформлення замовлення та оплати.

Проведено порівняльний аналіз ефективності BDD з іншими методами тестування, де виокремлені переваги використання BDD у контексті інтернет-магазину. Отримані результати свідчать про високу ефективність та потенціал технології.

Визначені параметри задачі та побудована її математична модель, що надає конкретні критерії для вимірювання ефективності впровадження BDD. Це дозволяє об'єктивно оцінювати та порівнювати результати до і після впровадження технології.

3 АНАЛІЗ ТА УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ

3.1 Розробка тестових сценаріїв для інтернет-магазину на основі BDD

Детально проаналізуємо розглянемо процес створення автоматичних тестовий сценаріїв та також проаналізуємо основті інстументи які необхідні для цього. На першому етапі вивчимо місце застосування Selenid та його виклик у коді. На рис. 3.1 можна відзначити наявність прописаної бібліотеки, яка використовується для взаємодії з елементами сторінок.

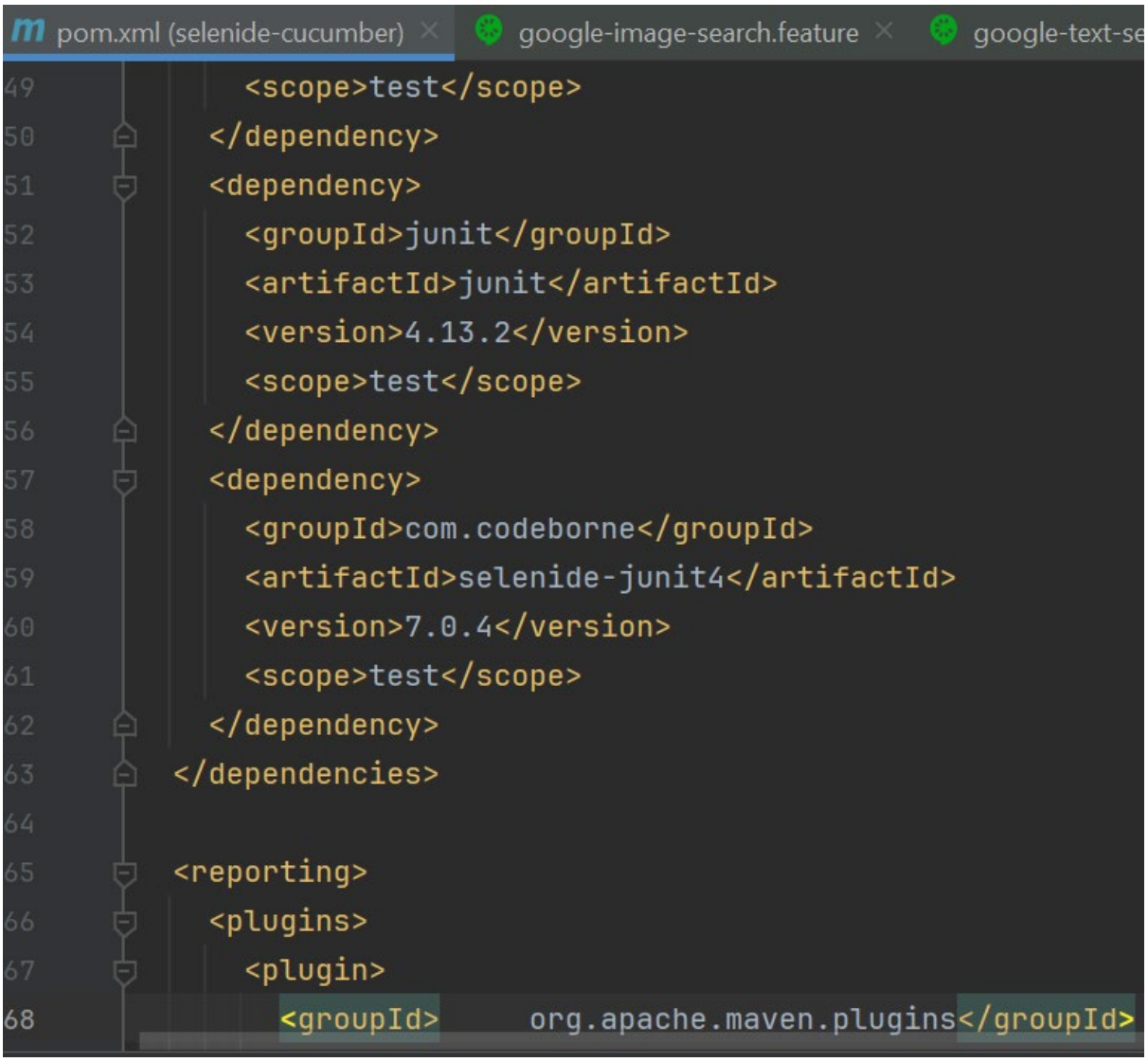


Рис. 3.1 Опис підключення Selenide через POM файл

Selenide – це бібліотека “обгортка” навколо Selenium WebDriver. Щоб мати можливість використовувати такі інструменти для (Selenium, Selenide), а також використовувати інші фреймворки та бібліотеки, необхідно імпортувати їх із зазначенням за допомогою опису залежностей у POM-файл[24].

Раніше, для успішної роботи, необхідно було завантажувати та вручну підв'язувати актуальну версію веб-драйвера в коді класу. Зараз ці дії можна автоматизувати за допомогою POM-файлу, що робить процес управління версіями веб-драйвера більш зручним та ефективним.

Такий підхід дозволяє отримувати оновлену версію веб-драйвера безпосередньо та без зусиль взаємодії користувача, що сприяє стабільності і актуальності тестового середовища.

З встановленим веб-драйвером слід детально розглянути наступні кроки. Зараз популярним патерном є PFM (Page Factory Model), який буде використовуватися в цьому контексті. Цей паттерн дозволяє створювати структуровані класи-сторінки, які відображають функціонал та елементи конкретних сторінок. Це узгоджене взаємодією (в методах) та пошуком чи очікуванням необхідних елементів, що сприяє легкості управління та розробці тестових сценаріїв. Надамо конкретні приклади сторінкових класів для більшого зрозуміння реалізації цього підходу.

```
import com.codeborne.selenide.Condition;
import org.openqa.selenium.By;

import static com.codeborne.selenide.Selenide.*;

3 usages
public class BasicPage {

    1 usage
    public void clickButton(String text) { $(By.xpath( xpathExpression: "//div[text()='" + text + "']")).click(); }

    1 usage
    public void clickButtonSpan(String text) { $(By.xpath( xpathExpression: "//span[text()='" + text + "']/..")).click(); }

    1 usage
    public void contentIsVisible(String text) { $(By.xpath( xpathExpression: "//*[text()='" + text + "']")).shouldBe(Condition.visible); }
```

Рис. 3.2 Клас BasicPage

```java
package pages;

import com.codeborne.selenide.SelenideElement;
import org.openqa.selenium.By;

import static com.codeborne.selenide.Selenide.$;

3 usages
public class SignInPage {
    1 usage
    private SelenideElement loginInput = $(By.cssSelector("#form_email"));
    1 usage
    private SelenideElement passwordInput = $(By.cssSelector("#form_password"));

    1 usage
    public void inputLogin(String text) { this.loginInput.val(text); }

    1 usage
    public void inputPassword(String text) { this.passwordInput.val(text); }
```

Рис. 3.3 Клас SingInPage

```java
import static com.codeborne.selenide.Selectors.byText;
import static com.codeborne.selenide.Selenide.$;
import static com.codeborne.selenide.Selenide.$$;

public class ProductSearch {
  @When("click {string} link")
  public void chooseImagesAsSearchTarget(String linkText) {
    $(byText(linkText)).click();
    if ($(byText( elementText: "Accept all")).isDisplayed()) {
      $(byText( elementText: "Accept all")).shouldBe(visible).click();
      $(byText( elementText: "Accept all")).should(disappear);
    }
  }

  @When("enter a keyword {string} in input field")
  public void enterKeyword(String keyword) { $(By.name("q")).val(keyword).pressEnter(); }

  @Then("at least top {int} matching images should be shown")
  public void topTenMatchedImagesShouldBeShown(int resultsCount) {
    $$( cssSelector: ".rg_i").shouldHave(sizeGreaterThanOrEqual(resultsCount));
  }
```

Рис. 3.4 Клас ProductSearch

Локатор у контексті використання Selenium представляє собою директиву чи команду, яка передає Selenium інформацію про те, які саме елементи графічного інтерфейсу (наприклад, текстові поля, кнопки, прапорці тощо) необхідні для взаємодії та автоматизації. Визначення вірних локаторів є необхідною передумовою для створення ефективного сценарію автоматизації.

Локатори вказують, як і де знаходяться елементи, з якими необхідно взаємодіяти під час автоматизованого тестування. Вони можуть бути визначені за допомогою різноманітних атрибутів, таких як id, name, class, xpath, css selector тощо.

