Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп’ютерних наук
(повна назва)

Кафедра Інформаційних управляючих систем
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів автоматизації тестування програмного забезпечення ІС

(тема)

Виконав:
здобувач 2 року навчання,
групи ІУСТм-23-1
Церковний Ілля Олександрович
(прізвище, ім’я, по батькові)

Спеціальність 122 Комп’ютерні науки
(код і повна назва спеціальності)

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

Освітня програма Інформаційні управляючі системи та технології
(повна назва освітньої програми)

Керівник: зав. каф. ІУС Петров К.Е.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ІУС _____ Петров К.Е.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Інформаційних управляючих систем

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

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

Освітня програма Інформаційні управляючі системи та технології
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
" 09 " грудня 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Церковному Іллі Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів автоматизації тестування програмного забезпечення ІС

затверджена наказом по університету від " 27 " листопада 2024 р. № 1249Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії " 18 " січня 2025 р.

3. Вихідні дані до роботи науково-технічні публікації та інтернет джерела з тематики кваліфікаційної роботи, матеріали передатестаційної практики, нормативна документація та стандарти з тестування програмного забезпечення

4. Перелік питань, що потрібно опрацювати у роботі провести аналіз предметної області та виконати постановку задачі дослідження; дослідити основні методі та інструменти автоматизованого тестування; визначити особливості створення та виконання тест-кейсів у системах автоматизації тестування; перевірка можливостей вдосконалення наявних методів автоматизації тестування на прикладі розробки нового методу; провести експериментальну перевірку ефективності розробленого методу, порівняти результати з існуючими аналогами

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області	09.12.2024 - 15.12.2024	Виконано
2	Постановка задачі дослідження	15.12.2024	Виконано
3	Дослідження існуючих рішень для створення та автоматизації тест-кейсів на основі BDD	16.12.2024 – 18.12.2024	Виконано
4	Формування вимог до методу створення тест-кейсів за допомогою UI інтерфейсу	19.12.2024 – 20.12.2024	Виконано
5	Реалізація методу для створення тест-кейсів	21.12.2024 – 24.12.2024	Виконано
6	Реалізація експорту тест-кейсів у BDD формат	25.12.2024 – 30.12.2024	Виконано
7	Порівняння запропонованого підходу з існуючими	31.12.2024	Виконано
8	Підготовка пояснювальної записки	01.01.2025 – 12.01.2025	Виконано
9	Підготовка презентації	13.01.2025	Виконано
10	Захист кваліфікаційної роботи	22.01.2025	Виконано

Дата видачі завдання 09 грудня 2024 р.

Здобувач _____
(підпис)

Керівник роботи _____ зав. каф. ІУС Петров К.Е.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 89 с., 6 рис., 2 дод., 23 джерел.

АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ, ІНФОРМАЦІЙНА СИСТЕМА, МЕТОДИ ТЕСТУВАННЯ, ПОБЛОКОВА РЕАЛІЗАЦІЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ТЕСТ-КЕЙС, BEHAVE-DRIVEN DEVELOPMENT.

Об'єктом дослідження є процес автоматизованого тестування програмного забезпечення (ПЗ).

Предметом дослідження є методи автоматизованого тестування програмного забезпечення інформаційних систем.

Метою дослідження є аналіз існуючих підходів до автоматизованого тестування, виявлення особливостей їх застосування та розробка власного методу автоматизованого тестування і надання рекомендацій для підвищення зручності й ефективності створення тест-кейсів з використанням методології Behavior-Driven Development (BDD).

Наукова новизна дослідження полягає розробці нових підходів до автоматизації тестування ПЗ, приділяючи увагу інтеграції методології BDD із підходами, орієнтованими на полегшення створення та підтримки тест-кейсів для нетехнічних спеціалістів. Дослідження спрямоване на розробку методу, який дозволяє забезпечити простоту, зручність і масштабованість процесу автоматизації тестування.

ABSTRACT

Master’s thesis: 89 pages, 6 figures, 2 appendices, 23 sources.

AUTOMATION, INFORMATION SYSTEM, TESTING METHODS, BLOCK-BASED IMPLEMENTATION, SOFTWARE, TEST CASE, BEHAVE-DRIVEN DEVELOPMENT.

The object of the research is the process of automated software testing.

The subject of the study is the methods of automated software testing of information systems.

The purpose of the research is to analyze existing approaches to automated testing, identify the features of their application, develop a proprietary method of automated testing, and provide recommendations for improving the convenience and efficiency of test case creation using the Behavior-Driven Development methodology.

The scientific novelty of the research lies in the development of new approaches to software testing automation, focusing on integrating the BDD methodology with approaches aimed at simplifying the creation and maintenance of test cases for non-technical specialists. The research aims to develop a method that ensures simplicity, convenience, and scalability in the process of testing automation.

ЗМІСТ

С.

Скорочення та умовні познаки 8

Вступ..... 9

1 Аналіз предметної області та постановка задачі дослідження..... 11

1.1 Аналіз процесу та основних підходів до тестування програмного забезпечення 11

1.2 Огляд інструментів та платформ для тестування 18

1.3 Аналіз технологій, що використовуються при тестуванні програмного забезпечення 20

1.4 Аналіз процесу створення тест-кейсів за допомогою BDD..... 22

1.5 Аналіз існуючих аналогів та рішень для впровадження BDD 25

1.6 Роль учасників команди в забезпеченні якості продукту 27

1.7 Постановка задачі дослідження..... 28

2 Методи автоматизованого тестування, що базуються на використанні BDD..... 31

2.1 Структура існуючих методів автоматизації тестування 31

2.2 Особливості інструментів для BDD тестування 32

2.3 Огляд поблокової реалізації створення тест-кейсу 34

3 Розробка методу автоматизації тестування на основі використання BDD..... 36

3.1 Структура та вимоги до створення та запуску тест-кейсів..... 36

3.2 Впровадження процесу створення та виконання тест-кейсів..... 45

3.2 Реалізація процесу створення тест-кейсу 52

3.2.1 Загальне розташування елементів сторінки 52

3.2.2 Реалізація валідації та збереження тест-кейсу..... 53

3.2.3 Реалізація експорту тест-кейсу під формат BDD 54

3.2.4 Реалізація обробки BDD тест-кейсів..... 56

4 Експериментальна перевірка та оцінка ефективності запропонованого методу 57
4.1 Основна мета і план проведення експерименту 57
4.2 Методологія проведення експерименту 59
4.3 Висновки за результатами проведеного експерименту 64
Висновки 66
Перелік джерел посилання 68
Додаток А Графічний матеріал кваліфікаційної роботи 71
Додаток Б Фрагменти коду 83
Б.1 Метод для конструювання інтерфейсу створення тест-кейсу 83
Б.2 Валідація та збереження тест-кейсу у репозиторій 85
Б.3 Експорт тест-кейсу під формат BDD 87
Б.4 Обробка BDD фреймворком тест-кейсів 88

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ПЗ – програмне забезпечення

BDD – Behavior-Driven Development – розробка, керована поведінкою

CI/CD – Continuous Integration / Continuous Delivery / Deployment – безперервна інтеграція та безперервна доставка/розгортання

DFD – Data Flow Diagram – діаграма потоків даних

IDE – Integrated Development Environment – інтегроване середовище розробки

IDEF – Integrated DEFinition – методологія функціонального моделювання

UI – User Interface – інтерфейс користувача

ВСТУП

Актуальність теми дослідження зумовлена стрімким зростанням складності сучасних програмних систем [1], які вимагають високої якості тестування для забезпечення стабільності, безпеки та функціональності. Зі збільшенням кількості розробників, аналітиків і тестувальників у командах виникає потреба у створенні інструментів, які дозволяють спростити процес розробки та автоматизації тест-кейсів [2]. Різні методології відкривають нові можливості для інтеграції бізнес-вимог у технічні процеси, однак відсутність зручних інтерфейсів і складність у використанні для нетехнічних спеціалістів залишаються суттєвими проблемами. У таких умовах стає критично важливим розробка інструменту із зручністю використання, адаптованого для різних рівнів технічної підготовки.

Об'єктом дослідження є процес автоматизованого тестування програмного забезпечення (ПЗ).

Предметом дослідження є методи автоматизованого тестування програмного забезпечення інформаційних систем.

Метою дослідження є аналіз існуючих підходів до автоматизованого тестування, виявлення особливостей їх застосування та розробка власного методу автоматизованого тестування і надання рекомендацій для підвищення зручності й ефективності створення тест-кейсів з використанням методології Behavior-Driven Development (BDD) [3].

Основна увага приділяється спрощенню процесу тестування ПЗ. Досягнення цієї мети спрямоване на підвищення доступності автоматизації тестування для команд із різнорівневою технічною підготовкою, оптимізацію процесів тестування та підвищення загальної якості програмного забезпечення.

Методика проведення дослідження передбачає виконання кількох ключових етапів. На початковому етапі виконується огляд і аналіз сучасних

методів тестування програмного забезпечення, включаючи BDD, а також інструментів, що використовуються для автоматизації тестування. Для досягнення поставленої мети застосовуються методи логічного, функціонального та системного аналізу, а також порівняння існуючих підходів за критеріями зручності, ефективності та доступності для нетехнічних спеціалістів.

У процесі розробки методу використовуються інженерні підходи до моделювання процесів, зокрема створення UML-діаграм (діаграми класів, станів, DFD), що дозволяє формалізувати структуру системи [4]. Крім того, виконується експериментальна перевірка ефективності розробленого рішення шляхом аналізу швидкості створення та виконання тест-кейсів у порівнянні з традиційними методами.

Наукова новизна дослідження полягає розробці нових підходів до автоматизації тестування програмного забезпечення. Особлива увага приділяється інтеграції методології BDD із підходами, орієнтованими на полегшення створення та підтримки тест-кейсів для нетехнічних спеціалістів. Дослідження спрямоване на розробку концепцій і методів, які дозволяють забезпечити простоту, зручність і масштабованість процесу автоматизації тестування.

Практична новизна полягає в можливості застосування запропонованих підходів у різних галузях розробки програмного забезпечення. Вони сприяють підвищенню доступності автоматизації тестування для широкого кола користувачів, включаючи бізнес-аналітиків та інших нетехнічних спеціалістів. Результати дослідження можуть бути основою для розробки нових інструментів і систем, здатних ефективно задовольняти потреби сучасного ринку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Аналіз процесу та основних підходів до тестування програмного забезпечення

Програмне забезпечення (ПЗ) займає провідне місце у сучасному світі, де практично кожна сфера діяльності людини залежить від його ефективності та надійності. У виробничих процесах, системах управління, фінансових операціях та навіть побутових пристроях роль програмного забезпечення є визначальною. Його безперебійна робота забезпечує стабільність і безпеку багатьох процесів.

Проте розвиток програмних систем супроводжується збільшенням їх складності, що створює значні ризики. Помилки у програмному забезпеченні не лише впливають на функціональність, але й можуть призвести до фінансових втрат, втрати репутації компанії або навіть небезпеки для життя користувачів. У випадках, коли програмний продукт є частиною критичних систем, таких як медичні пристрої або авіаційні системи, наслідки дефектів можуть бути катастрофічними.

Основною метою тестування є виявлення та усунення помилок на ранніх етапах розробки. Завдяки цьому процесу можна забезпечити відповідність програмного забезпечення вимогам замовника, стабільність роботи у реальних умовах, а також зменшити витрати на підтримку та доопрацювання продукту. Без належного тестування зростає ризик виникнення критичних помилок, які можуть стати причиною тривалих простоїв або значних збитків.

Забезпечення якості програмного забезпечення – це комплексний процес, який вимагає уваги на всіх етапах його життєвого циклу [5]. Особливу роль відіграє тестування, яке дозволяє перевіряти відповідність програмного продукту функціональним і нефункціональним вимогам. Воно має бути інтегроване на кожному етапі, починаючи з аналізу вимог і закінчуючи

підтримкою продукту після його впровадження.

В умовах глобалізації та цифровізації помилки у ПЗ стають більш помітними, оскільки сучасні системи часто інтегруються одна з одною, формуючи складні екосистеми. У таких умовах тестування стає критично важливим інструментом забезпечення стабільності та довговічності програмних продуктів. Зі зростанням потреб у швидкому випуску продуктів роль тестування лише збільшується, оскільки воно дозволяє не лише виявляти дефекти, але й запобігати їх виникненню завдяки виявленню слабких місць у дизайні системи або бізнес-логіці.

У світі, де конкуренція серед розробників ПЗ постійно зростає, якість продукту стає вирішальним фактором для залучення та утримання користувачів. Саме тому тестування є фундаментальною частиною розробки програмного забезпечення, яка дозволяє уникнути значних проблем у майбутньому.

Тестування ПЗ є важливим процесом, який дозволяє забезпечити його якість, надійність та відповідність заявленим вимогам. В умовах сучасного світу, де технології проникають у всі аспекти життя, тестування стало не просто бажаною, а обов’язковою складовою розробки. Воно спрямоване на те, щоб виявити недоліки й помилки в роботі програми до її випуску, що допомагає уникнути можливих фінансових втрат і ризиків для користувачів.

Процес тестування охоплює перевірку різних аспектів роботи системи. Це може бути аналіз функціональності, який оцінює, чи виконує продукт всі свої завдання, або перевірка продуктивності, щоб визначити, як система працює під час навантажень. Тестування також включає перевірку сумісності, аналіз користувацького досвіду та інші види перевірок, спрямовані на те, щоб зробити продукт максимально зручним і надійним [6].

Тестувальники займають особливу роль у цьому процесі. Вони є сполучною ланкою між командою розробників і бізнесом, забезпечуючи виконання всіх заявлених вимог. Їх завдання починаються з аналізу документації й вимог до продукту. На цьому етапі тестувальники мають

знайти потенційні прогалини, які можуть вплинути на функціональність або логіку системи.

Далі тестувальники створюють тест-кейси – детальні сценарії перевірок, які допомагають оцінити кожен елемент продукту. Виконання цих сценаріїв дозволяє знайти проблеми, які можуть не виявлятися під час звичайної розробки. У сучасних командах часто використовуються методи автоматизації, де тестувальники створюють спеціальні скрипти для перевірки повторюваних задач. Це значно економить час і підвищує точність процесу.

Крім технічної роботи, тестувальники активно комунікують із іншими учасниками процесу. Вони пояснюють знайдені помилки розробникам, уточнюють бізнес-вимоги з аналітиками та забезпечують, щоб кінцевий продукт відповідав потребам користувачів.

Важливо зазначити, що тестування – це не лише про пошук помилок. Це також про запобігання їх появі, аналіз логіки системи й удосконалення загального процесу розробки. Сучасні підходи, такі як BDD, підсилюють цей процес, інтегруючи тестування в ранні етапи розробки. Використовуючи прості й зрозумілі для всіх учасників команди формати опису сценаріїв, BDD дозволяє не лише знаходити помилки, а й створювати продукт, який максимально відповідає очікуванням клієнтів.

Тестування стає запорукою якості й стабільності в світі, де програмне забезпечення відіграє все більш значну роль. Завдяки ретельному тестуванню, розробники отримують впевненість у своєму продукті, бізнес – задоволених клієнтів, а користувачі – зручні й безпечні технології.