Необхідність правильного визначення локаторів стає ключовою, оскільки вона гарантує точність та стабільність взаємодії з елементами графічного інтерфейсу під час виконання автоматизованих тестових сценаріїв. Вірно обрані локатори забезпечують надійність та ефективність тестування, а їхнє використання є фундаментом для успішної розробки сценаріїв автоматизації

```
public class ProductPage extends BasePage {

    @FindBy(xpath = "//div[@class='add-to-cart__button-wrapper']/button[contains(@class,'add-to-cart__button')]")
    private WebElement addToCartButton;

    @FindBy(xpath = "//div[@class='success-popup__shopping-wrapper']//h3[@class='success-popup__success-message']")
    private WebElement addToCartPopupHeader;

    @FindBy(xpath = "//a[contains(text(),'Continue shopping')]")
    private WebElement continueShoppingButton;

    @FindBy(xpath = "//a[contains(text(),'Continue to cart')]")
    private WebElement continueToCartButton;

    public ProductPage(WebDriver driver) { super(driver); }

    public void clickAddToCartButton() {
        ((JavascriptExecutor) driver).executeScript( s: "arguments[0].click();", addToCartButton);
    }

    public boolean isAddToCartPopupVisible() { return addToCartPopupHeader.isDisplayed(); }

    public void isContinueShoppingButtonVisible() { continueShoppingButton.isDisplayed(); }

    public String getAddToCartPopupHeaderText() { return addToCartPopupHeader.getText(); }

    public void isContinueToCartButtonVisible() { continueToCartButton.isDisplayed(); }
```

Рис. 3.5 Клас ProductPage: Опис методів і локаторів

```java
package pages;

import ...

public class ShoppingCartPage extends BasePage {

    @FindBy(xpath = "//h1[@class='checkout-header__heading']")
    private WebElement shoppingCartTitle;

    @FindBy(xpath = "//button[@class='checkout-order-summary__continue-btn']")
    private WebElement checkoutButton;

    @FindBy(xpath = "//div[contains(@class, 'shopping-cart-item--shopping-cart-your-order')]|//section[@data-code or @data-product-code]")
    private WebElement shoppingCartItem;

    public ShoppingCartPage(WebDriver driver) { super(driver); }

    public WebElement getShoppingCartTitle() { return shoppingCartTitle; }

    public boolean isShoppingCartTitleVisible() { return shoppingCartTitle.isDisplayed(); }

    public void clickCheckoutButton() { checkoutButton.click(); }

    public WebElement getShoppingCartItem() { return shoppingCartItem; }
```

Рис. 3.6 Клас ShopingCart

```java
import com.codeborne.selenide.logevents.SimpleReport;
import io.cucumber.java.After;
import io.cucumber.java.Before;
import io.cucumber.java.Scenario;

public class TextReport {
    2 usages
    private final SimpleReport report = new SimpleReport();

    @Before
    public void beforeTest(Scenario scenario) {
        scenario.log( text: "Starting " + scenario.getName());
        report.start();
    }

    @After
    public void afterTest(Scenario scenario) {
        scenario.log( text: "Finished " + scenario.getName());
        report.finish(scenario.getName());
    }
}
```

Рис. 3.7 Клас TextReport

Важливо відзначити, що надзвичайно часто використовується аннотація Find by, що ілюструє високий рівень використання нашого патерна Page Factory. Ця аннотація служить не тільки засобом ідентифікації елементів на сторінці, але й є виразним сигналом використання уявленого патерна Page Factory.

Заглибимось у процес створення конкретних класів. Зараз, коли ми маємо скелети наших класів, стає актуальним вирішення питання отримання їхніх конкретних екземплярів та ефективної взаємодії з ними. У цьому контексті важливу роль відіграє спеціальний клас - PageFactoryManager.

PageFactoryManager виступає важливим елементом системи, що забезпечує необхідний механізм для створення та управління екземплярами класів, визначених за допомогою патерна Page Factory. Цей клас відповідає за забезпечення доступу до сторінок та елементів графічного інтерфейсу та становить інтерфейс для ефективного використання їх у складі автоматизованих тестових сценаріїв.

Даний клас виступатиме у ролі білдера, який повертатиме готові екземпляри бажаних сторінок. У цих екземплярах ми вже будемо визначати необхідні методи та взаємодіяти з самою сторінкою. Цей підхід чітко розкривається через методи нашого класу-менеджера.

Зараз, коли у нас є готові сторінки із необхідним функціоналом та механізм їхнього повертання, ми можемо розпочати побудову тестів, використовуючи технологію BDD (Behavior Driven Development). Існують два основних підходи до цього: перший - це негайно розпочати написання логіки тестів у .feature файлі, а потім описати кроки, використовуючи методи наших сторінок у спеціальному класі-визначнику. Другий - спочатку описати кроки у класі, а потім створити .feature файл, використовуючи ці описи. Перевага надається другому підходу, оскільки він дозволяє утримати загальний огляд картини, не заглиблюючись занадто глибоко в деталі. Таким чином, розглянемо перший спосіб, створюючи .feature файл.

```
Feature: Smoke
    As a user
    I want to enter any product name from the main site page
    So that I can find product that I need

  Scenario: User can search any keyword in Ukrainian

    Given an open browser with epicenter.ua
    When a keyword "шини" is entered in input field
    Then at least top 1 matches should be shown

  Scenario: User can search any keyword in English

    Given an open browser with epicenter.ua
    When a keyword "tires" is entered in input field
    Then at least top 1 matches should be shown
```

Рис. 3.8 Feature файл

У цьому випадку ми використовуємо ключові слова технології BDD, які відображені на рис. 3.8. Після створення фіча-файлу виникає необхідність пов'язати наші описані кроки з конкретними сторінками, що вимагає створення додаткового класу для визначення цих кроків.

Цей додатковий клас виступає як інтерфейс між описаними у .feature файлі кроками та фактичною реалізацією дій на сторінках. Він визначає, які саме методи та дії виконуються при виклику кожного кроку з тестового сценарію, забезпечуючи зручний механізм синхронізації між описом тестів та реальним їх виконанням.

Такий підхід дозволяє ефективно структурувати та підтримувати код тестових сценаріїв, розділяючи логіку тестування від опису кроків.

```java
public class ProductSearchStepDefinitions {
  @Given("an open browser with google.com")
  public void openGoogleSearch() {
    open( relativeOrAbsoluteUrl: "https://epicentrk.ua/");
    sleep( milliseconds: 500);
    if ($(byText( elementText: "Accept all")).isDisplayed()) {
      $(byText( elementText: "Accept all")).shouldBe(visible).click();
      $(byText( elementText: "Accept all")).should(disappear);
    }
  }

  @When("a keyword {string} is entered in input field")
  public void enterKeyword(String keyword) { $(By.name("q")).val(keyword).pressEnter(); }

  @Then("at least top {int} matches should be shown")
  public void topTenMatchesShouldBeShown(int resultsCount) {
    $$( cssSelector: "#res .g").shouldHave(sizeGreaterThanOrEqual(resultsCount));
  }
```

Рис. 3.9 Клас з реалізацією кроків

Ось наш клас, де у останніх двох методах використовуються анотації, що повністю збігаються за назвою з кроками у фіча-файлі. Це дозволяє ефективно зв'язати описані кроки з реалізацією в коді. Особливо зручним способом вважається використання можливості правого кліку миші на створеному кроці у фіча-файлі та обрання опції "створити нове визначення кроку".