Програмне забезпечення є основою сучасного цифрового світу, впливаючи на всі аспекти нашого життя. Воно використовується для оптимізації бізнес-процесів, управління даними, автоматизації завдань, створення інтерактивних платформ для навчання, діагностики й лікування в медицині, а також в управлінні складними системами транспорту й логістики. З розвитком технологій роль ПЗ лише зростає, охоплюючи нові сфери, такі як штучний інтелект і доповнена реальність. Водночас підвищуються вимоги до

його функціональності, надійності, продуктивності та безпеки.

Тестування програмного забезпечення має багато напрямів, кожен з яких спрямований на перевірку конкретних аспектів роботи системи. Це дозволяє забезпечити всебічну перевірку якості продукту, виявити можливі дефекти та мінімізувати ризики.

Функціональне тестування є одним із ключових видів перевірок, метою якого є переконання, що програма виконує всі заявлені функції. Цей тип тестування перевіряє, чи відповідають результати роботи системи очікуванням користувачів, визначеним у технічній документації. Наприклад, функціональне тестування може включати перевірку коректності обробки даних, виконання запитів до бази даних, а також перевірку взаємодії різних компонентів системи [7].

Нефункціональне тестування, на відміну від функціонального, орієнтоване на аспекти, які не стосуються основної функціональності, але критично впливають на якість продукту. Наприклад, тестування продуктивності допомагає оцінити, як система поводиться під час високих навантажень, наскільки швидко вона реагує на запити користувачів і чи здатна обробляти великі обсяги даних. Безпека, сумісність і зручність використання також є частиною нефункціонального тестування.

Регресійне тестування є невід’ємною частиною циклу розробки програмного забезпечення. Воно полягає у перевірці, чи не порушили нові зміни у коді вже існуючий функціонал. Регресійне тестування дозволяє переконатися, що виправлення помилок або додавання нового функціоналу не вплинули негативно на стабільність системи. Цей тип тестування є особливо актуальним для проєктів із тривалим життєвим циклом, де оновлення випускаються регулярно.

Навантажувальне тестування спрямоване на визначення можливостей системи працювати у складних умовах, коли кількість користувачів або обсяг даних значно перевищує середні показники. Метою цього тестування є оцінка максимальної пропускної здатності системи, виявлення вузьких місць і

перевірка стабільності роботи під час екстремальних навантажень. Наприклад, таке тестування дозволяє визначити, скільки одночасних користувачів здатен обслуговувати сервер без втрати продуктивності.

Чорне, біле та сіре тестування розрізняються рівнем доступу тестувальника до внутрішньої структури системи. Чорне тестування базується на перевірці системи без знання її внутрішньої логіки. Тестувальники працюють як кінцеві користувачі, оцінюючи результат роботи системи за введеними даними. Біле тестування, навпаки, передбачає повний доступ до вихідного коду, алгоритмів і внутрішніх структур програми. Цей метод дозволяє більш глибоко перевірити правильність реалізації функціоналу. Сіре тестування поєднує обидва підходи, що дозволяє охопити більший спектр можливих проблем.

Тестування юзабіліті спрямоване на оцінку зручності використання програмного забезпечення. Воно дозволяє зрозуміти, наскільки інтуїтивно зрозумілим і зручним є інтерфейс, як легко користувачі можуть знайти потрібний функціонал і чи відповідає дизайн їхнім очікуванням. Це важливо для забезпечення позитивного користувацького досвіду та підвищення задоволеності продуктом.

Кожен із цих видів тестування має своє місце в процесі забезпечення якості програмного забезпечення. Їх правильне поєднання та адаптація до специфіки проєкту дозволяють гарантувати створення надійного, продуктивного та зручного для користувачів продукту. Якщо потрібно, я можу детальніше розширити будь-який із видів тестування або додати приклади їх використання.

Сучасні програмні системи є надзвичайно складними, оскільки вони повинні інтегруватися з іншими продуктами, працювати в умовах великого навантаження та відповідати строгим стандартам. Наприклад, у сфері фінансів програмне забезпечення повинно обробляти мільйони транзакцій щодня, гарантуючи їх безпеку та точність. У медицині – забезпечувати безпомилкову діагностику, зберігання та обробку чутливих даних пацієнтів. У таких умовах

навіть найменший дефект або збій може призвести до серйозних наслідків, таких як фінансові втрати, порушення довіри клієнтів, штрафи за невідповідність вимогам регуляторів або навіть загроза життю.

Тестування ПЗ дозволяє забезпечити його відповідність високим стандартам, мінімізуючи ризики та підвищуючи якість кінцевого продукту. Воно допомагає виявляти помилки на ранніх етапах розробки, коли їх виправлення є менш витратним. Також тестування дозволяє переконатися, що система працює стабільно навіть за екстремальних умов, таких як пікове навантаження чи несподівані сценарії використання. Це особливо важливо у світі, де швидкість і точність є важливими факторами успіху.

Одним із ключових підходів до забезпечення якості ПЗ є мануальне тестування [8]. Воно передбачає ручну перевірку роботи системи, що дозволяє тестувальникам зануритися у процес взаємодії з програмою, імітуючи дії кінцевого користувача. Цей підхід є надзвичайно важливим, оскільки дозволяє зосередитися на перевірці функціональності, зручності використання та естетичних аспектів інтерфейсу, які є критично важливими для позитивного користувацького досвіду.

Під час мануального тестування тестувальники можуть перевіряти кожну функцію програми окремо, оцінюючи її поведінку в різних умовах. Наприклад, вони тестують, як система реагує на коректні та некоректні дії користувача, як відображаються помилки, чи є логічна послідовність у роботі інтерфейсу та чи відповідає програма заявленим вимогам. Ця форма тестування дозволяє виявити деталі, які автоматизовані скрипти можуть проігнорувати, наприклад, незручності у використанні, логічні суперечності або елементи дизайну, які не відповідають стандартам.

Мануальне тестування є особливо корисним у ситуаціях, коли потрібно перевірити новий функціонал, який ще не можна автоматизувати через його нестабільність або недостатню зрілість. Воно також використовується для оцінки користувацького досвіду, де важливо зрозуміти, наскільки продукт є інтуїтивно зрозумілим і зручним для широкої аудиторії. Наприклад,

тестувальник може визначити, чи зручно розташовані кнопки, чи легко користувачам знаходити потрібні функції та чи відповідає дизайн очікуванням аудиторії.

Таким чином, мануальне тестування залишається незамінним інструментом для забезпечення якості програмного забезпечення. Воно дозволяє не лише виявляти помилки, а й надавати важливу інформацію для покращення продукту, роблячи його більш дружнім і зручним для користувачів.

Протилежністю мануального тестування є автоматизоване тестування, яке базується на використанні спеціалізованих скриптів, програм та інструментів для перевірки роботи програмного забезпечення. Цей підхід є незамінним для великих і складних проєктів, де необхідно виконувати повторювані завдання, такі як регресійне тестування, перевірка змін у коді чи масштабування системи. Автоматизація дозволяє значно зменшити час на перевірку, що особливо важливо в умовах швидкого випуску нових версій продукту.

Автоматизоване тестування дозволяє створювати точні, структуровані та легко відтворювані тести, які зберігаються для подальшого використання. Ці тести виконуються набагато швидше, ніж під час мануального тестування, і можуть охоплювати широкий спектр сценаріїв, включаючи негативні тести, стресові ситуації та перевірку рідкісних кейсів. Завдяки автоматизації можна ефективно проводити перевірки у великих масштабах, які неможливо виконати вручну через обмеження часу чи людських ресурсів.

Однією з ключових переваг автоматизованого тестування є зменшення впливу людського фактора. У той час як мануальні перевірки можуть залежати від концентрації чи досвіду тестувальника, автоматизовані скрипти виконуються послідовно, з точним дотриманням заданих сценаріїв. Це особливо важливо для проєктів із жорсткими вимогами до стабільності, де навіть найменші помилки можуть спричинити серйозні наслідки.

Цей підхід є надзвичайно корисним для виконання регресійного

тестування, коли необхідно переконатися, що нові зміни або функціонал не порушили вже існуючу систему. Завдяки автоматизації тестувальники можуть швидко перевірити сотні або тисячі сценаріїв, що охоплюють всі аспекти програми, включаючи інтеграцію різних компонентів, перевірку взаємодії із зовнішніми системами та тестування продуктивності.

Крім того, автоматизація дозволяє значно підвищити ефективність командної роботи. Розробники можуть запускати тестові скрипти на своїх локальних середовищах, отримуючи миттєвий зворотний зв’язок про наявність дефектів. Це сприяє швидшому виправленню помилок і зменшенню часу, витраченого на налагодження.

Таким чином, автоматизоване тестування стає основою для забезпечення якості у великих проєктах. Воно не тільки прискорює процес перевірки, але й підвищує її надійність, дозволяючи тестувальникам зосередитися на більш складних і творчих аспектах своєї роботи, таких як аналіз бізнес-логіки чи створення нових сценаріїв.

1.2 Огляд інструментів та платформ для тестування

Інструменти та платформи для тестування є невід'ємною частиною сучасного процесу забезпечення якості програмного забезпечення. Вони дозволяють автоматизувати перевірки, аналізувати результати, створювати звіти та підвищувати ефективність процесу тестування. Існує широкий спектр інструментів, які використовуються для різних цілей, таких як функціональне тестування, нефункціональне тестування, перевірка API, навантажувальне тестування тощо.

Можна виділити кілька ключових інструментів і платформ, які використовуються для забезпечення якості програмного забезпечення. Вони дозволяють автоматизувати перевірки, створювати звіти, оптимізувати процес

тестування та зменшувати ризики помилок.

Одним із найпоширеніших інструментів для функціонального тестування є Pytest, який забезпечує простоту та гнучкість у створенні тестів [9]. Цей інструмент підтримує модульне, інтеграційне та системне тестування, дозволяючи розробникам створювати тести з мінімальними зусиллями. Крім того, Pytest пропонує широкий вибір плагінів, таких як pytest-html для створення детальних звітів і pytest-benchmark для аналізу продуктивності.

Ще одним важливим інструментом є Selenium, який широко використовується для автоматизації тестування веб-додатків [10]. Його можливості інтегруються з Python, що дозволяє створювати гнучкі скрипти для перевірки веб-інтерфейсів. Selenium підтримує всі основні браузери та дозволяє імітувати дії користувача, такі як заповнення форм, натискання кнопок або навігація по сторінках.

Для забезпечення легкого входу в автоматизацію варто згадати Robot Framework, який базується на ключових словах [11]. Це робить його доступним не лише для технічних спеціалістів, а й для тих, хто не має досвіду програмування. Завдяки використанню бібліотек, таких як SeleniumLibrary, цей інструмент є чудовим вибором для тестування Python-додатків.

Для нефункціонального тестування також існує низка корисних інструментів. Наприклад, Locust дозволяє писати сценарії навантаження мовою Python. Він дає можливість змоделювати поведінку багатьох користувачів, що взаємодіють із системою одночасно. Це допомагає оцінити, як система справляється з великим навантаженням і чи є потенційні вузькі місця у її роботі.

Особливу увагу слід приділити інструментам для тестування API. Postman є одним із найпоширеніших рішень для цієї задачі [12]. Він дозволяє вручну та автоматизовано перевіряти API-запити й отримувати швидкий зворотний зв'язок про їх відповідність. Крім того, у Python можна використовувати спеціалізований інструмент PyRestTest, який дає змогу легко

тестувати REST API, забезпечуючи точність і надійність результатів.

Для управління процесами тестування можна використовувати такі платформи, як Allure Framework, яка дозволяє створювати детальні звіти про результати тестування [13]. Інтеграція з Pytest та іншими інструментами робить її зручною для аналізу. Крім того, TestRail забезпечує централізоване управління тест-кейсами, дозволяючи організовувати й відстежувати весь процес тестування, включаючи зв’язок із багатьма іншими системами, такими як Jira.

Інструменти для автоматизації, такі як Jenkins або GitLab Continuous Integration/Continuous Deployment (CI/CD), дозволяють інтегрувати тестування у процеси розробки та автоматизувати його запуск після кожного оновлення коду [14]. Це значно спрощує роботу команд, які працюють за Agile або DevOps методологіями, і забезпечує швидкий зворотний зв’язок.

Таким чином, інструменти для тестування ПЗ охоплюють різні аспекти забезпечення якості, допомагаючи оптимізувати процеси, підвищити продуктивність та мінімізувати ризики. Їх правильний вибір і використання є важливою складовою успішної розробки.

У процесі забезпечення якості програмного забезпечення часто використовуються пов’язані технології, які доповнюють тестування та дозволяють оптимізувати роботу команди. Ці технології забезпечують тісний зв’язок між бізнес-вимогами, розробкою та тестуванням, підвищуючи ефективність і прозорість процесу створення програмного продукту.

1.3 Аналіз технологій, що використовуються при тестуванні програмного забезпечення

Однією з найбільш популярних технологій тестування є BDD [15]. Цей підхід дозволяє інтегрувати тестування на ранніх етапах розробки та

забезпечити чітку комунікацію між технічними й нетехнічними членами команди. У BDD сценарії тестів описуються зрозумілою для всіх учасників природною мовою за допомогою синтаксису Given-When-Then. Це дозволяє бізнес-аналітикам, розробникам і тестувальникам працювати разом над створенням продукту, який точно відповідає вимогам замовника [16]. Інструменти, такі як Cucumber, Behave або pytest-bdd, спрощують автоматизацію таких сценаріїв і дозволяють інтегрувати їх у процес розробки.

Не менш важливим є підхід Test-Driven Development (TDD), який зосереджується на створенні тестів до початку написання самого коду. У TDD розробники спочатку пишуть тести, які описують бажану поведінку функціоналу, а вже потім реалізують сам код, щоб ці тести проходили. Цей підхід допомагає виявити помилки ще до того, як код стане частиною основної системи, і забезпечує високу якість програмного забезпечення. TDD підтримується багатьма інструментами для тестування, такими як Pytest, JUnit або TestNG.

Сучасні методології розробки, такі як DevOps, також нерозривно пов’язані з тестуванням. DevOps забезпечує інтеграцію процесів розробки та операцій, дозволяючи автоматизувати тестування, розгортання та моніторинг продукту. Використання пайплайнів CI/CD дозволяє запускати автоматизовані тести після кожної зміни в коді. Це забезпечує миттєвий зворотний зв’язок для розробників, дозволяючи швидко виправляти помилки. Інструменти, такі як Jenkins, GitLab CI/CD або CircleCI, забезпечують гнучкість і масштабованість таких процесів.

Також слід згадати про контейнеризацію та віртуалізацію, які відіграють важливу роль у сучасному тестуванні. Використання таких технологій, як Docker або Kubernetes, дозволяє створювати ізольовані середовища для тестування, що забезпечує стабільність і повторюваність результатів [17]. Це особливо корисно для тестування складних систем, які мають багато залежностей, або для симуляції реальних умов роботи програми.

Іншою технологією є спостереження за продуктивністю, яке дозволяє

моніторити стан і поведінку програмного забезпечення в реальному часі. Інструменти, такі як New Relic або Datadog, дають змогу тестувальникам і розробникам отримувати детальну інформацію про роботу системи, виявляти вузькі місця та запобігати проблемам ще до того, як вони стануть критичними [18].

Ці технології створюють фундамент для сучасного забезпечення якості програмного забезпечення. Вони не лише полегшують тестування, але й забезпечують більш прозорий, ефективний і адаптивний процес розробки, що відповідає потребам бізнесу та очікуванням користувачів.