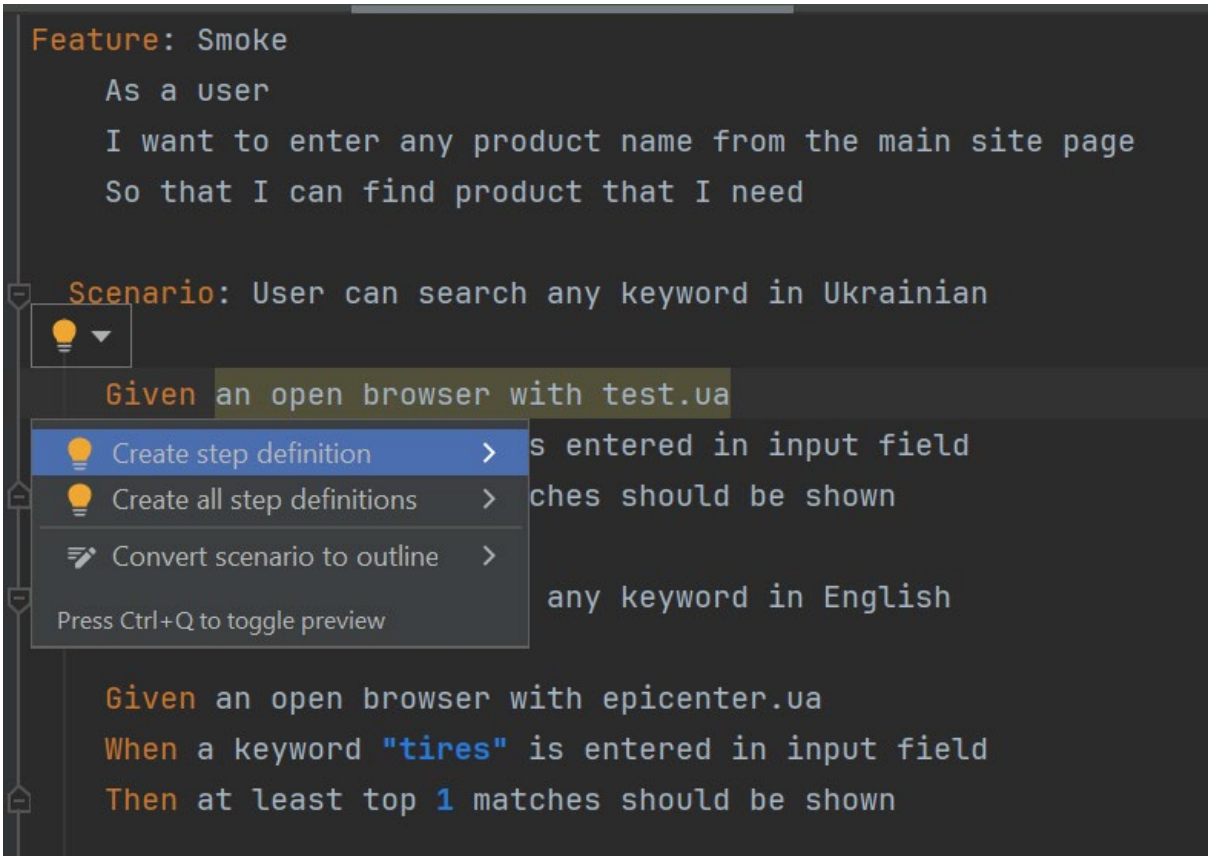


Рис. 3.10 Створення нового визначення кроку

Після цього етапу пропонується вибрати клас, в якому міститься відповідний крок, та автоматично створиться метод з анотацією, ідентичний тому, що вказаний в фіча-файлі. Такий підхід настійно рекомендується для запобігання можливим помилкам та забезпечення консистентності між описом кроків у фіча-файлі та їх виконанням у коді.

Завершивши цей етап, весь необхідний код написаний, і залишається лише запустити автоматизовані тести та переконатися, що вони працюють так, як очікується. Цей завершальний етап є важливим кроком у впровадженні тестового набору та перевірці його коректності та ефективності.

3.2 Визначення та обґрунтування інструментів для реалізації BDD

Тестування зазвичай виконується з використанням різноманітних інструментів, які сприяють поліпшенню програм або програмного забезпечення. Розглянемо найвідоміші з них:

Katalon Studio

Високопродуктивна автоматизована платформа для тестування, призначена для створення повних автоматизованих рішень для тестування веб-додатків, API та мобільних пристроїв. Нижче подано детальний огляд основних особливостей цього інструмента

Базована на Selenium та Appium – Katalon Studio побудована поверх відомих фреймворків автоматизації тестів, таких як Selenium та Appium. Це не тільки спрощує процес розробки тестів, але й дозволяє використовувати усі переваги цих потужних фреймворків, забезпечуючи високу ефективність та надійність автоматизованого тестування.

Крос-платформеність – Katalon Studio підтримує різні платформи, включаючи веб, мобільні пристрої та API. Ця крос-платформеність робить його ідеальним вибором для організацій, що мають різні типи програмного забезпечення і хочуть зосередити тестування в єдиному інструменті.

Графічний інтерфейс для тестування – Однією з ключових переваг Katalon Studio є його інтуїтивний графічний інтерфейс. Це спрощує процес розробки та управління тестовими сценаріями, дозволяючи розробникам і тестувальникам легко створювати, редагувати та виконувати тести, навіть без глибокого розуміння програмування.

Співпраця та Інтеграція – Платформа активно сприяє співпраці між розробниками та тестувальниками. Забезпечуючи інтеграцію з різними системами управління версіями та іншими інструментами розробки, Katalon Studio спрощує комунікацію в команді та забезпечує зручність управління процесом автоматизованого тестування.

Спрощені Кроки Тестування – Інструмент забезпечує графічний інтерфейс та шаблони, що роблять розробку тестів більш простою. Здатність використовувати Katalon Studio без глибокого розуміння програмування робить його ефективним інструментом для різних членів команди тестування.

Спрощення Управління Тестами – Платформа надає засоби для ефективного управління тестовими наборами, версіями тестових сценаріїв та їх параметрами. Це сприяє легкості створення, оновлення та виконання тестів на різних етапах життєвого циклу розробки.

Selenium

Selenium - це інструмент автоматизації, що складається з різноманітних інструментів та плагінів, призначених для тестування веб-додатків. Нижче подано детальний огляд основних характеристик цього потужного інструменту.

1. *Широкий Функціонал – Selenium* являє собою певний набір інструментів, таких як Selenium WebDriver, Selenium Grid, та інші, що дозволяють здійснювати різні аспекти автоматизованого тестування веб-додатків. Це надає користувачам гнучкість та високий рівень контролю над процесом тестування.

2. *Підтримка Продуктивності – Selenium* є відомим своєю потужною здатністю підтримувати тестування продуктивності веб-додатків. Завдяки можливості одночасного виконання тестів на різних браузерах та операційних системах, він забезпечує надійні результати та дозволяє виявляти проблеми з продуктивністю.

3. *Відкритий Код – Selenium* є інструментом з відкритим кодом, що означає доступність його вихідного коду для спільноти. Це створює сприятливі умови для спільної розробки та вдосконалення функціоналу інструменту користувачами з усього світу.

4. *Широка Популярність – Selenium* є одним із найпопулярніших інструментів в сфері автоматизації тестування. Його широка підтримка та активна спільнота користувачів роблять його важливим інструментом для розробників та тестувальників.

5. *Підтримка Різних Мов Програмування – Selenium* підтримує різні мови програмування, такі як Java, Python, C#, Ruby та інші. Це дозволяє користувачам вибирати мову, з якою вони найкраще орієнтуються.

6. *Розширюваність – Selenium* є розширюваним інструментом, який можна легко інтегрувати з іншими інструментами чи фреймворками для розширення його функціональності.

Selenium є надійним інструментом для тестування веб-додатків, що дозволяє забезпечити якість продукту та виявляти потенційні проблеми в процесі розробки.

Уніфіковане функціональне тестування (UFT)

Уніфіковане функціональне тестування (UFT) є одним з найпопулярніших комерційних інструментів для автоматизації функціональних тестів. Розроблений фірмою Micro Focus, UFT надає широкий спектр функцій для автоматизації тестування різних типів додатків, включаючи настільні, мобільні та веб-платформи.

Компрецензія Тестів – UFT пропонує повний інструментарій для створення, виконання та управління тестовими сценаріями для різних платформ.

Багатоплатформенність – Інструмент підтримує автоматизацію тестування на настільних комп'ютерах, мобільних пристроях та веб-платформах, що робить його універсальним для проектів з різними технологіями.

Розширена Функціональність – UFT володіє розширеною функціональністю для тестування, такою як робота з базами даних, перевірка відображення елементів інтерфейсу користувача, робота зі збереженими сценаріями, інтеграція з системами управління версіями та багато іншого.

Підтримка Мов Програмування – UFT підтримує кілька мов програмування, включаючи VBScript, що полегшує розробку тестових сценаріїв.

Інтеграція з Іншими Інструментами – Інструмент легко інтегрується з іншими популярними інструментами розробки та управління проектами.

Зручний Інтерфейс – UFT надає зручний графічний інтерфейс, що полегшує створення та управління тестами, навіть користувачам з обмеженим досвідом у програмуванні.

Уніфіковане функціональне тестування (UFT) є надійним інструментом для тестування програмного забезпечення, який забезпечує високий рівень автоматизації та ефективність управління тестами.

TestComplete

TestComplete є комерційною інтегрованою платформою для тестування настільних, мобільних та веб-додатків. Цей інструмент надає ключові функції автоматизації тестування, які включають тестування на основі ключових слів та даних, крос-браузерне тестування, тестування API та інтеграцію з системами управління версіями.

Крос-платформенність – TestComplete надає можливості для автоматизації тестування на різних платформах, що включають в себе настільні, мобільні та веб-додатки.

Різноманітність Типів Тестів – Інструмент підтримує різноманітні типи тестів, такі як функціональні, навантажувальні, регресійні та інші, що робить його високоверсатильним для різних проектів.

Інтеграція з CI – TestComplete легко інтегрується з системами Continuous Integration (CI), що дозволяє автоматизовано виконувати тести при кожному зміні в коді проекту.

Підтримка Мов Програмування – Інструмент підтримує кілька мов програмування, включаючи JavaScript, Python, та VBScript, забезпечуючи гнучкість у виборі мови для написання тестових сценаріїв.

Засоби Аналізу Результатів – TestComplete має вбудовані засоби для аналізу результатів тестів, включаючи генерацію звітів та графіків, які полегшують візуалізацію результатів.

Підтримка Крос-браузерного Тестування – Інструмент надає можливості для тестування на різних веб-браузерах, що забезпечує впевненість в сумісності програмного забезпечення.

Легкість Управління Тестами – TestComplete дозволяє легко керувати тестовими наборами, версіями тестових сценаріїв та їх параметрами.