1.4 Аналіз процесу створення тест-кейсів за допомогою BDD

Тест-кейс – це детальний набір інструкцій, що описують послідовність дій для перевірки конкретної функції або властивості програмного забезпечення. Він містить інформацію про вхідні дані, очікувані результати, а також кроки, які потрібно виконати для досягнення цих результатів. Тест-кейси допомагають забезпечити покриття всіх можливих сценаріїв і гарантують, що програмне забезпечення працює відповідно до вимог.

Тест-кейси є невід'ємною частиною процесу тестування програмного забезпечення, оскільки вони визначають, які дії потрібно виконати для перевірки конкретних функціональних можливостей або поведінки системи.

Тест-кейси відіграють ключову роль у процесі тестування програмного забезпечення з кількох причин.

По-перше, це систематичний підхід до тестування: тест-кейси дозволяють чітко визначити, що саме повинно бути протестовано, які кроки необхідно виконати, і які результати очікуються. Це допомагає зменшити ризик пропуску важливих сценаріїв під час тестування.

По-друге, це документування тестування: тест-кейси виконують роль

документації, яка описує, які саме функції та сценарії були протестовані. Це особливо важливо для великих проектів, де необхідно відслідковувати історію тестування та надавати інформацію про покриття тестами.

По-третє, це повторюваність тестування: тест-кейси дозволяють повторно проводити тестування при кожній зміні в програмному забезпеченні. Це особливо важливо для регресійного тестування, коли необхідно перевірити, що нові зміни не вплинули на вже працюючі функціональності.

І останнє, це узгодженість тестування: наявність чітких інструкцій у тест-кейсах забезпечує узгодженість результатів, оскільки різні тестувальники, використовуючи одні й ті ж тест-кейси, отримають однакові результати. Це дозволяє уникнути людського фактору та помилок, пов'язаних з різними інтерпретаціями.

Кожен тест-кейс має свої основні компоненти, що забезпечують його ефективність та ясність:

– ідентифікатор тест-кейсу: унікальний номер або код, який дозволяє легко знайти конкретний тест-кейс.

– назва тест-кейсу: короткий опис того, що саме тестується;

– передумови: умови, які повинні бути виконані перед початком тестування (наприклад, авторизація користувача);

– кроки виконання: детальний опис кожного кроку, який потрібно виконати під час тестування;

– вхідні дані: дані, які використовуються для виконання тесту;

– очікуваний результат: опис того, що повинно відбутися після виконання кожного кроку.

Існує кілька підходів до створення тест-кейсів, залежно від того, які цілі мають бути досягнуті та які ресурси є в наявності [19]. Розглянемо їх.

Першим підходом є створення ручних тест-кейсів: Цей підхід передбачає створення тест-кейсів тестувальником. Створення таких тест-кейсів дозволяє глибше вникнути у деталі роботи програмного забезпечення та створювати точні та детальні тест-кейси, що враховують специфічні вимоги

проекту. Проте, цей підхід може бути часозатратним, особливо для великих проектів із великою кількістю функціональностей.

Іншим підходом можна виділити автоматизоване створення тест-кейсів. Цей підхід використовує спеціальні інструменти, які генерують тест-кейси на основі коду або існуючих тестових сценаріїв. Цей підхід дозволяє зекономити час і ресурси, особливо при роботі з великими обсягами даних або складними сценаріями. Однак автоматизація не завжди дозволяє врахувати всі нюанси та специфіку проекту, тому часто поєднується з ручним підходом.

Останнім підходом є створення тест-кейсів для Behavior-Driver Development процесу. Використання підходу BDD дозволяє створювати тест-кейси у зрозумілому форматі, який описує поведінку системи з точки зору користувача. Тест-кейси, написані у форматі Given-When-Then, можуть бути легко зрозумілі не тільки розробникам, але й бізнес-аналітикам та іншим учасникам проекту. Це дозволяє покращити комунікацію в команді та забезпечити, що всі члени команди розуміють, що саме перевіряється.

Тест-кейси у форматі BDD зазвичай пишуться у вигляді сценаріїв у форматі Given-When-Then, що робить їх зрозумілими як для технічних фахівців, так і для бізнес-сторони проекту.

Можна зазначити наступні переваги використання підходу BDD:

– покращення комунікації між членами команди. Тест-кейси BDD описуються мовою, зрозумілою для розробників, тестувальників та бізнес-аналітиків, що дозволяє зменшити ризик неправильного розуміння вимог;

– залучення всіх учасників проекту до процесу тестування. Завдяки зрозумілому опису сценаріїв всі учасники можуть брати участь у створенні тестів, що підвищує якість покриття;

– легкість у підтримці. Тест-кейси BDD часто слугують як документація, що допомагає зберігати їх актуальність при внесенні змін у систему.

Недоліками же у підході BDD є наступні причини:

– необхідність навчання. Для впровадження BDD команда повинна

мати базові знання щодо написання тест-кейсів у форматі BDD та використовувати спеціальні інструменти, як-от Cucumber чи SpecFlow;

– початкові витрати на впровадження. Впровадження BDD потребує часу та ресурсів, щоб організувати процес та навчити команду.

Таким чином, можна зробити висновок, що найефективнішим підходом є використання тест-кейсів з фокусом на BDD. Це дозволяє забезпечити високу якість програмного забезпечення, залучаючи до процесу тестування всіх учасників проекту, та робить тест-кейси зрозумілими для кожного, незалежно від технічного рівня. Тест-кейси з BDD фокусом повинні стати ключовою складовою системи, щоб забезпечити стабільність, надійність і задоволення вимог користувачів.

1.5 Аналіз існуючих аналогів та рішень для впровадження BDD

Існує кілька популярних інструментів для реалізації BDD, таких як Cucumber, SpecFlow та PyTest, але всі вони мають певні обмеження, які ускладнюють їх використання для широкого кола користувачів, особливо для тих, хто не має глибоких технічних знань.

Здебільшого, інструменти для реалізації BDD є опен-сорс рішеннями, що робить їх доступними для багатьох команд. Наприклад, Cucumber є одним з найпопулярніших інструментів для створення BDD тест-кейсів. Він дозволяє описувати сценарії у зрозумілому форматі Given-When-Then, що полегшує комунікацію між бізнес-аналітиками, розробниками та тестувальниками. Інший приклад – SpecFlow, який інтегрується зі стеком .NET і дозволяє створювати BDD-тести у середовищі Microsoft. PyTest, в свою чергу, орієнтований на Python та забезпечує можливість описувати сценарії також у форматі Given-When-Then [20].

Попри свої переваги, ці інструменти є досить нішевими і часто

потребують додаткових знань та навичок для їх налаштування та використання. Наприклад, для роботи з Cucumber або SpecFlow необхідно мати знання відповідних мов програмування (Java, C#) і розуміти принципи автоматизації тестування. Це може стати бар'єром для впровадження BDD у команди, що не мають достатнього досвіду або знань у програмуванні.

Головною проблемою більшості існуючих рішень для реалізації BDD є відсутність зручного інтерфейсу користувача (UI), що ускладнює роботу для людей, які не мають глибоких технічних знань. Наприклад, бізнес-аналітики або менеджери проектів, які безпосередньо взаємодіють з вимогами, часто не мають достатніх технічних навичок для написання тест-кейсів за допомогою текстових редакторів або IDE. Відсутність інтуїтивно зрозумілого UI інтерфейсу обмежує можливості таких користувачів брати участь у процесі створення тестів.

Ця проблема призводить до того, що створення та підтримка тест-кейсів покладається виключно на технічних спеціалістів, що збільшує навантаження на команду розробників і тестувальників. Бізнес-аналітики і менеджери проектів не можуть повною мірою брати участь у створенні тестів, хоча саме вони найкраще розуміють вимоги та очікувану поведінку системи. Це створює розрив між бізнес-стороною і технічною командою, що може призводити до втрати важливих деталей або неправильного трактування вимог.

З огляду на аналіз існуючих рішень для реалізації BDD, можна зробити висновок, що основна проблема полягає у відсутності зручного інтерфейсу користувача, який би дозволяв нетехнічним спеціалістам брати активну участь у процесі створення тест-кейсів. Рішенням цієї проблеми може стати створення методу для побудови тест-кейсів на основі BDD за допомогою UI інтерфейсу, що дозволить значно спростити процес створення тестів, зробити його доступним для широкого кола користувачів та покращити комунікацію між бізнес-стороною і технічною командою. Такий підхід дозволить зробити тестування більш ефективним, знизити навантаження на технічних фахівців та забезпечити більш високу якість програмного забезпечення.

1.6 Роль учасників команди в забезпеченні якості продукту

У процесі розробки ПЗ якість продукту залежить від злагодженої роботи всієї команди, яка включає різні ролі [21]. У сучасних методологіях розробки, таких як Agile чи DevOps, залучення кожного члена команди до забезпечення якості є важливим аспектом для успішного створення продукту. Ролі в команді розподіляються залежно від їх функцій, але всі учасники можуть і повинні робити внесок у створення тест-кейсів і забезпечення якості на кожному етапі життєвого циклу розробки.

Менеджери проєктів часто відповідають за координацію процесу розробки, визначення пріоритетів і управління ресурсами. Їх участь у тестуванні може проявлятися через уточнення вимог і перевірку відповідності бізнес-цілей тест-кейсів. Бізнес-аналітики, своєю чергою, розробляють сценарії використання системи, які можуть бути основою для створення тест-кейсів у форматі BDD. Розробники інтегрують створені сценарії у кодову базу, забезпечуючи технічну реалізацію тестів, а тестувальники створюють деталізовані тести та перевіряють їх виконання.

Для досягнення ефективної співпраці між усіма цими ролями потрібен інтуїтивно зрозумілий інтерфейс, який дозволяє кожному члену команди долучитися до процесу створення та виконання тестів. Простий і зрозумілий інструмент дає змогу менеджерам і бізнес-аналітикам створювати базові тест-кейси без технічних знань, що розширює можливості команди та зменшує навантаження на технічних спеціалістів. Наприклад, використання інструментів, які підтримують формат Given-When-Then у BDD, дозволяє нетехнічним учасникам легко описувати сценарії, які розробники та тестувальники можуть автоматизувати.

Інтуїтивно зрозумілий інтерфейс також сприяє прозорості процесу тестування. Кожен член команди може бачити, які сценарії перевіряються, як вони пов’язані з бізнес-вимогами, і які результати було отримано. Це зменшує

кількість непорозумінь і сприяє швидшому виявленню та вирішенню проблем.

Залучення нетехнічних учасників до створення тестів і можливість прогоняти їх у реальному часі мають значні переваги. Це дозволяє забезпечити якість продукту ще на етапі його проектування, зменшити кількість помилок, які можуть виникнути через неправильне розуміння вимог, та пришвидшити час на впровадження змін.

Підхід BDD є оптимальним у таких умовах, оскільки його структура забезпечує зрозумілість для всіх учасників команди. Формат Given-When-Then дозволяє бізнес-аналітикам легко описувати поведінку системи, а розробникам і тестувальникам – швидко автоматизувати ці сценарії.

Таким чином, створення інтерфейсу, який дозволить залучати всіх членів команди, незалежно від їхнього технічного рівня, є критично важливим завданням. Він стане основою для прозорої, інтегрованої та ефективної співпраці в команді.

1.7 Постановка задачі дослідження

В ході аналізу підходів до автоматизованого тестування програмного забезпечення було виявлено кілька ключових проблем, які впливають на ефективність та доступність цього процесу:

– висока складність створення тест-кейсів для нетехнічних учасників команди;

– значне навантаження на технічних спеціалістів через відсутність зрозумілих для бізнесу інструментів;

– низький рівень залучення менеджменту та бізнес-аналітиків до процесу тестування;

– відсутність інтуїтивно зрозумілих інтерфейсів для роботи з методологією BDD, що ускладнює використання цього підходу.

Таким чином, враховуючи вищезазначені проблеми, можна визначити об'єкт, предмет та мету дослідження.

Об'єктом дослідження є процес автоматизованого тестування ПЗ.

Предметом дослідження є методи автоматизованого тестування ПЗ інформаційних систем.

Метою дослідження є аналіз існуючих підходів до автоматизованого тестування, виявлення особливостей їх застосування та розробка власного методу автоматизованого тестування і надання рекомендацій для підвищення зручності й ефективності створення тест-кейсів з використанням методології BDD.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– провести детальний аналіз методології BDD, її переваг та недоліків;

– дослідити процеси створення тест-кейсів та визначити ключові аспекти, які впливають на їх ефективність;

– розробити власний метод автоматизованого тестування, що базується на використанні методології BDD;

– визначити вимоги до інтерфейсу для створення тест-кейсів, який буде доступним для всіх членів команди;

– розробити архітектуру інструменту для створення тест-кейсів із використанням принципів BDD;

– провести експериментальну перевірку розробленого інструменту та оцінити його ефективність у порівнянні з традиційними підходами;

– сформулювати рекомендації щодо впровадження інструменту в реальні робочі процеси.

Основним практичним результатом роботи стане створення користувальницького застосунку для побудови тест-кейсів із використанням принципів BDD із проводженням порівняння ефективності створення автоматизованих тест-кейсів. Додаток повинен забезпечити можливість створення тест-кейсів через зрозумілий графічний інтерфейс, доступний як

для технічних, так і для нетехнічних фахівців, таких як бізнес-аналітики та менеджери.

Таким чином, основна задача дослідження полягає у розробці методу по автоматизації тест-кейсів через інструментарій, який дозволить учасникам команди різного рівня брати активну участь у процесі створення тест-кейсів. Це дозволить оптимізувати процес тестування, знизити навантаження на технічних фахівців та підвищити загальну якість кінцевого продукту.

2 МЕТОДИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ, ЩО БАЗУЮТЬСЯ НА ВИКОРИСТАННІ BDD

2.1 Структура існуючих методів автоматизації тестування

Тестування програмного забезпечення є невід'ємною частиною його життєвого циклу, забезпечуючи якість та надійність кінцевого продукту. Автоматизація цього процесу стала необхідністю для сучасних розробників, оскільки дозволяє підвищити ефективність, зменшити кількість людських помилок і пришвидшити випуск нових версій програмного забезпечення. Особливу увагу привертають методології, що інтегрують бізнес-вимоги з технічними аспектами розробки. Однією з таких методологій є BDD.

BDD є підходом до автоматизації тестування, який дозволяє забезпечити тісну співпрацю між бізнес-аналітиками, менеджерами, розробниками та тестувальниками. У його основі лежить ідея опису сценаріїв поведінки системи за допомогою природної мови, зрозумілої для всіх учасників команди. Використовуючи формат Given-When-Then, BDD дозволяє створювати зрозумілі тест-кейси, які описують конкретні умови, дії та очікувані результати.

Методи автоматизації тестування на основі BDD мають значні переваги. Вони сприяють прозорості процесу розробки, дозволяючи бізнесу та технічній команді спільно працювати над створенням продукту. Крім того, BDD інтегрується з сучасними практиками розробки, такими як Continuous Integration/Continuous Deployment (CI/CD), забезпечуючи швидке виконання автоматизованих тестів після кожної зміни в коді.

Проте, незважаючи на переваги, автоматизація тестування з використанням BDD також стикається з певними викликами. Існуючі інструменти часто мають складний інтерфейс, який ускладнює їх використання для нетехнічних учасників команди. Це створює потребу у розробці більш доступних та зручних рішень, які дозволять всім членам

команди брати участь у процесі тестування.

Таким чином, метою цього розділу є аналіз існуючих методів автоматизації тестування на основі BDD, розгляд їх переваг та недоліків, а також визначення шляхів вдосконалення цих підходів для розширення їх доступності та ефективності.

2.2 Особливості інструментів для BDD тестування

Автоматизація тестування на основі методології BDD значно спрощується завдяки використанню спеціалізованих інструментів. Вони забезпечують створення, виконання та аналіз тест-кейсів, написаних у форматі Given-When-Then. Серед найбільш популярних інструментів для BDD-тестування варто виділити Cucumber, Behave та pytest-bdd, причому останні два є чудовим вибором для команд, які працюють з Python.

Cucumber є одним із найпоширеніших інструментів для BDD. Він підтримує різні мови програмування, включаючи Java, Ruby, JavaScript і Python, що робить його універсальним для багатомовних команд. Основна перевага Cucumber – це його підтримка Gherkin, яка дозволяє писати тест-кейси у форматі Given-When-Then. Однак Cucumber має високий поріг входу для новачків і вимагає значних зусиль для інтеграції з існуючими середовищами розробки.

Behave – інструмент, який спеціально орієнтований на Python. Його зрозумілий синтаксис робить його зручним для команд, які працюють із цією мовою. Behave дозволяє швидко створювати та автоматизувати тест-кейси, забезпечуючи їх просту інтеграцію з пайплайнами CI/CD. Завдяки фокусу на Python цей інструмент є ідеальним вибором для невеликих і середніх команд, які потребують ефективного інструменту для BDD.

pytest-bdd є розширенням популярного фреймворку Pytest, який додає

підтримку BDD. Його перевага в глибокій інтеграції з Pytest, що дозволяє використовувати всі його можливості, такі як плагіни, параметризація тестів та аналіз продуктивності. Pytest-bdd забезпечує високий рівень продуктивності та гнучкості, а також легкий поріг входу для команд, знайомих із Pytest. Крім того, він дозволяє швидко інтегрувати BDD у вже існуючу Python-інфраструктуру, що робить його чудовим вибором для Python-орієнтованих проєктів.

Вибір відповідного інструменту для BDD-тестування залежить від специфіки проєкту та потреб команди. Для команд, які працюють із кількома мовами програмування, Cucumber може стати універсальним рішенням, але для Python-орієнтованих проєктів Behave і pytest-bdd є оптимальними варіантами. Обидва ці інструменти забезпечують високу продуктивність, простоту налаштування та підтримку ключових можливостей BDD.

Крім того, обрання Python-орієнтованих інструментів дозволяє легко інтегрувати тестування з популярними CI/CD системами, такими як Jenkins або GitLab CI/CD, що робить автоматизацію тестування швидшою та ефективнішою. У результаті команди можуть отримати прозорий і масштабований процес тестування, який забезпечує швидке виявлення та вирішення проблем.

Підсумовуючи, для автоматизації тестування на основі BDD Python-орієнтовані інструменти, такі як Behave та pytest-bdd, є найкращим вибором для проводження дослідження. Їхній синтаксис, гнучкість і легкість інтеграції забезпечують ефективну підтримку процесу тестування та відповідають потребам сучасних команд. Тому, подальші приклади з обгрунтуванням коду можуть бути створені за допомогою мови програмування Python з відповідними бібліотеками для роботи. Це забезпечить найкращу різницю у порівнянні з іншими способами автоматизації тест-кейсів.

2.3 Огляд поблокової реалізації створення тест-кейсу

Розробка інтуїтивно зрозумілого інтерфейсу для створення тест-кейсів є головним завданням у контексті автоматизації тестування на основі BDD як новітній метод. Основною метою такого методу є забезпечення можливості для учасників команди будь-якого рівня технічної підготовки брати участь у створенні, налаштуванні та виконанні тест-кейсів. Одним із найбільш перспективних підходів для досягнення цієї мети є використання ідеології Scratch [22].

Scratch є популярною платформою для візуального програмування, яка дозволяє створювати програми шляхом збирання блоків, що представляють собою окремі команди або логічні структури. Цей підхід, орієнтований на користувачів без глибоких знань програмування, забезпечує високий рівень доступності та інтуїтивності. Блоки з'єднуються між собою у логічній послідовності, що дозволяє візуально спостерігати за структурою програми.

Цей підхід можна адаптувати для створення тест-кейсів у контексті BDD. Наприклад, замість написання тестів у форматі Given-When-Then, користувачі можуть обирати готові блоки, що відповідають цим частинам сценарію. Кожен блок може мати інтуїтивно зрозумілий опис, наприклад:

- система знаходиться у стані X;
- користувач натискає кнопку Y;
- очікується відображення повідомлення Z.

Поблоковий підхід має кілька важливих переваг. По-перше, він забезпечує доступність для нетехнічних користувачів. Менеджери, бізнес-аналітики та інші члени команди без програмістських навичок можуть брати участь у створенні тест-кейсів. По-друге, цей підхід забезпечує прозорість, оскільки візуальне представлення тест-кейсів дозволяє чітко зрозуміти їх структуру та логіку. По-третє, поблоковий підхід спрощує модифікацію, адже окремі блоки можна легко замінити, додати чи видалити, що робить

редагування тестів зручнішим. Нарешті, модульність цього підходу дозволяє повторно використовувати блоки, створюючи бібліотеку стандартних дій і перевірок.

Традиційні текстові сценарії у BDD, хоча і зрозумілі більшості команд, вимагають певної дисципліни та уваги до синтаксису. Помилка в описі чи форматуванні може призвести до некоректної роботи тесту. Поблоковий підхід усуває ці ризики, оскільки користувачі працюють лише з попередньо налаштованими елементами. Крім того, блокова структура дозволяє уникнути дублювання коду. Наприклад, якщо є стандартна перевірка авторизації, її можна створити один раз у вигляді блоку й використовувати в різних тест-кейсах.

Підхід Scratch може бути інтегрований у спеціалізовані інструменти для BDD, створюючи можливість для команд візуально збирати тест-кейси. Наприклад, користувач може перетягувати блоки з бібліотеки в робоче поле, де формується повний сценарій. Після цього тест може автоматично конвертуватися у формат Given-When-Then для виконання за допомогою інструментів, таких як pytest-bdd або Behave. Цей підхід дозволяє значно підвищити ефективність процесу тестування, зменшити бар’єри для участі нетехнічних членів команди та забезпечити високу якість кінцевого продукту.

3 РОЗРОБКА МЕТОДУ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ НА ОСНОВІ ВИКОРИСТАННЯ BDD

3.1 Структура та вимоги до створення та запуску тест-кейсів

Забезпечення якості ПЗ є одним із найважливіших етапів його життєвого циклу. У сучасному світі, де технології інтегруються в усі сфери життя, від бізнесу до медицини, вимоги до надійності та функціональності програмних рішень постійно зростають. Критично важливим є не лише створення продукту, який працює відповідно до вимог, але й гарантія того, що він залишатиметься стабільним та ефективним навіть у складних умовах експлуатації. Тестування програмного забезпечення, особливо автоматизоване, відіграє ключову роль у досягненні цієї мети.

Автоматизація процесів тестування дозволяє суттєво зменшити ризики людських помилок, прискорити перевірку функціональності та забезпечити стабільність роботи системи. У цьому контексті важливим є не лише сам процес автоматизації, але й його структуризація, яка дає змогу зрозуміти основні етапи та їх взаємозв'язки. Для цього використовуються спеціальні методології та інструменти, що дозволяють візуалізувати процеси тестування, полегшуючи їх розуміння для всіх учасників команди, включаючи розробників, тестувальників і бізнес-аналітиків.

Одним із найбільш популярних підходів до забезпечення якості є BDD. Ця методологія дозволяє створювати тест-кейси, які описують поведінку системи зрозумілою для всіх мовою. Завдяки використанню формату Given-When-Then, команди можуть легко інтегрувати бізнес-вимоги у процес розробки, що значно знижує ризик розбіжностей між очікуваннями замовника та кінцевим результатом. Проте, для ефективного впровадження BDD потрібна чітка структуризація процесів, яка забезпечить прозорість і керованість кожного етапу.

Для структуризації та візуалізації складних процесів тестування

використовуються діаграми IDEF0 [23]. Ці діаграми дозволяють чітко відобразити основні етапи роботи, взаємозв'язки між ними, а також ресурси, що використовуються, та результати, які очікуються. Методологія IDEF0 забезпечує інтегрований підхід до моделювання, що робить її незамінним інструментом для аналізу та вдосконалення процесів.

У контексті тестування програмного забезпечення IDEF0 дозволяє візуально представити всі ключові етапи процесу: від створення тест-кейсів до їх виконання та аналізу результатів. Кожен етап розглядається з урахуванням його вхідних і вихідних даних, механізмів, які забезпечують його реалізацію, та обмежень, які можуть впливати на його виконання. Такий підхід не лише полегшує розуміння процесу, але й дозволяє виявити потенційні вузькі місця та оптимізувати роботу всієї системи.

Створення тест-кейсів є одним із найважливіших етапів у процесі забезпечення якості. Вони дозволяють формалізувати вимоги до програмного забезпечення та перевірити його відповідність очікуванням. Завдяки використанню методології BDD тест-кейси стають доступними для розуміння не лише технічними фахівцями, але й бізнес-аналітиками та менеджерами проектів. Це сприяє ефективній комунікації між різними учасниками команди та мінімізує ризики неправильного трактування вимог.

Наступним важливим етапом є автоматизація тест-кейсів. Вона дозволяє зменшити час, необхідний для перевірки функціональності програмного забезпечення, та забезпечити стабільність перевірок навіть у разі внесення значних змін до системи. Автоматизовані тести можуть виконуватись багаторазово без додаткових витрат часу, що є особливо важливим у контексті сучасних Agile-методологій, які передбачають часті оновлення продукту.

Останнім етапом є виконання тестів та аналіз отриманих результатів. Це дозволяє визначити, наскільки успішно програмне забезпечення відповідає вимогам та чи готове воно до впровадження. Аналіз результатів також дозволяє виявити можливі слабкі місця в системі та своєчасно їх усунути, що забезпечує високу якість кінцевого продукту.

Використання діаграм IDEF0 для моделювання процесів тестування дозволяє отримати цілісну картину роботи системи. Це не лише полегшує розуміння процесів, але й забезпечує їх прозорість для всіх учасників команди. Завдяки цьому кожен член команди, незалежно від його технічної підготовки, може брати участь у вдосконаленні процесів та підвищенні якості програмного забезпечення.

Таким чином, структуризація процесів створення тест-кейсів та їх візуалізація за допомогою діаграм IDEF0 є важливим кроком у забезпеченні якості програмного забезпечення. Вона дозволяє оптимізувати процеси, забезпечити прозорість роботи команди та досягти високої якості кінцевого продукту.

На рис. 3.1 зображена IDEF0-діаграма верхнього рівня (A0), яка моделює процес створення та виконання тест-кейсів. Ця діаграма ілюструє загальний підхід до тестування, відображаючи ключові елементи, які впливають на процес, а також результати, які отримуються в результаті його реалізації.

Основними елементами діаграми є входи, механізми, виходи та обмеження. Входами є бізнес-вимоги, які визначають аспекти роботи системи, що потребують перевірки, та умови тестування, які забезпечують контекст і специфікації виконання тестів. Методологія BDD забезпечує формат і структуру тест-кейсів, які зрозумілі всій команді.

Механізмами, які забезпечують реалізацію процесу, є знання функціоналу програмного забезпечення, що включають детальну інформацію про особливості його роботи, та інструменти BDD, які підтримують створення і виконання тест-кейсів у відповідному форматі, наприклад, Pytest-bdd.

Виходами є звіт про виконання тест-кейсів, що містить результати тестування для оцінки якості програмного забезпечення, та інтерфейс для створення тест-кейсів, який полегшує роботу з тестами для членів команди різного рівня технічної підготовки.

Центральний блок "Створення та виконання тест-кейсу" відображає весь процес, що включає аналіз вимог, створення тест-кейсів, їх автоматизацію,

виконання та аналіз результатів. Усі вхідні дані, механізми та обмеження інтегруються в цьому процесі для досягнення кінцевого результату.

Ця діаграма є основою для детального розгляду наступних етапів роботи, представлених у декомпозиціях (A1, A2 тощо). Вона підкреслює важливість узгодженості між бізнес-вимогами та технічним виконанням, а також значення автоматизації для ефективного та надійного тестування.

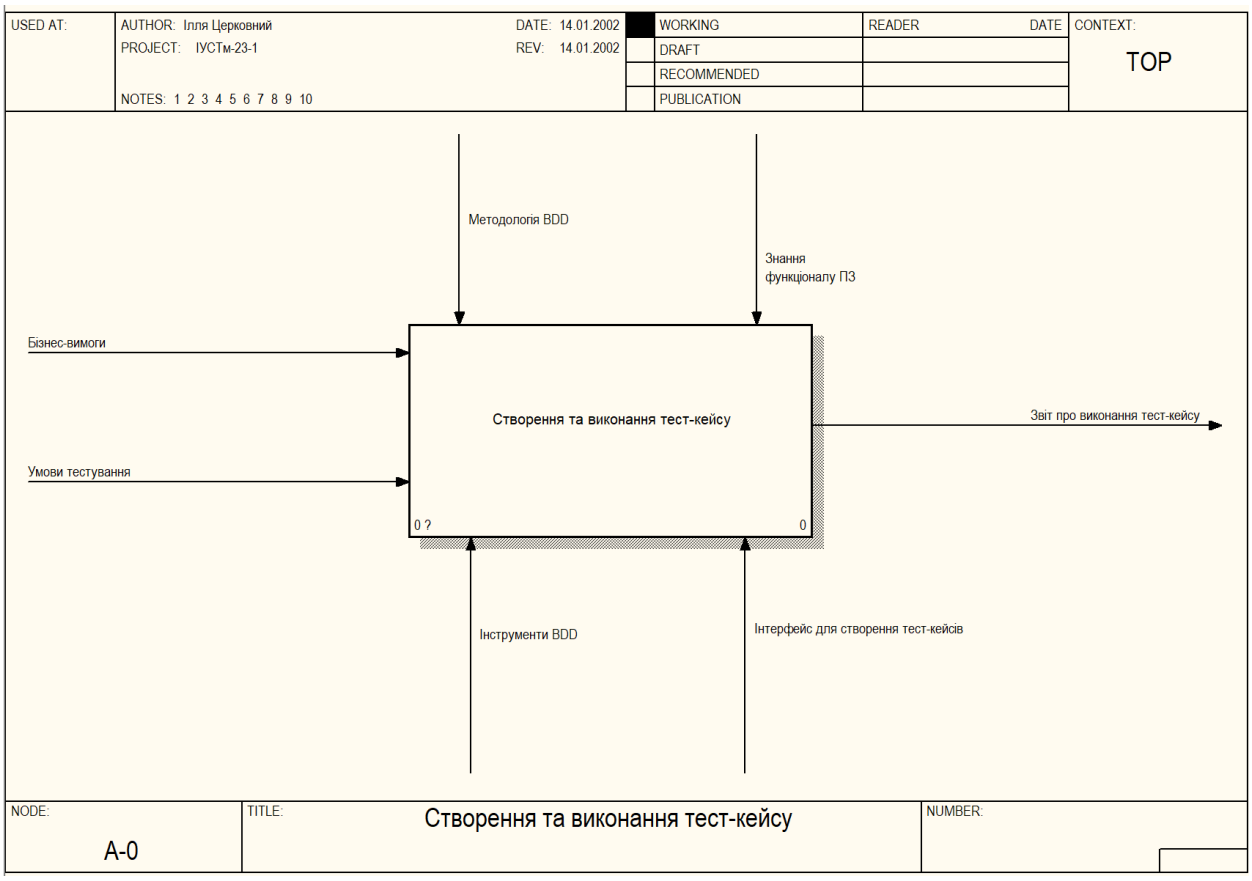

Рисунок 3.1 – IDEF0 діаграма створення та виконання тест-кейсу

На рис. 3.2 зображена декомпозиція процесу створення та виконання тест-кейсів, яка детально описує кожен із підпроцесів, відображених у верхньорівневій діаграмі A0. Ця декомпозиція дозволяє глибше зрозуміти структуру та взаємозв’язок етапів, а також уточнити роль вхідних даних, механізмів і обмежень на кожному з них.