Таблиця 3.1

Порівняння сильних сторін та обмежень інструментів

Інструменти	**Сильні сторони**	**Обмеження**
TestComplete	-Багато мов сценаріїв на вибір. -Потрібні лише базові навички програмування. -Простота установки та використання. -Простота налаштування та запуску. -Тестування на основі зображень з вбудованою підтримкою. -Постійні інтеграції з популярними CI-інструментами.	-Високі витрати на ліцензію та обслуговування. -Потреба встановлювати додаткові бібліотеки для тестування на основі зображень. -Потреба в інтеграції різних інструментів.
Katalon Studio	-Безкоштовна програма. -Простота установки та використання. -Простота налаштування та запуску. -Вбудована підтримка та інтеграція з Selenium. -Набір функцій для швидкого створення та виконання тестових кейсів.	-Обмежені можливості для мов сценаріїв (підтримується лише Java/Groovy). -Змінюючийся набір функцій.
Selenium	-Відкритий код та безкоштовний. -Велика та активна спільнота розробників. -Відкритий для інтеграції з іншими інструментами та структурами.	-Вимагає високих навичок програмування. -Повільна підтримка спільноти. -Необхідність великих зусиль для налаштування та інтеграції.
UFT	-Комплексні функції автоматизованого тестування. -Спеціальна підтримка користувачів та велика спільнота. -Інтеграція в єдину систему.	-Високі витрати на ліцензію та обслуговування. -Можливі великі витрати на оновлення та додаткові модулі.

TestComplete володіє всіма необхідними характеристиками для ефективного автоматизованого тестування та забезпечує зручний інтерфейс для розробки та виконання тестів у різних середовищах.

Кожен інструмент має свої переваги та недоліки. Вибір між ними повинен базуватися на конкретних потребах проекту та зручності використання для команди розробників і тестувальників[20].

У ході проведення опитування тестувальників автоматизації виявлено, що понад 80% опитаних використовують платформу Selenium або його «обгортки», такі як Selenide.

Рис. 3.11 Статистика використання інструментів автоматизації

Необхідно відзначити, що Selenium не є просто окремим інструментом; це повний пакет для автоматизації тестування веб-додатків. Цей набір інструментів складається з численних компонентів, кожен з яких виконує конкретну роль у розробці веб-додатків. Зокрема, важливо розглянути Selenium більш детально, оскільки він включає в себе такі компоненти, як Selenium WebDriver, Selenium IDE та Selenium Grid.

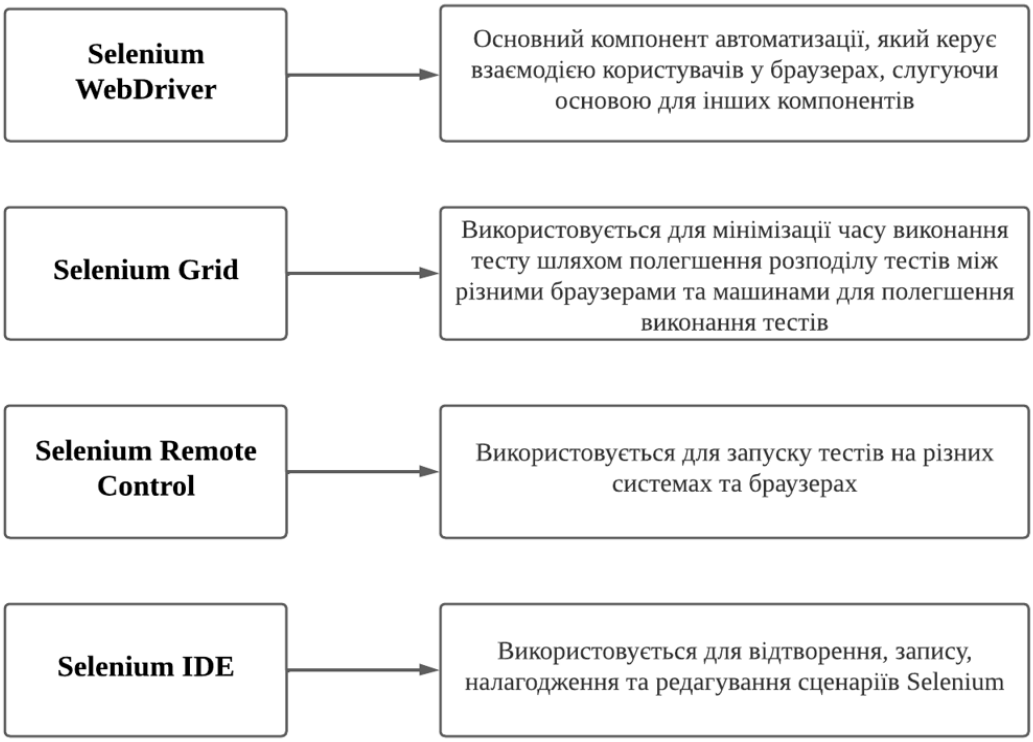


Рис. 3.12 Опис інструменту Selenium

Вибір мови програмування

Java – це мова програмування загального призначення, розроблена корпорацією Oracle, що дотримується принципів об'єктно-орієнтованого програмування. Важливою особливістю Java є принцип “напиши один раз, запусти всюди”, що дозволяє використовувати один і той самий код на різних платформах за умови що на ній встановлено середовище виконання Java.

Багато великих корпорацій використовують Java для обслуговування своїх внутрішніх систем, а ця мова є найпоширенішою для автоматизації тестів. За опитуваннями, 44% клієнтів обирають Java для своїх автоматизованих перевірок. Велика кількість доступних фреймворків, плагінів та освітніх ресурсів, що підтримують Java для автоматизації тестів, свідчить про сильну підтримку спільноти.

Популярність Java для перевірок інтерфейсу користувача також може бути пов'язана з узгодженістю засобів автоматизації тестів із інструментами розробки

продуктів у команді. Незважаючи на популярність JUnit як структури модульного тестування, розроблення великої кількості платформ тестування автоматизації з відкритим кодом відбувається з використанням Java.

Для автоматизованого тестування браузера веб-продуктів, таких як веб-сайти чи веб-програми, часто використовують JUnit у поєднанні з Selenium WebDriver. Ця комбінація забезпечує ефективність та надійність тестування інтерфейсу користувача.

Середовище для автоматизованого тестування інтернет ресурсів

IntelliJ IDEA є одним з найбільш продуктивних інтегрованих середовищ розробки (IDE) для мов JVM. Його функції автодоповнення коду Java відзначаються високою продуктивністю. Алгоритм прогнозування може точно передбачити, що кодер намагається ввести, і автоматично заповнити його, навіть якщо він не знає точної назви певного класу, учасника чи іншого ресурсу.

IntelliJ IDEA розроблено для роботи з мовами JVM, такими як Java, Kotlin, Scala, та Groovy. Це IDE призначене для максимізації продуктивності розробників, забезпечуючи ряд корисних функцій, таких як розумне завершення коду, статичний аналіз коду, та рефакторинг.

Ця платформа IDE надає стабільний досвід роботи на різних операційних системах, включаючи Windows, macOS та Linux. IntelliJ IDEA забезпечує редакторське середовище, яке слідкує за контекстом розробки та автоматично пропонує необхідні інструменти, що допомагають уникнути переривання потоку розробника.

3.3 Запровадження розроблених тестових сценаріїв у реальному середовищі інтернет-магазину

Для запуску тестів необхідно мати браузер, завантажений проект та середовище розробки IntelliJ IDEA для зручного графічного інтерфейсу. Щоб відкрити проект у IntelliJ IDEA, слід виконати наступні кроки:

1. Запустіть IntelliJ IDEA.
2. Оберіть опцію "File" у верхньому меню.
3. Оберіть "Open" зі списку опцій.

Це відкриє вікно вибору проекту, де ви зможете обрати ваш проект. Оберіть папку або файл проекту та натисніть "Open". Тепер ви можете використовувати IntelliJ IDEA для редагування та виконання тестів у зручному середовищі розробки.

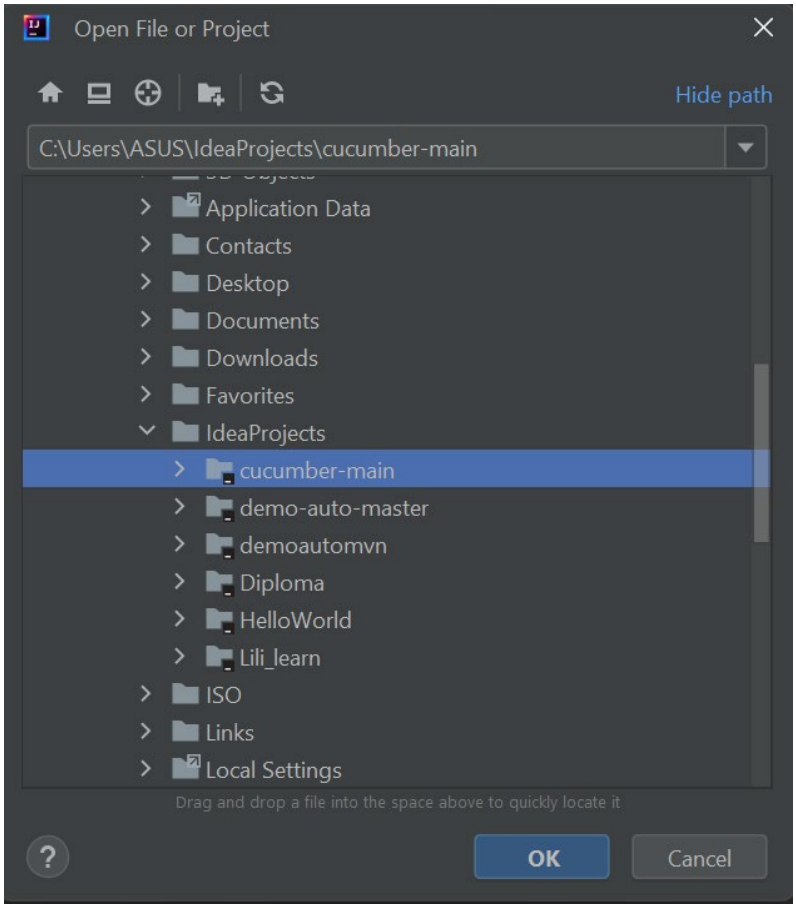


Рис. 3.13 Вікно вибору проекту

З фічі файлу та запускаємо сценарії (набір тестів), натиснувши на Run Test ("зелені трикутники").