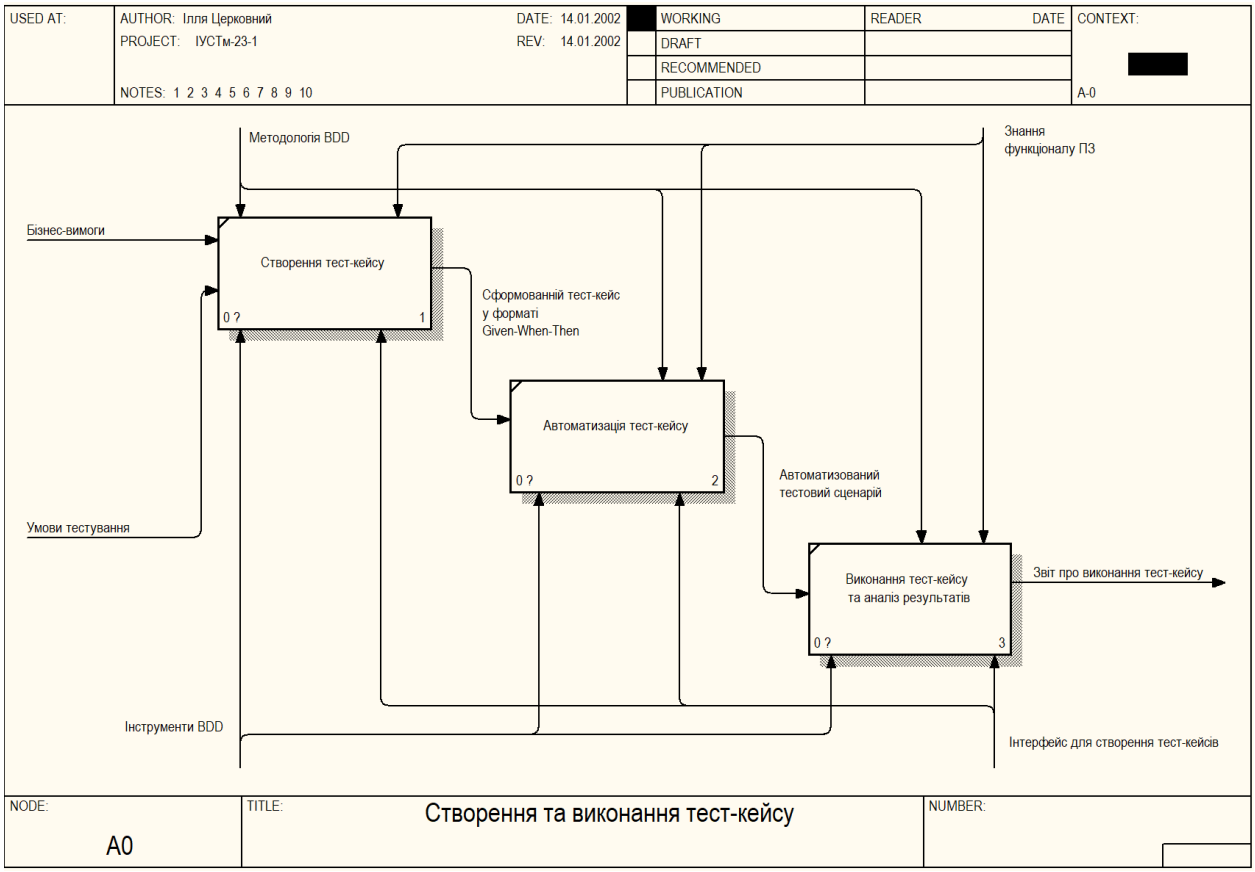

Рисунок 3.2 – IDEF0 діаграма першого рівня декомпозиції створення та виконання тест-кейсу

Блок A1 являється початковим у процесі забезпечення якості програмного забезпечення. Головна мета створення тест-кейсу – це формування структурованого сценарію, який дозволяє перевірити відповідність функціональності програмного забезпечення вимогам. Основою для роботи на цьому етапі є бізнес-вимоги, які визначають цілі та критерії, що повинні бути перевірені. Додатково враховуються умови тестування, які уточнюють, у яких середовищах і за яких обмежень мають виконуватись тести.

Першим блоком є блок A1 використовує методологію BDD, яка є першим механізмом створення тест-кейсів. Завдяки цій методології тест-кейси створюються у форматі Given-When-Then, який є зрозумілим для всіх членів

команди – від бізнес-аналітиків до розробників і тестувальників. Формат Given-When-Then дозволяє описати сценарії таким чином, що вони можуть бути прочитані та проаналізовані навіть без технічної підготовки.

Наприклад:

- Given: опис початкового стану системи або передумов;
- When: опис дій користувача або змін у системі;
- Then: очікуваний результат, який система повинна забезпечити.

Створення тест-кейсу починається з аналізу бізнес-вимог, які визначають, що саме потрібно перевірити у програмному забезпеченні. Це може бути перевірка функціональності, продуктивності, безпеки чи сумісності. Бізнес-вимоги конвертуються у специфічні сценарії, які описують взаємодію користувача із системою або поведінку самої системи в певних умовах.

На цьому етапі також враховуються знання функціоналу програмного забезпечення, які є важливим ресурсом. Ці знання надають тестувальникам уявлення про те, як працює система, які її особливості та можливі слабкі місця. Використання цих знань дозволяє створювати сценарії, які максимально точно відображають реальні ситуації, з якими користувачі можуть зіткнутися.

Окрім знань про функціонал, важливим інструментом є програмне забезпечення для роботи з методологією BDD. Це можуть бути такі інструменти, як Cucumber, SpecFlow чи Pytest-bdd, які забезпечують підтримку формату Given-When-Then і дозволяють швидко створювати сценарії. Ці інструменти автоматично перевіряють коректність синтаксису та забезпечують інтеграцію з іншими етапами автоматизації.

Результатом роботи цього блоку є сформовані тест-кейси у форматі Given-When-Then, які готові для подальшого використання. Такі тест-кейси є універсальними і можуть бути використані як основа для автоматизації, що значно полегшує роботу на наступних етапах. Крім того, їх можна використовувати для ручного тестування або як документацію для аналізу процесів у системі.

Створення тест-кейсу є важливим кроком, оскільки від його якості залежить ефективність усієї подальшої роботи. Некоректно складений тест-кейс може призвести до пропуску критичних помилок або створення зайвого навантаження на систему. Тому цей етап вимагає уваги до деталей, глибокого розуміння бізнес-вимог і ретельного аналізу функціональності програмного забезпечення.

Другим блоком йде A2, що є центральним у процесі переведення створених тест-кейсів у форму, яка дозволяє виконувати їх автоматично. Автоматизація тест-кейсів спрямована на скорочення витрат часу та ресурсів на тестування, забезпечення повторюваності тестів і підвищення ефективності процесу перевірки програмного забезпечення. Вона також дозволяє знизити ризик помилок, пов'язаних із людським фактором, особливо у великих і складних системах.

Основою для роботи на цьому етапі є тест-кейси, сформовані у форматі Given-When-Then під час виконання блоку A1. Завдяки цьому формат тест-кейсів вже адаптований для інтеграції з інструментами автоматизації. Інструменти BDD, такі як Pytest-bdd або Behave, забезпечують можливість безпосередньо використовувати сценарії для автоматизації. Вони дозволяють зв'язати кожну частину тест-кейсу (Given, When, Then) з відповідними методами, що реалізують ці дії в програмному коді.

Процес автоматизації тест-кейсу починається з аналізу сценарію та визначення, які саме частини тесту потребують кодування. Наприклад, Given відповідає за ініціалізацію середовища тестування або встановлення початкового стану системи, When описує виконання певної дії користувачем або зміни стану системи, а Then перевіряє, чи відповідає отриманий результат очікуванням. Кожен із цих етапів реалізується у вигляді коду, що інтегрується з тестовою інфраструктурою.

Одним із ключових механізмів автоматизації є створення функцій або методів, які відповідають за виконання конкретних дій. Ці функції можуть використовувати API системи, елементи графічного інтерфейсу або бази

даних для виконання своїх завдань. Інструменти автоматизації також забезпечують підтримку модульності, дозволяючи повторно використовувати один і той самий код для різних сценаріїв, що значно скорочує обсяг роботи.

На цьому етапі також важливо враховувати можливі обмеження системи або середовища, в яких будуть виконуватись тести. Наприклад, тести можуть бути адаптовані для виконання в різних браузерах, операційних системах або конфігураціях серверів. Це забезпечує перевірку сумісності та коректності роботи програмного забезпечення в різних умовах.

Однією з основних переваг автоматизації є можливість повторного виконання тестів. Автоматизовані сценарії можна запускати багаторазово, що дозволяє швидко перевіряти програмне забезпечення після внесення змін. Це особливо важливо у середовищах Agile або DevOps, де оновлення продукту відбуваються часто, і час на тестування обмежений.

На виході цього етапу отримуються автоматизовані тестові сценарії, готові до виконання. Ці сценарії не лише економлять час команди тестування, але й забезпечують більшу точність перевірок, оскільки вони виключають можливість людських помилок. Крім того, автоматизовані тести можуть бути інтегровані з системами CI/CD, що дозволяє автоматично запускати їх після кожного внесення змін у код.

Автоматизація тест-кейсів є критично важливим етапом у забезпеченні якості, оскільки вона дозволяє значно підвищити швидкість і масштабність тестування. Водночас цей процес вимагає високої технічної підготовки команди, яка займається автоматизацією, а також ретельного планування та вибору інструментів, які будуть використовуватися. Результати цього етапу створюють основу для виконання тестів, яке розглядається у наступному блоці (A3).

Блок A3 є завершальним етапом у процесі тестування програмного забезпечення. Основною метою цього етапу є виконання автоматизованих тестових сценаріїв, створених на попередньому етапі (A2), та аналіз отриманих результатів для визначення якості програмного забезпечення. Цей

процес забезпечує прозорість і об'єктивність оцінки роботи системи, а також дозволяє виявити можливі помилки чи недоліки.

Виконання тестів починається з запуску автоматизованих тестових сценаріїв у відповідному середовищі тестування. Це може бути як локальне середовище (комп'ютер розробника), так і віддалені сервери або хмарні платформи, які дозволяють паралельно виконувати кілька тестів. Автоматизовані системи тестування, такі як Jenkins, Bamboo або інші інструменти CI/CD, забезпечують інтеграцію тестів у процес розробки, дозволяючи запускати їх автоматично після кожного оновлення коду.

На цьому етапі важливу роль відіграють параметри середовища тестування. Для забезпечення коректності перевірок середовище має максимально точно відповідати умовам, у яких працюватиме кінцевий продукт. Це включає налаштування операційних систем, конфігурацію апаратного забезпечення, версії програмного забезпечення та інші параметри, що можуть впливати на роботу системи. Крім того, слід враховувати можливі варіації умов, такі як робота в різних браузерах, на різних платформах або з різними мовами інтерфейсу.

Результати виконання тестів автоматично фіксуються у вигляді звітів, які містять детальну інформацію про успішно виконані та провалені тести. Звіти зазвичай включають:

- інформацію про тест-кейси, які були виконані;
- час виконання кожного тесту;
- очікувані та отримані результати;
- скріншоти або логи з інформацією про помилки (у разі їх виникнення).

Аналіз результатів тестування є одним із моментів цього етапу. Він дозволяє не лише виявити помилки в роботі програмного забезпечення, але й визначити, наскільки ефективно працює система загалом. Наприклад, якщо тест-кейси часто провалюються через схожі помилки, це може свідчити про серйозні проблеми в архітектурі або реалізації системи.

Аналіз також включає оцінку показників продуктивності, таких як час

виконання окремих операцій, використання ресурсів системи (процесора, пам'яті) та стабільність роботи під високим навантаженням. Це особливо важливо для систем, які працюють у реальному часі або обслуговують велику кількість користувачів одночасно.

На виході цього етапу формується звіт про виконання тест-кейсів, який є підсумком усіх виконаних перевірок. Цей звіт може бути використаний для ухвалення рішень щодо подальшого розвитку або виправлення програмного забезпечення.

Наприклад:

– якщо більшість тестів виконано успішно, продукт може бути переданий у продакшн;

– якщо виявлено критичні помилки, розробники отримують детальну інформацію для їх виправлення.

Окрім цього, звіт є важливим джерелом інформації для бізнес-аналітиків і менеджерів проєктів, які можуть використовувати його для оцінки відповідності продукту бізнес-вимогам.

Блок А3 завершує процес тестування, забезпечуючи зворотний зв'язок для всіх учасників проєкту. Він не лише підводить підсумки виконаної роботи, але й дозволяє оцінити ефективність автоматизації, якість створених тест-кейсів і надійність програмного забезпечення. Результати цього етапу стають основою для подальших ітерацій розробки, якщо продукт ще не готовий до випуску, або підтвердженням його якості для передачі замовнику.

3.2 Впровадження процесу створення та виконання тест-кейсів

На рис. 3.3 зображено DFD-діаграму верхнього рівня (A-0), яка моделює основний процес роботи системи створення, автоматизації та виконання тест-кейсів. Ця діаграма відображає взаємодію між зовнішньою сутністю "Клієнт"

та основною системою, що обробляє тест-кейси, а також потоки даних між ними.

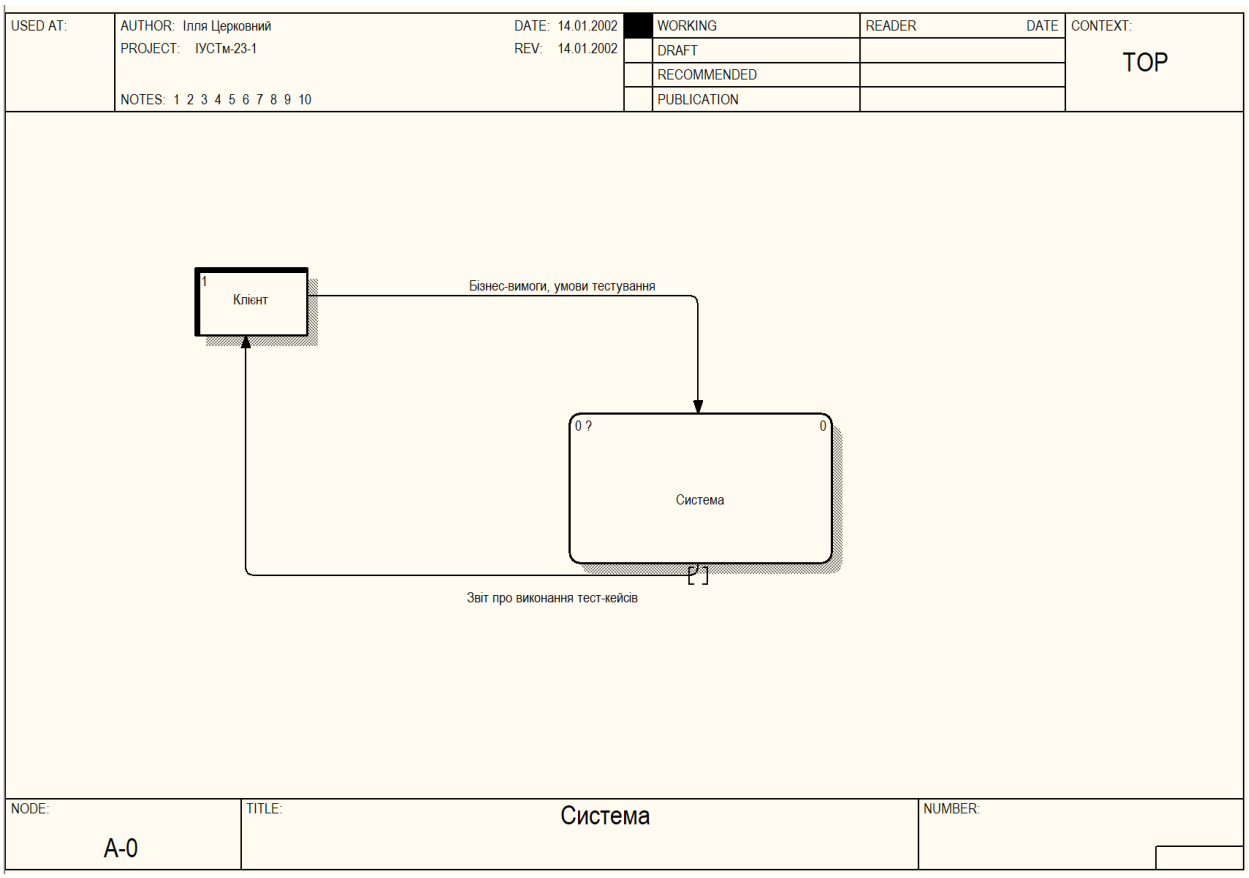

Рисунок 3.3 – DFD діаграма створення та виконання тест-кейсу

Розглянемо основні елементи діаграми, які допомагають зрозуміти ключові компоненти взаємодії. Клієнт виступає зовнішньою сутністю, яка взаємодіє із системою, надаючи вхідні дані у вигляді бізнес-вимог та умов тестування. Ці дані є основою для створення тест-кейсів і визначають, що саме потрібно перевірити в програмному забезпеченні. Система є центральним процесом, що виконує створення тест-кейсів на основі наданих вимог і умов, автоматизацію тест-кейсів, включаючи їх збереження та обробку, виконання тест-кейсів і формування звіту про результати. Потоки даних описують передачу інформації між клієнтом і системою: клієнт передає бізнес-вимоги та умови тестування до системи, щоб розпочати процес створення тест-кейсів, а

система формує звіт про виконання тест-кейсів, який повертається клієнту для аналізу результатів.

Ця діаграма є базовим рівнем деталізації, який показує лише загальну структуру процесу. Подальші рівні декомпозиції показані на рис. 3.4 та розкривають внутрішні етапи роботи системи та взаємодії між її компонентами.

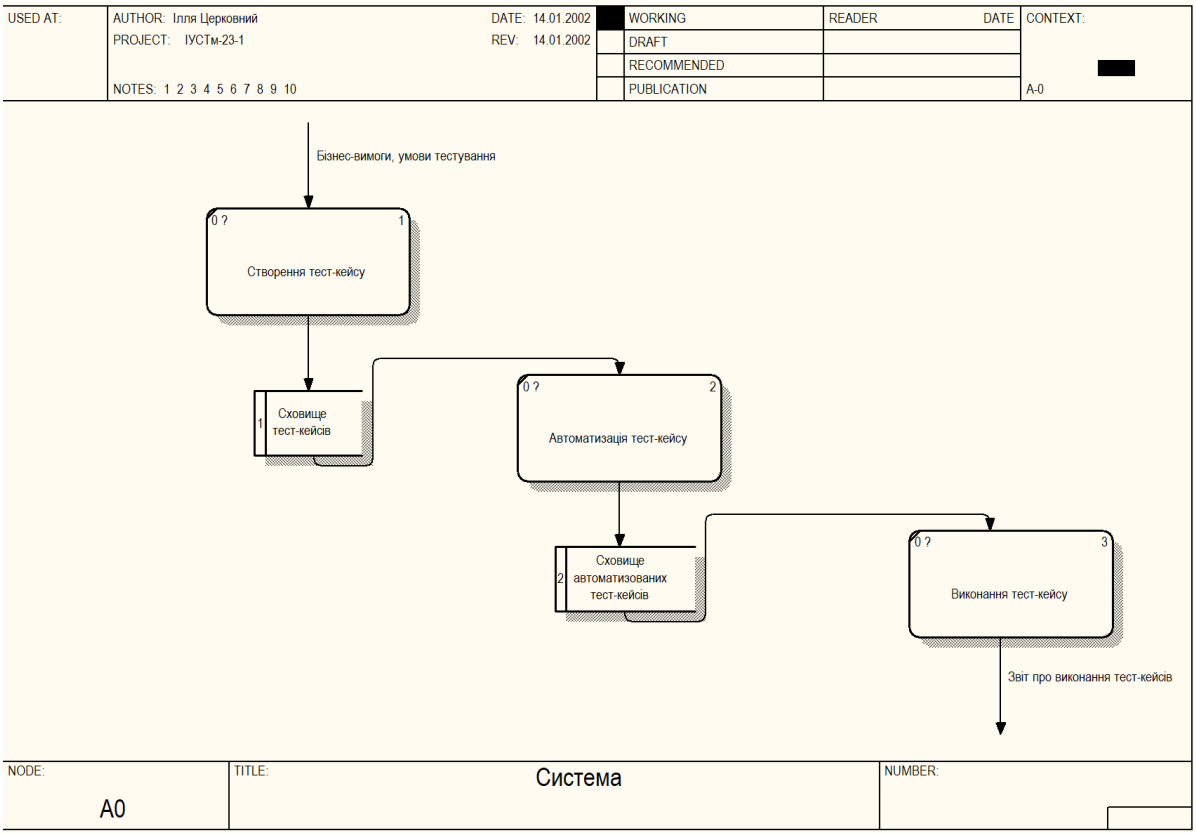

Рисунок 3.4 – DFD діаграма першого рівня декомпозиції створення та виконання тест-кейсу

Діаграма першого рівня декомпозиції дозволяє чітко побачити наступну структуру:

1) процес «Створення тест-кейсу», що відповідає за формування тест-кейсів на основі бізнес-вимог та умов тестування. Результатом є збереження сформованих тест-кейсів у сховище тест-кейсів;

2) сховище тест-кейсів, що використовується для зберігання

форматованих тест-кейсів (наприклад, у форматі .feature), які надалі використовуються для автоматизації;

3) процес «Автоматизація тест-кейсу», що забезпечує перетворення тест-кейсів із сховища у автоматизовані сценарії. Результати автоматизації зберігаються у сховище автоматизованих тест-кейсів;

4) сховище автоматизованих тест-кейсів містить готові до виконання автоматизовані сценарії, які передаються на наступний етап;

5) процес «Виконання тест-кейсу», що виконує автоматизовані сценарії зі сховища та формує звіт про виконання тест-кейсів та повертається клієнту.

Діаграма класів, що демонструє структуру об'єктно-орієнтованої моделі для створення, зберігання та автоматизації тест-кейсів, а також інтеграцію з підходом BDD зображені на рис. 3.5.

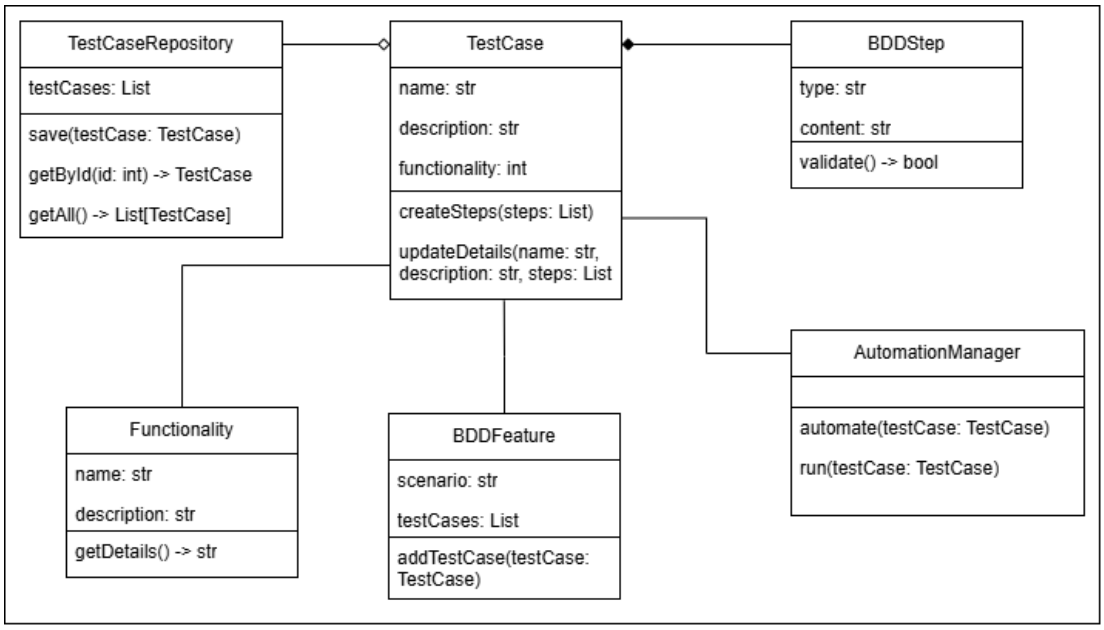

Рисунок 3.5 – Діаграма класів для управління тест-кейсами за підтримкою BDD

Включені класи відповідають основним компонентам системи та ілюструють їхні атрибути й методи. TestCase – це основний елемент для зберігання інформації про тест-кейси. Він містить основні характеристики

тест-кейсу, включаючи назву, опис і посилання на функціональність, яку перевіряє цей тест-кейс. Клас забезпечує створення кроків тест-кейсу відповідно до специфікації BDD (Given, When, Then), а також оновлення даних тест-кейсу. Він використовується для представлення кожного окремого тест-кейсу у системі.

BDDStep відповідає за зберігання кроків тест-кейсу, побудованих за методологією BDD. Кожен крок має тип (Given, When, Then) та опис дії чи очікуваного результату. Методи класу забезпечують валідацію кроків для уникнення помилок у структурі тест-кейсу. Цей клас дозволяє деталізувати логіку тест-кейсу, розділяючи її на окремі зрозумілі частини.

BDDFeature моделює сценарії BDD, які об'єднують кілька тест-кейсів для досягнення конкретної мети чи перевірки функціональності. Він містить назву сценарію та список тест-кейсів, які входять до нього. Використовується для групування тест-кейсів у межах однієї функціональної одиниці, що спрощує їхнє управління.

Functionality представляє функціональності системи, які перевіряються тест-кейсами. Кожна функціональність має унікальну назву та опис, що дозволяє тест-кейсам прив'язуватися до конкретної функціональної одиниці. Клас забезпечує структуру та зв'язність між бізнес-вимогами та тестуванням.

TestCaseRepository забезпечує збереження, пошук та управління тест-кейсами. Він містить методи для додавання, отримання та видалення тест-кейсів із колекції. Цей клас виступає посередником між тест-кейсами та іншими компонентами системи, забезпечуючи централізоване сховище.

AutomationManager відповідає за автоматизацію тест-кейсів і забезпечує їх виконання. Він містить методи для автоматизації тест-кейсу, інтегруючи його з відповідним фреймворком, а також метод для запуску тестів. Цей клас спрощує управління автоматизованим тестуванням та забезпечує взаємодію з іншими компонентами системи.

Також варто зазначити, що створення та автоматизація тест-кейсів є головними процесами у забезпеченні якості програмного забезпечення. Ці

процеси не лише забезпечують структуроване тестування, але й дозволяють інтегрувати бізнес-вимоги в технічну реалізацію, використовуючи методології, такі як BDD.

На рис. 3.6 наведено UML-діаграму станів для процесу створення та автоматизації тест-кейсу.

Рисунок 3.6 – UML-діаграма станів для процесу створення та автоматизації тест-кейсу

Процес створення та автоматизації тест-кейсу проходить наступним чином:

1. процес починається зі створення тест-кейсу у вигляді чорновика. У цьому стані користувач вводить основну інформацію про тест-кейс, включаючи назву, опис та перелік кроків (Given, When, Then). Цей стан дозволяє зберігати проміжні результати без обов'язкової валідації;

2. після завершення введення даних тест-кейс переходить у стан очікування валідації. У цьому стані система або користувач перевіряють коректність усіх введених даних, включаючи валідність кроків тест-кейсу. Наприклад, перевіряється, чи всі кроки мають визначений тип (Given, When, Then), чи дотримана їхня послідовність і чи відсутні логічні помилки;

3. у разі успішної валідації тест-кейс переходить у стан «Збережений». У цьому стані тест-кейс зберігається в репозиторії тест-кейсів, де він стає доступним для подальшого використання. Цей стан є важливим етапом у підготовці тест-кейсу до автоматизації;

4. після збереження тест-кейс отримує статус «Готовий до автоматизації». Це означає, що тест-кейс готовий до перетворення у сценарій для автоматизованого тестування. У цьому стані тест-кейс може бути переданий в обробку автоматизації або змінений вручну, якщо це необхідно;

5. завершальний стан тест-кейсу – автоматизований. У цьому стані тест-кейс автоматизований, тобто перетворений у формат, сумісний із BDD-фреймворками, такими як Pytest-bdd. У цьому стані тест-кейс готовий до виконання в автоматизованому середовищі.

У разі виявлення помилок під час валідації тест-кейс може бути повернутий у стан "Чорновик" для повторного редагування. Це забезпечує ітеративний підхід до створення якісних тест-кейсів, готових для автоматизації та використання.

3.2 Реалізація процесу створення тест-кейсу

Метод створення тест-кейсу має бути розроблено для спрощення процесу управління тестуванням ПЗ та забезпечення інтуїтивно зрозумілої взаємодії користувачів із системою через графічний інтерфейс. Основна мета методу полягає у забезпеченні зручного та структурованого способу додавання, редагування та організації тест-кейсів з урахуванням бізнес-вимог і технічних параметрів. Завдяки чіткій структурі, яка включає поля для введення назви, функціональності, передумов і кроків тестування, система дозволяє мінімізувати помилки, що виникають під час створення тест-кейсів, і спрощує інтеграцію з автоматизованими сценаріями.