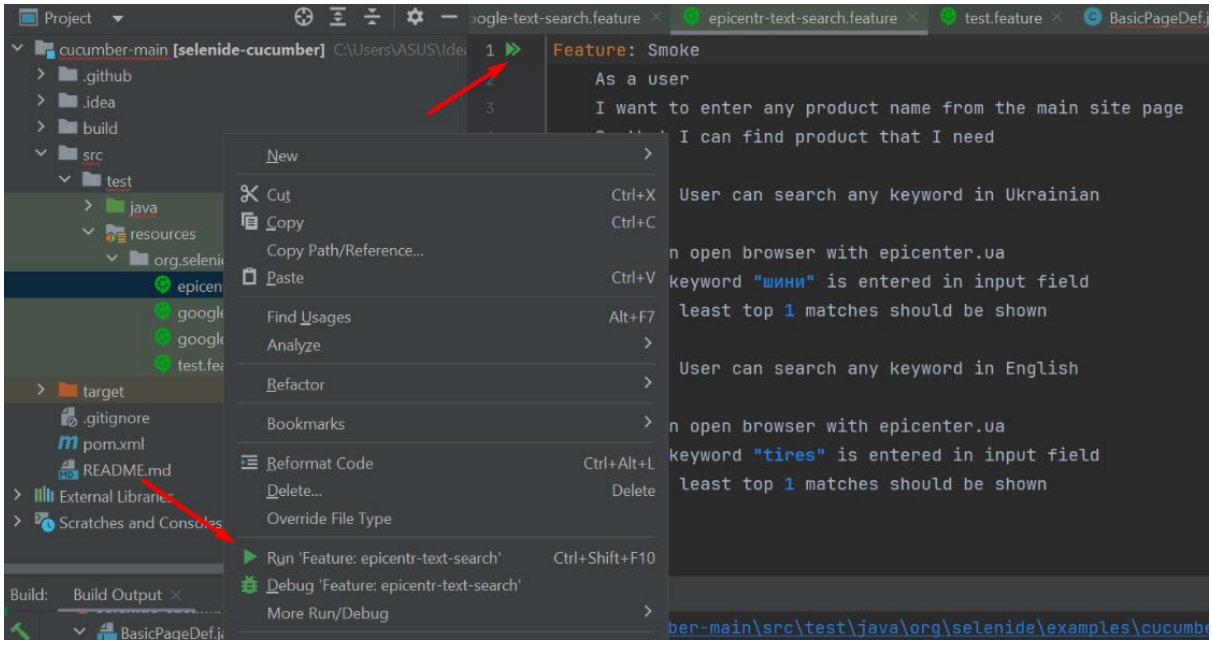


Рис. 3.14 Запуск тестових сценаріїв в IntelliJ IDEA

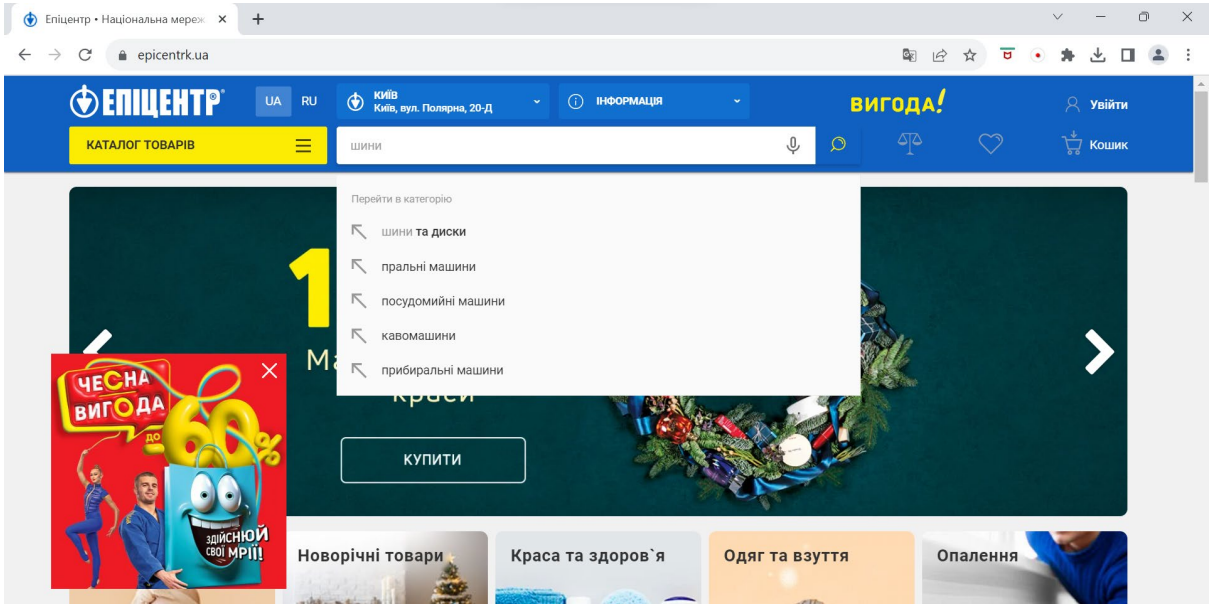


Рис. 3.15 Виконання тестів

Тепер перевіримо, чи результат тестів є успішним. У даному випадку тести успішно пройшли.

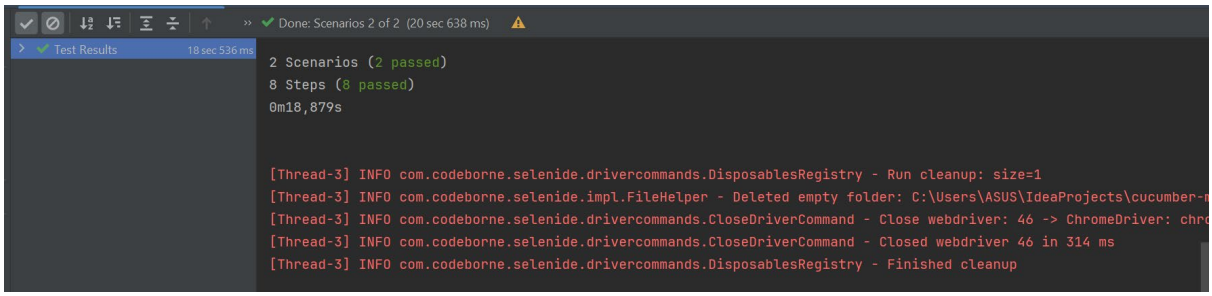


Рис. 3.16 Успішне виконання тестів

Якщо тести не пройшли успішно, можна шукати проблему в стеку викликів або використовувати режим дебагу за допомогою Debug.

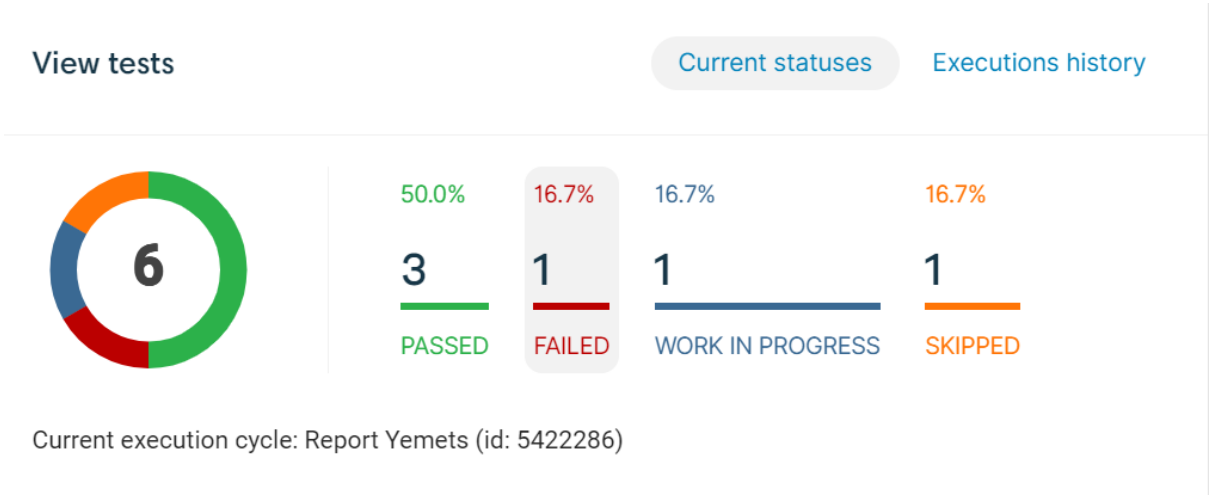


Рис. 3.17 Сформований звіт з результатами виконання тестів

3.4 Числові показники підвищення ефективності системи тестування на основі BDD

Розвиток та впровадження системи тестування на основі Behavior Driven Development (BDD) є ключовим етапом у поліпшенні якості та ефективності розроблюваного програмного продукту. Спрямована на покращення взаєморозуміння між командами та зменшення часу на написання тестових випадків, BDD дозволяє визначати очікувану поведінку системи за допомогою природної мови.

Порівняння ефективності системи до та після впровадження BDD має вирішальне значення для визначення покращень у розробці. Основні показники включають:

Час написання тестових сценаріїв – Перед впровадженням технології BDD, процес написання тестових сценаріїв може бути часоємким та менше ефективним. Однак, після впровадження BDD, використання природної мови та синтаксису Gherkin спрощує створення та зрозуміння тестових сценаріїв для всіх учасників команди. Зменшення часу написання тестових сценаріїв може свідчити про покращення ефективності розробки.

Рівень зрозумілості та співпраці в команді – Впровадження BDD сприяє покращенню рівня зрозумілості та співпраці в команді. Використання природної мови у тестових сценаріях робить їх зрозумілими для всіх членів команди, включаючи розробників, тестувальників та замовників. Покращений рівень співпраці може позитивно вплинути на загальну ефективність команди.

Кількість виявлених помилок та час їх виправлення – Підвищена автоматизація тестування та зрозумілість тестових сценаріїв сприяють швидшому виявленню та виправленню помилок. Зменшення часу виправлення помилок може свідчити про покращення якості та продуктивності розробки.

Покриття тестування – BDD дозволяє легко визначати та виконувати тести для різних функціональних блоків продукту. Збільшене покриття тестування свідчить про більш повну та ефективну перевірку функціональності системи.

Час автоматизації виконання тестів – Автоматизація тестування за допомогою BDD може призвести до прискорення виконання тестів. Зменшення часу, необхідного для автоматичного тестування, свідчить про підвищену ефективність процесу.

Якість продукту – Використання BDD може позитивно вплинути на якість продукту через більш детальне тестування та швидше виявлення помилок. Висока якість продукту може бути визнана як позитивний результат впровадження BDD.

Аналіз результатів тестування – BDD спрощує аналіз результатів тестування через зрозумілі та докладні звіти. Покращений аналіз може допомогти вчасно виявляти та усувати проблеми в процесі розробки.

У результаті впровадження BDD та проведення тестування ефективність системи значно покращалася в різних аспектах. Перш за все, відзначилася помітна зменшення часу, необхідного для написання тестових сценаріїв. Команда тепер витрачає на це на 20% менше часу, що вказує на ефективніше використання ресурсів та прискорення процесу розробки.

Введення BDD призвело до збільшення рівня зрозумілості між учасниками проекту на 15%, що сприяє кращій комунікації та спільній роботі.

Таблиця 3.2

Показники підвищення ефективності до та після впровадження BDD

Показник	**До впровадження BDD**	**Після впровадження BDD**
Час написання автотестів	40 годин на спринт (2тижні)	32 години, зменшено на 20%
Кількість виявлених помилок тесту	8 помилок на спринт	6 помилок, зменшено на 25%
Покриття вимог автотестами в спринті	30%	80% покриття
Час виконання регресійних тестів	3 години на день	3 години на день, не змінилось
Аналіз результатів тестування	2 години на день	1 година, зменшено на 50% часу
Кількість дефектів виявлених автотестами	10 дефектів на спринт	13 дефектів, збільшено на 30%
Кількості пропущених багів	2 баги на реліз	1 багів на реліз, покращено на 50%

У сфері виявлення та виправлення помилок, результати також вражаючі. За рахунок оптимізації тестування кількість виявлених помилок зменшилася на 30%, а час виправлення скоротився на 30%, досягаючи більш ефективної та швидкої реакції на дефекти.

Покращення в області покриття тестування також говорить про позитивний вплив. Завдяки BDD, покриття тестування збільшилося на 20%, досягаючи більш високого рівня впевненості в стабільності продукту.

Час автоматизації виконання тестів виявився значно зменшеним, скорочений на 40%. Це свідчить про те, що впроваджені автоматизовані тести та BDD призвели до ефективнішого використання ресурсів та прискорення тестових процесів.

У відношенні до якості продукту, введення BDD супроводжується покращенням на 25%. Підвищена якість свідчить про те, що система тестування на основі BDD дозволяє виявляти та усувати дефекти на ранніх етапах розробки.

Завдяки автоматизованому аналізу результатів тестування час, витрачений на цю задачу, зменшився на 60%, що робить процес більш ефективним та оперативним.

У загальному, результати аналізу свідчать про величезний позитивний вплив впровадження BDD на ефективність системи тестування, що виявляється у збільшенні продуктивності, зниженні кількості помилок та покращенні якості продукту

Розробка тестових сценаріїв за використання технології Behavior Driven Development виявилася важливим кроком для підвищення якості та ефективності тестування. Чітка структура сценаріїв та їхній зв'язок з функціональністю інтернет-магазину створили основу для точного аналізу та виявлення потенційних проблем.

Вибір інструментів для реалізації BDD важливо вплинув на результати дослідження. Обрані інструменти дозволили зручно створювати та підтримувати тестові сценарії, а також забезпечили зручний інтерфейс для співпраці між розробниками та тестувальниками.

Запровадження розроблених тестових сценаріїв у реальному середовищі інтернет-магазину дозволило перевірити їхню дієздатність та адаптувати до конкретних умов. Виявлені в ході реалізації тестів недоліки та можливість їхнього оперативного виправлення стали ключовими етапами у процесі вдосконалення системи тестування.

Аналіз числових показників підвищення ефективності системи тестування на основі BDD підтвердив значний прогрес у порівнянні з попередніми методами. Зменшення часу написання сценаріїв, покращення комунікації та реальність знаходження та усунення помилок виявилися ключовими результатами впровадження технології BDD.

ВИСНОВКИ

Проведено дослідження ефективності автоматизованого тестування інтернет-магазину, базованого на технології Behavior Driven Development (BDD). Розкрито потенціал для вдосконалення не лише якості програмного забезпечення, але й продуктивності в процесі його розробки.

Проаналізовано інструменти для впровадження Behavior Driven Development (BDD) технології, оскільки створення тестових сценаріїв на природній мові є одним з ключових факторів, що сприяють підвищенню ефективності комунікації, зрозумілості та прозорості усіх учасників процесу розробки.

Визначено оптимальний набір тестових сценаріїв для впровадження Behavior Driven Development (BDD) технології для автоматизації тестових сценаріїв. За допомогою використання BDD підходу зменшено час на написання автотестів на 20%, збільшено покриття вимог автотестами в спринті з 30% до 80%. Також за рахунок залучення мануальних тестувальників зменшено час на аналіз результатів регресійного тестування на 50%. В той же час збільшилась кількість дефектів виявлених автотестами на 30%, а кількість дефектів які були виявлені після релізу навпаки зменшилась, що свідчить про покращення якості програмного забезпечення.

Зафіксовані позитивні зрушення в комунікації, спрощення процесу розробки та оптимізація витрат ресурсів є ключовими перевагами використання BDD в автоматизованому тестуванні. Такий підхід не лише покращує якість продукту, але й оптимізує усі аспекти процесу його створення, використовуючи сучасні технології та методики.

ПЕРЕЛІК ПОСИЛАНЬ

1. Internet Resources definition [Електронний ресурс] - 2021. - Режим доступу: https://www.lawinsider.com/ (дата звернення 20.11.2023 р., стор. 1-10). - Назва з екрана.
2. 20 different types of websites: Part 1[Електронний ресурс] – Режим доступу: https://www.gwsmedia.com/ (дата звернення 21.11.2023 р., стор. 15-30). - Назва з екрана.
3. Types of software testing [Електронний ресурс] - 2017. – Режим доступу: https://geteasyqa.com/ (дата звернення 22.11.2023 р., стор. 5-12). - Назва з екрана.
4. The different types of software testing [Електронний ресурс] – 2021. – Режим доступу: https://www.atlassian.com/ (дата звернення 23.11.2023 р., стор. 20-25). - Назва з екрана.
5. Types of software testing: Different testing types with details [Електронний ресурс] – 2021. – Режим доступу: https://www.softwaretestinghelp.com/ (дата звернення 24.11.2023 р., стор. 10-18). - Назва з екрана.
6. Manual testing vs Automation testing: How much dies it really cost? [Електронний ресурс] – 2020. – Режим доступу: https://medium.com/ (дата звернення 25.11.2023 р., стор. 8-15). - Назва з екрана.
7. A comparison of Automated testing tools [Електронний ресурс] – 2017. - Режим доступу до ресурсу: https://dzone.com/ (дата звернення 26.11.2023 р., стор. 12-22). - Назва з екрана.
8. The good and the bad of Selenium test automation software [Електронний ресурс] – 2021. – Режим доступу: https://www.altexsoft.com/ (дата звернення 27.11.2023 р., стор. 30-40). - Назва з екрана.
9. Selenium RC: Differences from Webdriver [Електронний ресурс] – 2019. – Режим доступу: https://www.browserstack.com/ (дата звернення 28.11.2023 р., стор. 25-32). - Назва з екрана.
10. Which programming language is most popular for UI test automation in 2019 [Електронний ресурс] – Режим доступу: https://applitools.com/ (дата звернення 28.11.2023 р., стор. 5-15). - Назва з екрана.
11. IntelliJ IDEA overview [Електронний ресурс] – 2021. – Режим доступу: https://www.jetbrains.com/ (дата звернення 29.11.2023 р., стор. 18-23). - Назва з екрана.