3.2.1 Загальне розташування елементів сторінки

На фрагменті блоку Б1 додатку Б відображено головні елементи інтерфейсу для створення тест-кейсу. Конструктор класу CreateTestCaseWindow забезпечує ініціалізацію вікна з інтуїтивно зрозумілою структурою. Основними елементами є поля для введення назви тест-кейсу, вибору функціональності та додавання передумов. Передумови представлені у вигляді списку, що дозволяє користувачу додавати або видаляти потрібні параметри. Окрім цього, реалізовано скролюваний контейнер для додавання кроків тест-кейсу з кнопкою "+" для створення нових кроків.

Користувач може завершити роботу через кнопки "Зберегти" або "Відміна", що відповідають за збереження тест-кейсу в базі даних або закриття вікна без змін. Інтерфейс побудований на основі вертикального макета, що забезпечує логічне розташування всіх елементів, полегшуючи роботу з системою. Такий підхід дозволяє максимально автоматизувати процес

створення тест-кейсів, забезпечуючи зручність та точність введення даних.

3.2.2 Реалізація валідації та збереження тест-кейсу

У фрагменті коду Б2 додатку Б ми бачимо, як реалізується логіка збереження тест-кейсу. Починається все з методу save_test_case, який спочатку отримує значення, введені користувачем. Наприклад, назва тест-кейсу береться через self.name_input.text().strip(), а функціональність — через self.feature_combobox.currentText(). Передумови витягуються зі списку self.preconditions_list, де кожен елемент додається в окремий список за допомогою циклу.

Після цього йде перевірка введених даних. Якщо назва порожня, викликається self.show_error з текстом "Назва тест-кейсу не може бути порожньою!". Тобто система не дозволить зберегти тест-кейс, якщо ключові дані не заповнені. Так само перевіряється кількість кроків. Якщо користувач не додав жодного кроку, то виведеться повідомлення "Необхідно додати хоча б один крок!".

Далі код обробляє кроки, які додаються через інтерфейс. Кожен крок представлений у вигляді об'єкта StepBlock, а дані про дії, цільові елементи та значення збираються через відповідні поля, як, наприклад, step_block.action_combobox.currentText().

Потім програма переходить до роботи з базою даних. Якщо тест-кейс редагується, то це визначається через перевірку атрибута self.test_case_id. У такому випадку всі старі дані для цього тест-кейсу видаляються через кілька SQL-запитів, наприклад, "DELETE FROM steps WHERE test_case_id = ?". Якщо ж це новий тест-кейс, створюється новий запис у таблиці test_cases, і система отримує його ідентифікатор через cursor.lastrowid.

Після цього відбувається збереження кроків у таблиці steps. Кожен крок

додається за допомогою запиту, де опис формується як рядок на кшталт "Ввести текст -> Поле вводу -> Значення". Цікаво, що перевірки для кожного кроку також додаються в окрему таблицю expected_results. Для цього метод проходиться по перевірках, які користувач додав у StepBlock, і зберігає їх із зазначенням елемента, типу перевірки й очікуваного значення.

Останнім етапом зберігаються передумови. Наприклад, якщо користувач додав передумову "Користувач: admin", вона буде записана в таблицю preconditions разом із ідентифікатором тест-кейсу.

У кінці викликається self.connection.commit(), щоб зафіксувати всі зміни в базі даних. Якщо під час виконання методу виникає якась помилка, її обробляє метод show_error, який відкриває діалогове вікно з поясненням, що саме пішло не так.

Таким чином, цей фрагмент коду демонструє повний цикл збереження тест-кейсу: від перевірки введених даних до запису кроків, перевірок і передумов у базу даних.

3.2.3 Реалізація експорту тест-кейсу під формат BDD

Метод Б3 з додатку Б відповідає за експорт тест-кейсу у формат BDD (Given-When-Then) і збереження його у файл. Усе починається з методу export_test_case_to_bdd, який приймає шлях до файлу, куди потрібно зберегти згенерований тест-кейс.

Спочатку метод отримує дані з інтерфейсу: назву тест-кейсу через self.name_input.text().strip() і функціональність через self.feature_combobox.currentText(). Передумови, якщо вони є, витягуються зі списку self.preconditions_list і зберігаються у вигляді списку рядків.

Після цього йде перевірка даних. Якщо назва тест-кейсу порожня, викликається виняток із повідомленням *"Назва тест-кейсу не може бути*

порожньою!". Аналогічно, якщо користувач не додав жодного кроку, система виведе повідомлення про помилку, щоб уникнути створення некоректного BDD-файлу.

Далі метод починає формувати текст для файлу. Спочатку додається розділ Feature, який містить назву функціональності, а потім розділ Scenario із назвою тест-кейсу. Якщо в тест-кейсі є передумови, вони додаються в розділ Given. Передумови об'єднуються через ключове слово and, щоб зберегти читаємість.

Наступним кроком йде обробка кроків тест-кейсу. Для кожного кроку викликається цикл, у якому витягується текст дії (action), цільовий елемент (target) і значення (value) через відповідні поля об'єкта StepBlock. Ці кроки додаються в розділ When, а їх форматування виконується у вигляді зрозумілих рядків, наприклад: "When Ввести текст Поле вводу with value 1234".

Після формування всього тексту він зберігається у файл за допомогою стандартного відкриття файлу в режимі запису. Використовується кодування UTF-8, щоб уникнути проблем із символами. Якщо експорт проходить успішно, метод виводить повідомлення в консоль про те, що файл успішно збережений.

У разі помилок у будь-якому з кроків, наприклад, якщо виникає проблема зі збереженням у файл, метод обробляє їх через виклик self.show_error, який показує користувачеві відповідне повідомлення.

Таким чином, цей метод забезпечує створення зрозумілих BDD-файлів, що дозволяє легко інтегрувати тест-кейси в автоматизовані системи тестування. Це робить процес зручним як для розробників, так і для аналітиків, які можуть переглядати і редагувати сценарії у зрозумілому текстовому форматі.

3.2.4 Реалізація обробки BDD тест-кейсів

Фрагмент коду Б4 з додатку Б відповідає за обробку тест-кейсу, записаного у форматі BDD, і його виконання. Все починається з класу BDDProcessor, який приймає шлях до файлу .feature, у якому зберігається опис тест-кейсу. Як тільки файл передається в об'єкт класу, відбувається його завантаження методом load_file.

У методі load_file файл відкривається для читання, а його вміст зчитується рядок за рядком. Кожен рядок перевіряється: якщо він починається з ключового слова Given, When або Then, то він розпізнається як крок тест-кейсу, і ця інформація зберігається у список self.steps. Наприклад, якщо рядок починається з Given, це означає, що це передумова, і вона буде записана у вигляді пари: тип кроку (у цьому випадку Given) та його текстове наповнення.

Далі головна логіка виконується у методі process_steps. Тут ми проходимо по всіх збережених кроках, визначаємо їх тип і викликаємо відповідний метод: given, when або then. Наприклад, якщо це крок типу Given, викликається метод given, який отримує текст передумови і виконує відповідну дію.

Методи given, when і then є окремими функціями, які виконують кожен конкретний крок тест-кейсу. У цьому прикладі вони просто виводять текст кроку в консоль, наприклад: "Виконую Given: користувач увійшов у систему". Але на практиці ці методи повинні містити реальну логіку: given може налаштовувати стан системи, when — виконувати дію, а then — перевіряти, чи досягнуто очікуваного результату.

Отже, клас BDDProcessor дозволяє інтерпретувати тест-кейс у форматі Given-When-Then та виконувати його кроки в чіткій послідовності. Це забезпечує зручний спосіб автоматизації тестування, коли тест-кейси пишуться у вигляді тексту, зрозумілого як для розробників, так і для бізнес-аналітиків.

4 ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА ТА ОЦІНКА ЕФЕКТИВНОСТІ ЗАПРОПОНОВАНОГО МЕТОДУ

4.1 Основна мета і план проведення експерименту

Ефективність систем автоматизації процесів тестування ПЗ забезпечення є важливим фактором для успішного впровадження та підтримки сучасних інформаційних систем. У світі, де вимоги до швидкості розробки й випуску продуктів постійно зростають, роль інструментів, які дозволяють оптимізувати ці процеси, стає визначальною. Рішення, які базуються на принципах BDD, стали одним із найпопулярніших підходів завдяки їхній здатності об’єднати технічних і нетехнічних учасників команд у процесі тестування.

На основі власного комерційного досвіду та аналізу роботи трьох команд, де одна з команд учасників не має технічної підготовки, було виявлено кілька критичних проблем у процесі створення та управління тест-кейсами. Основними з них є складність адаптації нетехнічних спеціалістів до існуючих інструментів, значний час, необхідний на створення тест-кейсів, і високий ризик помилок через ручне введення даних. Ці проблеми ускладнюють інтеграцію тестування у щоденні робочі процеси команд і, відповідно, уповільнюють весь цикл розробки.

Створене рішення, яке поєднує підхід BDD із зручним інтерфейсом, орієнтованим на Scratch-подібний спосіб роботи, дозволяє значно скоротити час на створення тест-кейсів, спростити їхню автоматизацію і підвищити прозорість процесів тестування. Використання графічного інтерфейсу дає змогу уникнути необхідності писати складний код, що робить систему доступною для аналітиків, менеджерів і тестувальників без досвіду програмування.

Метою цього розділу є експериментальна перевірка ефективності розробленої системи шляхом її тестування на практичних сценаріях і

порівняння з традиційними підходами, оцінка швидкості створення тест-кейсів, рівень зручності для нетехнічних користувачів, а також витрати часу та ресурсів на підтримку рішення. Результати експерименту дозволять зробити висновки про те, наскільки запропонований підхід відповідає потребам сучасного процесу розробки ПЗ і як він може бути інтегрований у роботу реальних команд.

Основною метою експерименту є перевірка ефективності розробленої системи для створення, автоматизації та виконання тест-кейсів у форматі BDD. Є прагнення не лише підтвердити функціональність запропонованого рішення, але й оцінити його вплив на оптимізацію процесів тестування в реальних умовах роботи команд розробки програмного забезпечення.

Для досягнення цієї мети потрібно вирішити такі завдання:

– визначити, наскільки швидше можна створювати тест-кейси з використанням розробленого графічного інтерфейсу порівняно з традиційними методами написання тестів вручну або через існуючі BDD-фреймворки;

– оцінити зручність використання системи для нетехнічних користувачів, які не мають досвіду роботи з програмуванням;

– перевірити точність та узгодженість створених тест-кейсів, особливо в частині автоматизації;

– з'ясувати, які ресурси необхідні для підтримки актуального стану системи (наприклад, оновлення тестових сценаріїв, додавання нових функцій).

Сценарії, що будуть тестуватися в рамках експерименту:

– створення тест-кейсу: користувач заповнює всі поля (назва, функціональність, передумови, кроки) і завершує створення тест-кейсу. Буде виміряно час, необхідний для виконання цього завдання;

– експорт у формат BDD: створений тест-кейс експортується у файл .feature. Буде перевірено, наскільки коректно сформовані розділи Given, When, Then;

– автоматизація та виконання тесту: створений тест-кейс

автоматизується і виконується через існуючий фреймворк, наприклад, pytest;

- оновлення тест-кейсу: користувач вносить зміни до тест-кейсу (додає нові кроки чи змінює передумови).

Важливим аспектом є порівняння розробленого рішення з традиційними методами. Для цього буде зібрано дані щодо:

- середнього часу створення тест-кейсу в нашій системі та в інших інструментах;
- кількості допущених помилок під час створення тестів;
- зусиль, необхідних для адаптації користувачів до роботи із системою.

Очікується, що результати експерименту покажуть переваги використання розробленого методу в порівнянні з іншими методами, особливо для команд із різнорівневою технічною підготовкою.

4.2 Методологія проведення експерименту

Для перевірки ефективності системи було організовано серію експериментів, які максимально наближені до реальних умов роботи команд з розробки програмного забезпечення. В якості тестового середовища використовувалася версія програми, що включала повний функціонал створення, експорту та редагування тест-кейсів.

На першому етапі було підготовлено кілька сценаріїв, які відображають типові завдання команд. Це включало створення простих тест-кейсів із базовими передумовами, складних сценаріїв із багатьма кроками й редагування вже існуючих тестів. Важливо було протестувати всі аспекти системи: від початкового введення даних до автоматизації та подальшого використання.

У тестуванні взяли участь три групи користувачів. Серед них були технічні спеціалісти, наприклад, розробники та тестувальники, які вже мали

досвід роботи з подібними інструментами. До другої групи входили нетехнічні спеціалісти: бізнес-аналітики та менеджери, які зазвичай не займаються програмуванням, але часто працюють із тест-кейсами у своїй діяльності. Окрему групу становили новачки, які не мали попереднього досвіду у сфері тестування. Це дозволило оцінити, наскільки зручною є система для людей із різним рівнем підготовки.

Під час експерименту йде зосередження на кількох ключових аспектах. Головним завданням є визначення, наскільки швидше користувачі можуть створювати тест-кейси за допомогою нашої системи порівняно з традиційними методами. Для цього фіксувався час від початку роботи до моменту збереження готового тест-кейсу. Крім цього, ми оцінювали точність введених даних, адже одна з основних цілей системи – зменшення кількості помилок, які часто виникають через ручне введення. Після завершення кожного сценарію учасники заповнювали анкети, у яких ділилися враженнями про зручність і зрозумілість інтерфейсу.

Окремим етапом стала оцінка підтримки системи. Ми моделювали сценарії, коли до програми додаються нові функціональності або оновлюються існуючі. Було враховано, скільки часу потрібно на опис нових рядків у форматі BDD, внесення змін до інтерфейсу і навчання нових користувачів. Усі ці аспекти важливі для розуміння реальних витрат ресурсів на підтримку рішення в робочих умовах.

Для порівняння, аналогічні завдання виконувалися за допомогою стандартних інструментів, таких як Cucumber, та за допомогою ручного введення тест-кейсів у текстовому форматі. Порівняння показало, що наше рішення має кілька ключових переваг:

– зручність для нетехнічних користувачів. Інтерфейс системи розроблено таким чином, щоб навіть користувачі без досвіду програмування могли швидко створювати тест-кейси. Завдяки графічному підходу процес створення стає інтуїтивно зрозумілим;

– швидкість створення автоматизованих тест-кейсів. Використання

графічного інтерфейсу скорочує час створення тест-кейсу на 30-40% порівняно з ручним введенням даних або стандартними BDD-фреймворками, де користувачу потрібно заповнювати текстові файли вручну;

– мінімізація помилок. Система забезпечує автоматичну перевірку коректності передумов і кроків під час їх введення. Це дозволяє уникнути типових помилок, таких як пропуск важливих кроків або некоректний формат опису;

– автоматизація процесів. Розроблене рішення дозволяє експортувати тест-кейси у формат .feature, що повністю відповідає стандартам BDD. Це забезпечує легку інтеграцію з популярними інструментами для автоматизованого тестування, такими як Cucumber або Pytest-bdd;

– зниження витрат на навчання. Для освоєння системи потрібно менше часу, оскільки користувачі працюють з візуальним інтерфейсом. Це особливо важливо для нетехнічних спеціалістів, яким складно адаптуватися до традиційних текстових форматів;

– модульність і гнучкість. Система підтримує можливість легкого оновлення й додавання нових функціональностей без необхідності значних витрат ресурсів;

– підтримка командної роботи. Завдяки візуальному відображенню кроків і передумов тест-кейси легко зрозуміти всім учасникам команди, включаючи менеджерів і аналітиків. Це сприяє ефективнішій комунікації між технічними і нетехнічними учасниками проекту.