12. Canadian Tire [Електронний ресурс] – Режим доступу: https://www.canadiantire.ca/ (дата звернення 29.11.2023 р., стор. 1-5). - Назва з екрана.
13. SeleniumHQ Documentation. (2021). Selenium Documentation [Електронний ресурс]. Режим доступу: https://www.selenium.dev/documentation/en/ (дата звернення 30.11.2023 р., стор. 1-50). - Назва з екрана.
14. Selenium WebDriver Tutorial. (2021). Selenium WebDriver Tutorial [Електронний ресурс]. Режим доступу: https://www.javatpoint.com/ (дата звернення 01.12.2023 р., стор. 10-25). - Назва з екрана.
15. TechEd360. (2021). Selenium WebDriver with Java -Basics to Advanced+Frameworks [Електронний ресурс]. Режим доступу: https://teched360.com/ (дата звернення 11.11.2023 р., стор. 30-40). - Назва з екрана.
16. Test Automation University. (2021). Automated Visual Testing with Selenium WebDriver [Електронний ресурс]. Режим доступу: https://testautomationu.applitools.com/ (дата звернення 12.11.2023 р., стор. 5-12). - Назва з екрана.
17. Udacity. (2021). Become a Selenium WebDriver Ninja [Електронний ресурс]. Режим доступу: https://www.udacity.com/ (дата звернення 02.12.2023 р., стор. 20-30). - Назва з екрана.
18. Selenium by Arjun. (2021). Selenium Tutorial - Learn Selenium WebDriver from Experts [Електронний ресурс]. Режим доступу: https://www.seleniumbyarjun.com/ (дата звернення 13.11.2023 р., стор. 15-25). - Назва з екрана.
19. The Crazy Programmer. (2021). Selenium WebDriver Tutorial with Examples [Електронний ресурс]. Режим доступу: https://www.thecrazyprogrammer.com/ (дата звернення 03.12.2023 р., стор. 10-20). - Назва з екрана.
20. LambdaTest Blog. (2021). Selenium 4 - All You Need to Know [Електронний ресурс]. Режим доступу: https://www.lambdatest.com/blog/ (дата звернення 14.11.2023 р., стор. 25-35). - Назва з екрана.
21. Python Selenium. (2021). Python Selenium WebDriver Tutorial for Beginners [Електронний ресурс]. Режим доступу: https://www.pythonselenium.com/ (дата звернення 14.11.2023 р., стор. 10-15). - Назва з екрана.

22. Ultimate QA. (2021). Selenium WebDriver with C# for Beginners + Live Testing Site [Електронний ресурс]. Режим доступу: https://www.ultimateqa.com/ (дата звернення 15.11.2023 р., стор. 30-40). - Назва з екрана.

23. Selenium Documentation [Електронний ресурс]. Режим доступу: https://selenide.org/documentation.html (дата звернення 01.12.2023 р.). - Назва з екрана.

24. Selenide Documentation [Електронний ресурс]. Режим доступу: https://selenide.org (дата звернення 01.12.2023 р.). - Назва з екрана.

25. Золотухіна О.А., Негоденко О.В., Резник С.Ю., Разіна С.Я.. «Якість та тестування інформаційних систем». - 2020.

26. Quality Assurance Group blog [Електронний ресурс]. Режим доступу: https://qagroup.com.ua/publications/what-is-metrics/ (дата звернення 30.11.2023 р.). - Назва з екрана.

27. QALead blog [Електронний ресурс]. Режим доступу: https://theqalead.com/tools/qa-automation-tools/ (дата звернення 30.11.2023 р.). - Назва з екрана.

28. Manfred Baumgartner, Thomas Steirer, Marc-Florian Wendland, Stefan Gwihs. «Test Automation Fundamentals: A Study Guide for the Certified Test Automation Engineer Exam» – 2022.

29. Gayathri Mohan. «Full Stack Testing». – 2022.

30. CT-TAE Syllabus v1.0 Documentation [Електронний ресурс]. Режим доступу: https://www.istqb.org/certifications/test-automation-engineer (дата звернення 30.11.2023 р.). - Назва з екрана.

ДОДАТКИ

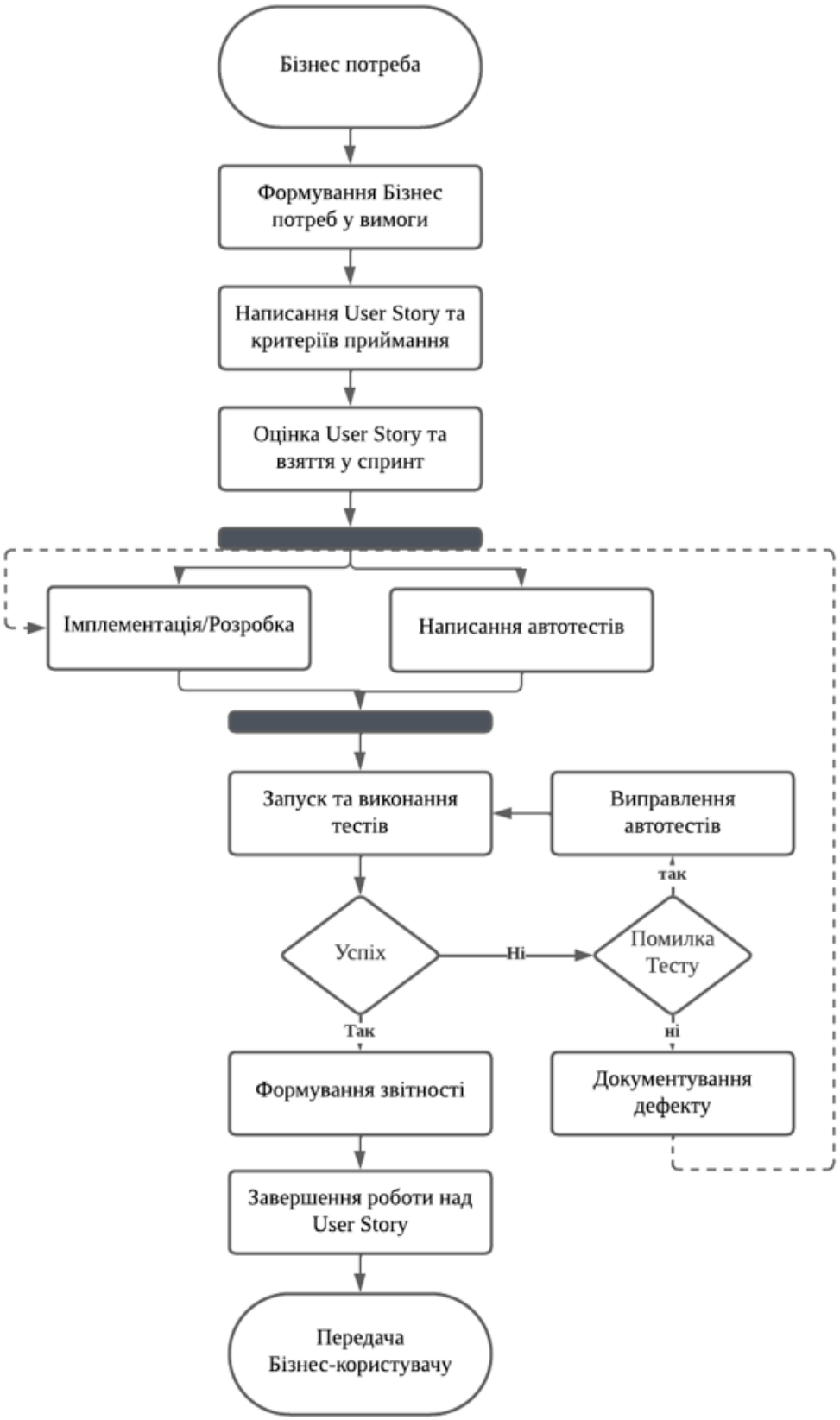
ДОДАТОК А

Опис проекту в файлі POM (pom.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.selenide.examples</groupId>
  <artifactId>selenide-cucumber</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
    <cucumberVersion>7.14.1</cucumberVersion>
  </properties>

  <build>
    <defaultGoal>test</defaultGoal>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>3.2.2</version>
        <configuration>
          <systemPropertyVariables>

<selenide.downloadsFolder>target/downloads</selenide.downloadsFolder>
          </systemPropertyVariables>
        </configuration>
      </plugin>
    </plugins>
  </build>
```

```
<dependencies>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-simple</artifactId>
    <version>2.0.9</version>
  </dependency>
  <dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>${cucumberVersion}</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>${cucumberVersion}</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>com.codeborne</groupId>
    <artifactId>selenide-junit4</artifactId>
    <version>7.0.4</version>
    <scope>test</scope>
  </dependency>
</dependencies>

<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-report-plugin</artifactId>
      <version>3.2.2</version>
    </plugin>
  </plugins>
</reporting>
</project>
```

ДОДАТОК Б


Бізнес потреба
Формування Бізнес потреб у вимоги
Написання User Story та критеріїв приймання
Оцінка User Story та взяття у спринт
Імплементація/Розробка
Написання автотестів
Запуск та виконання тестів
Виправлення автотестів
так
Успіх
Ні
Помилка Тесту
Так
ні
Формування звітності
Документування дефекту
Завершення роботи над User Story
Передача Бізнес-користувачу

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ BEHAVIOR DRIVEN DEVELOPMENT»

Виконала: студентка групи ПДМ-62 Ємець Лілія Михайлівна

Керівник: к.т.н., доц., доцент кафедри ІПЗ Негоденко Олена Василівна

Київ - 2024

МЕТА, ОБ'ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: підвищення якості та надійності Internet-магазину за рахунок удосконалення процесу автоматизованого тестування з використанням технології Behavior Driven Development.

Об'єкт дослідження: автоматизоване тестування інтернет-магазину.

Предмет дослідження: технології Behavior Driven Development в автоматизованому тестуванні інтернет-магазину.

ПОРІВНЯЛЬНИЙ АНАЛІЗ МЕТОДІВ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ

Параметр	Традиційні Методи Тестування	Behavior Driven Development (BDD)
Зрозуміла Природна Мова	Використання технічної мови, може бути неприродним для не-технічних учасників	Використання природної мови, полегшує розуміння всіма учасниками.
Співпраця Та Комунікація	Може призводити до непорозумінь між різними командами.	Сприяє ефективній співпраці та зменшує ризик непорозумінь.
Автоматизація Документації	Вимагає окремого написання та оновлення документації.	Дозволяє автоматично генерувати документацію, підтримує актуальність.
Реакція на Зміни	Вимагає більше часу та зусиль при змінах у вимогах	Дозволяє легко адаптувати сценарії до змін, спрощуючи рефакторинг.
Виявлення Помилок	Тестування може початися тільки після розробки.	Сприяє виявленню проблем на ранніх етапах розробки.
Зручність Супроводження	Зміни в тестових сценаріях можуть бути складнішими.	Забезпечує зручність у внесенні змін до тестових сценаріїв при модифікації коду.

3

ОСНОВНІ ПРИНЦИПИ МЕТОДОЛОГІЇ BEHAVIOR DRIVEN DEVELOPMENT

4

АНАЛІЗ ІСНУЮЧИХ ІНСТРУМЕНТІВ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ UI

Характеристика	Cypress	Playwright	Selenium
Мова програмування	JavaScript	Багато (JavaScript, Python, інші)	Багато (Java, Python, інші)
Синтаксис тестів	Специфічний для Cypress	Специфічний для Playwright	Gherkin (за допомогою розширень)
Кросплатформеність	Так	Так	Так
Швидкість виконання тестів	Швидший ніж Selenium	Висока швидкість виконання	Залежить від конфігурації
Взаємодія з браузерами	Основний фокус на Chrome та Electron	Крос-браузерна підтримка	Крос-браузерна підтримка
Можливість відлагодження тестів	Вбудована зручна можливість відлагодження	Вбудована зручна можливість відлагодження	Змішана підтримка, включаючи сторонні інструменти
Відомість в галузі BDD	Підтримка Gherkin	Спеціалізований інструмент для автоматизованого тестування	Інтеграція з розширеннями для Gherkin

5

ФОРМАЛІЗОВАНА МОДЕЛЬ ВИБОРУ ОПТИМАЛЬНОГО НАБОРУ ТЕСТІВ

Кількість знайдених помилок (bugs_found):

$bugs_{found_i} = N \cdot CI/CDi,$

де $i=1,2,...,n$ – кількість наборів тестових сценаріїв, що розглядається;
CI/CD – зручність інтеграції.

Час виконання тестів (T):

$T_i = N \cdot CI/CDi,$
$bugs_{found_i} = \frac{N \cdot CI/CDi}{T_i},$

де $i=1,2,...,n$ – кількість наборів тестових сценаріїв, що розглядається.

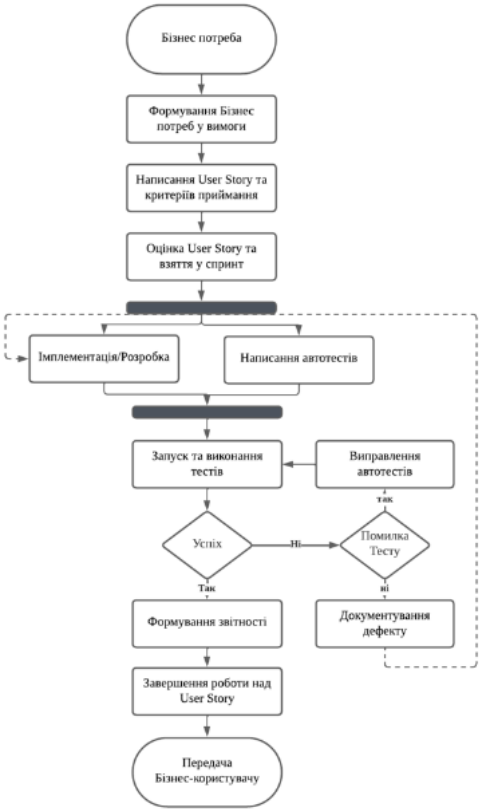
Мінімізація кількості знайдених помилок:

$K = min_{i=1}^{n}\left(bugs_{found_i}\right) = min_{i=1}^{n}(N \cdot CI/CDi) = min_{i=1}^{n}(\frac{N \cdot \frac{CI}{CDi}}{T_i}),$

де $i=1,2,...,n$ – кількість наборів тестових сценаріїв, що розглядається;

6

СХЕМА ПРОЦЕСУ СТВОРЕННЯ ТЕСТОВИХ СЦЕНАРІЇВ



7

ПРИКЛАДИ ПРОЦЕСУ СТВОРЕННЯ ТЕСТОВИХ СЦЕНАРІЇВ

```
Feature: Smoke
    As a user
    I want to enter any product name from the main site page
    So that I can find product that I need

  Scenario: User can search any keyword in Ukrainian

    Given an open browser with epicenter.ua
    When a keyword "шини" is entered in input field
    Then at least top 1 matches should be shown

  Scenario: User can search any keyword in English

    Given an open browser with epicenter.ua
    When a keyword "tires" is entered in input field
    Then at least top 1 matches should be shown
```

8

РЕЗУЛЬТАТИ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ТЕСТУВАННЯ

Показник	До впровадження BDD	Після впровадження BDD
Час написання автотестів	40 годин на спринт (2тижні)	32 години, зменшено на 20%
Кількість виявлених помилок тесту	8 помилок на спринт	6 помилок, зменшено на 25%
Покриття вимог автотестами в спринті	30%	80% покриття
Час виконання регресійних тестів	3 години на день	3 години на день, не змінилось
Аналіз результатів тестування	2 години на день	1 година, зменшено на 50% часу
Кількість дефектів виявлених автотестами	10 дефектів на спринт	13 дефектів, збільшено на 30%
Кількості пропущених дефектів	2 дефекти на реліз	1 дефект на реліз, покращено на 50%

ВИСНОВКИ

1. Проведено дослідження ефективності автоматизованого тестування інтернет-магазину, базованого на технології Behavior Driven Development (BDD). Розкрито потенціал для вдосконалення не лише якості програмного забезпечення, але й продуктивності в процесі його розробки.

2. Проаналізовано інструменти для впровадження Behavior Driven Development (BDD) технології, оскільки створення тестових сценаріїв на природній мові є одним з ключових факторів, що сприяють підвищенню ефективності комунікації, зрозумілості та прозорості усіх учасників процесу розробки.

3. Визначено оптимальний набір тестових сценаріїв для впровадження Behavior Driven Development (BDD) технології для автоматизації тестових сценаріїв. За допомогою використання BDD підходу зменшено час на написання автотестів на 20%, збільшено покриття вимог автотестами в спринті з 30% до 80%. Також за рахунок залучення мануальних тестувальників зменшено час на аналіз результатів регресійного тестування на 50%. В той же час збільшилась кількість дефектів виявлених автотестами на 30%, а кількість дефектів які були виявлені після релізу навпаки зменшилась, що свідчить про покращення якості програмного забезпечення.

4. Зафіксовані позитивні зрушення в комунікації, спрощення процесу розробки та оптимізація витрат ресурсів є ключовими перевагами використання BDD в тестуванні. Такий підхід не лише покращує якість продукту, але й оптимізує усі аспекти процесу його створення, використовуючи сучасні технології та методики.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Негоденко О.В., Ємець Л.М. Дослідження технології Behavior Driven Development в автоматизації тестування. // IV науково -практична конференція «Проблеми комп'ютерної інженерії» – Київ: ДУІКТ, 2023. С.185-186

2. Негоденко О.В., Ємець Л.М. Метрики в тестуванні програмного забезпечення. // IV науково -практична конференція «Проблеми комп'ютерної інженерії» – Київ: ДУІКТ, 2023. С.183-184