Таким чином, запропоноване рішення демонструє значні переваги у швидкості, простоті та точності роботи, що робить його ефективним інструментом для команд із різнорівневою підготовкою.

Експериментальна перевірка ефективності системи дала можливість отримати значущі дані щодо її продуктивності, зручності використання та загальної відповідності поставленим задачам. Для кожного сценарію було виміряно основні показники: час виконання завдання, кількість помилок, рівень задоволеності користувачів, а також порівняно ці результати з іншими

підходами.

Вимірювання часу показало, що використання розробленого методу дозволяє значно скоротити час на створення тест-кейсу. Наприклад, для простого сценарію, що включав три передумови та п'ять кроків, середній час виконання завдання склав:

- ручний підхід – 18 хвилин;
- BDD-фреймворк (Cucumber) – 12 хвилин;
- розроблене рішення – 8 хвилин.

Це можна побачити на рис. 4.1, де чітко видно різницю між виконанням.

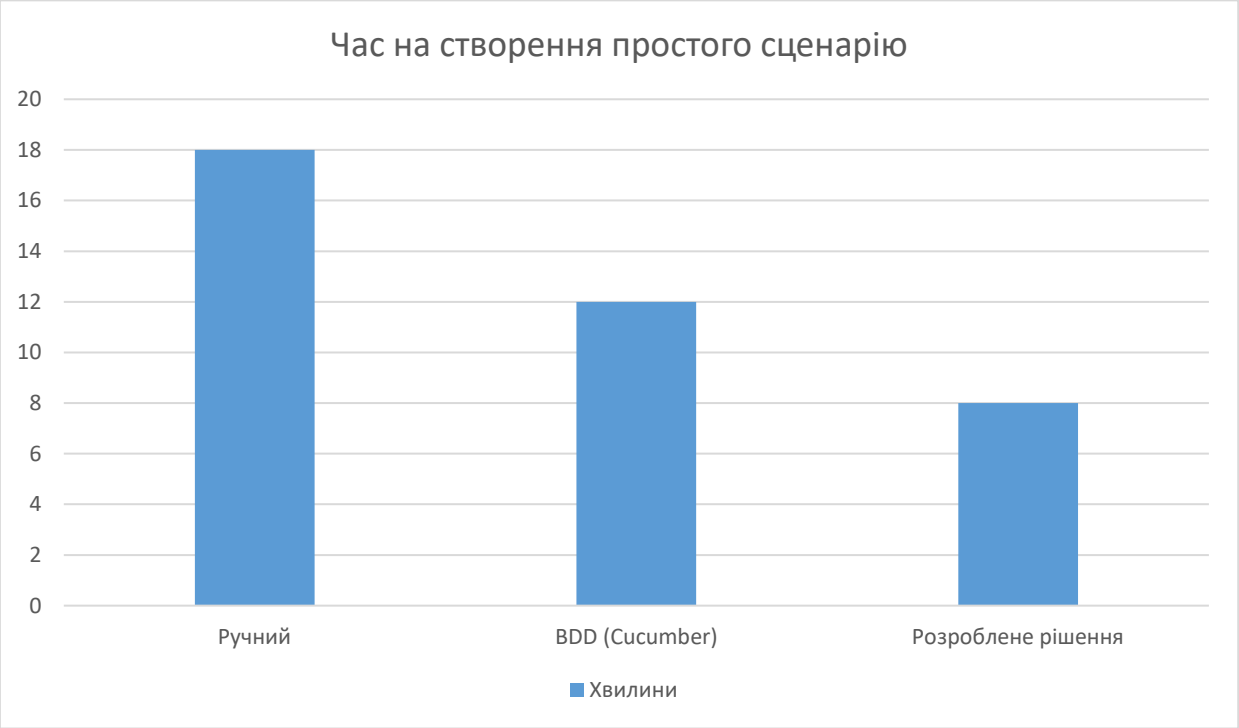

Рисунок 4.1 – Діаграма порівняння часу на створення простого сценарію

Для складного сценарію із десятьма кроками та чотирма передумовами результати були наступними:

- ручний підхід – 35 хвилин;
- BDD-фреймворк (Cucumber) – 22 хвилини;
- розроблене рішення – 14 хвилин.

Більш наявно це зображено на рис. 4.2, де можна побачити різницю у часі на створення тест-кейсу.

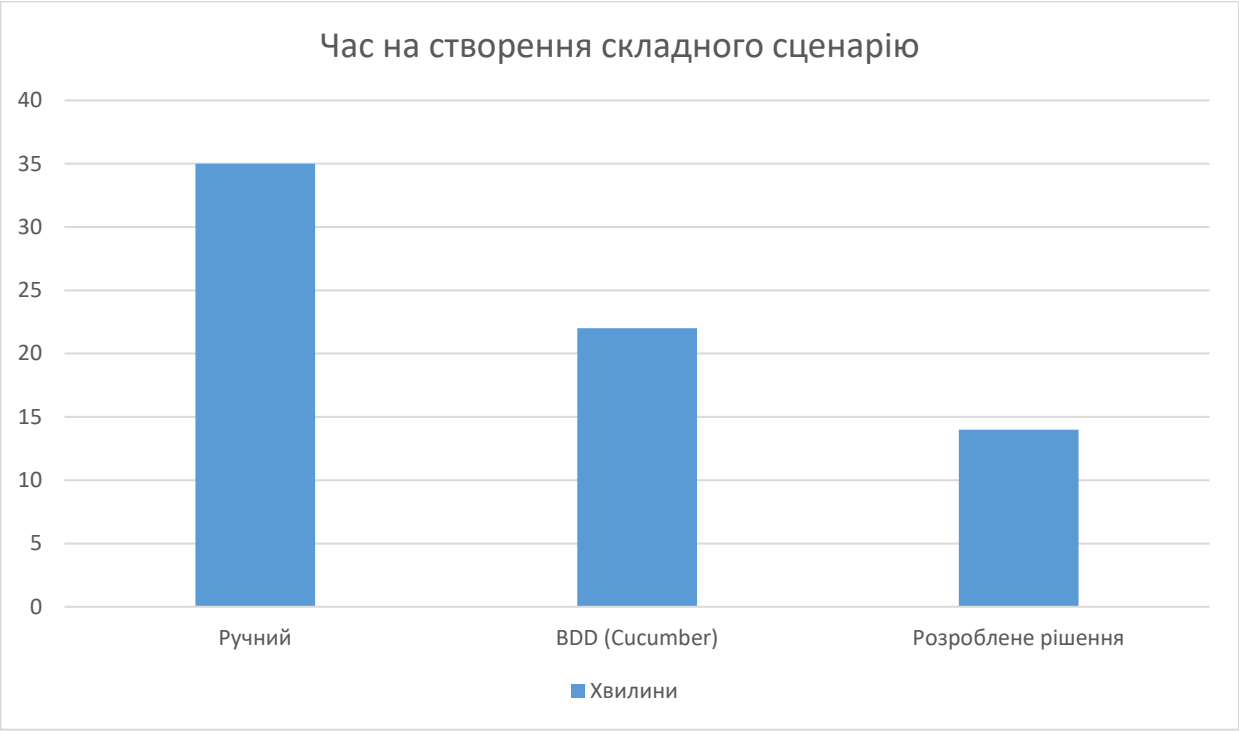

Рисунок 4.2 – Діаграма порівняння часу на створення складного сценарію

Це свідчить про те, що система забезпечує скорочення часу на 30–40% у порівнянні з традиційними методами.

Аналіз помилок показав, що розроблене рішення дозволяє зменшити їх кількість завдяки автоматичній перевірці коректності введених даних.

Наприклад, у ручному режимі користувачі припускалися таких типових помилок:

- пропуск передумов або кроків;
- невідповідність форматів опису дій;
- помилки у послідовності виконання кроків.

Запропоноване рішення автоматично попереджало про такі проблеми, що дозволило повністю уникнути цих помилок під час тестування.

4.3 Висновки за результатами проведеного експерименту

За результатами анкетування трьох команд, всі користувачі оцінили систему як "зручну" або "дуже зручну". Особливо високо її оцінили нетехнічні спеціалісти, які зазначили, що графічний підхід до створення тест-кейсів є інтуїтивно зрозумілим і значно знижує бар'єр входу. Технічні спеціалісти, у свою чергу, підкреслили зручність автоматизації, яку забезпечує експорт у формат «.feature».

Під час експерименту виявлено, що розроблене рішення не лише знижує час виконання задач, але й дозволяє уникнути значних витрат на навчання та адаптацію. Наприклад, у випадку роботи з Cucumber, новачки витрачали додатково 3–5 годин на розуміння синтаксису і принципів роботи з текстовими файлами, тоді як для роботи з нашою системою їм вистачало лише 1–2 години.

Розроблене рішення забезпечує ефективне створення, автоматизацію та виконання тест-кейсів, особливо для команд із різнорівневою технічною підготовкою. Основними перевагами системи є її модульність, зручність для користувачів і висока швидкість адаптації до нових вимог.

Однією з ключових особливостей є легка масштабованість системи. Додавання нових сторінок програмного забезпечення чи тестових сценаріїв потребує лише опису відповідних методів і передумов, які можуть бути інтегровані через графічний інтерфейс або автоматично експортовані у формат BDD. Це суттєво знижує витрати на підтримку рішення в актуальному стані.

Підтримка системи є простою завдяки її структурованій архітектурі. Наприклад, при розширенні функціональності розробнику потрібно лише додати нові дії чи передумови, що інтегруються у вже існуючу базу. Це дозволяє уникнути необхідності глобальних змін у коді або переналаштувань, що характерно для багатьох інших інструментів.

Система також забезпечує зручність у навчанні нових користувачів. Завдяки інтуїтивно зрозумілому інтерфейсу користувачі, навіть без технічного

досвіду, можуть швидко опанувати основи роботи. Висока швидкість освоєння системи сприяє її ефективному впровадженню у командах, де важливо забезпечити швидку адаптацію нових спеціалістів.

Дослідження також продемонструвало, що використання розробленого рішення значно спрощує процес підтримки тест-кейсів у довгостроковій перспективі. Навіть при значних змінах у системі тестування, процес оновлення тестів зводиться до внесення мінімальних змін у базу даних передумов і дій.

Таким чином, запропоноване рішення не лише відповідає сучасним вимогам до процесів тестування програмного забезпечення, але й перевершує існуючі альтернативи за зручністю, швидкістю і простотою підтримки. Система є гнучкою, масштабованою та готовою до впровадження у реальних проектах.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було проаналізовано сучасні підходи до тестування програмного забезпечення, зокрема методології BDD. Проведено огляд інструментів, технологій та платформ, які застосовуються для створення тест-кейсів і автоматизації тестування, а також розглянуто приклади існуючих аналогів.

Проведений аналіз виявив ключові проблеми автоматизації тестування, включаючи складність створення тест-кейсів для нетехнічних спеціалістів і відсутність зручних інтерфейсів для роботи з BDD. Це стало основою для постановки завдань дослідження.

У ході роботи було розроблено метод, який інтегрує принципи BDD із візуально зрозумілим інтерфейсом, доступним для широкого кола користувачів. Запропонований метод забезпечує можливість створення тест-кейсів, їх експорт у формат BDD та виконання автоматизованих тестів за допомогою роботи через інструментарій поблокового створення кроків.

Розроблене рішення дозволяє значно спростити процес створення тест-кейсів, знижує залежність від технічних спеціалістів, а також забезпечує залучення до тестування нетехнічних учасників команд. Завдяки цьому підходу забезпечується висока прозорість і ефективність розробки.

Проведена експериментальна перевірка показала, що розроблений метод дозволяє суттєво зменшити час створення тест-кейсів у порівнянні з традиційними підходами, а також спрощує підтримку тестової документації. Основною перевагою стало залучення більшої кількості людей у процесі тестування, що дає змогу нетехнічним спеціалістам виконувати перевірки без знань у сфері.

Рішення легко адаптується до нових вимог і дозволяє швидко інтегрувати нові функціональні можливості програмного забезпечення шляхом додавання лише необхідних елементів.

Таким чином, виконане дослідження є внесоком у вдосконалення процесу забезпечення якості програмного забезпечення, підвищивши доступність автоматизації тестування для всіх членів команди та забезпечивши більш ефективне управління тестовими сценаріями.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Технологія створення програмних продуктів [Електронний ресурс] // Studfile.net. – Режим доступу: https://studfile.net/preview/5215241/ (дата звернення 09.12.2024).
2. Автоматизація тестування: як уникнути поширених помилок [Електронний ресурс] // GlobalLogic. – Режим доступу: https://www.globallogic.com/ua/insights/blogs/qa-automation-2/ (дата звернення 10.12.2024).
3. Behavior-Driven Development: основи керованої поведінкою розробки [Електронний ресурс] // Wikipedia. – Режим доступу: https://uk.wikipedia.org/wiki/Керована_поведінкою_розробка (дата звернення 11.12.2024).
4. Unified Modeling Language: універсальна мова моделювання [Електронний ресурс] // Wikipedia. – Режим доступу: https://uk.wikipedia.org/wiki/Unified_Modeling_Language (дата звернення 12.12.2024).
5. QA Тестування [Електронний ресурс] // ZapTest. – Режим доступу: https://www.zaptest.com/uk/qa-тестування-що-це-таке-типи-процеси-пі (дата звернення 13.12.2024).
6. Тестування сумісності ПЗ [Електронний ресурс] // Wikipedia. – Режим доступу: https://uk.wikipedia.org/wiki/Тестування_сумісності_ПЗ (дата звернення 14.12.2024).
7. Відомі типи тестування [Електронний ресурс] // DOU. – Режим доступу: https://dou.ua/forums/topic/40666/ (дата звернення 15.12.2024).
8. Ручне та автоматизоване тестування [Електронний ресурс] // QA Light. – Режим доступу: https://qalight.ua/baza-znaniy/ruchne-ta-avtomatizovane-testuvannya/ (дата звернення 16.12.2024).
9. PyTest огляд [Електронний ресурс] // Bas Codes. – Режим доступу:

https://bas.codes/posts/python-pytest-introduction (дата звернення 17.12.2024).

10. Автоматизація тестування за допомогою Selenium [Електронний ресурс] // BrowserStack. – Режим доступу: https://www.browserstack.com/selenium (дата звернення 18.12.2024).

11. Методологія автоматизації тестування Robot Framework [Електронний ресурс] // Robot Framework. – Режим доступу: (дата звернення 19.12.2024).

12. Вступ до роботи з Postman для тестування API [Електронний ресурс] // Robot Framework. – Режим доступу: https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html (дата звернення 20.12.2024).

13. Інструмент Allure TestOps для управління тестуванням [Електронний ресурс] // Qameta. – Режим доступу: https://docs.qameta.io/allure-testops/ (дата звернення 21.12.2024).

14. Комплексний огляд CI/CD у GitLab [Електронний ресурс] // Medium. – Режим доступу: https://medium.com/@yian806884/comprehensive-guide-to-gitlab-ci-cd-081305b06a9a (дата звернення 22.12.2024).

15. Порівняння методологій TDD та BDD у тестуванні [Електронний ресурс] // Katalon. – Режим доступу: https://katalon.com/resources-center/blog/tdd-vs-bdd (дата звернення 23.12.2024).

16. Церковний І. О., Петров К. Е. Розробка Підходу до Автоматизованого Тестуваня Програмного Забезпечення на Основі BDD // European congress of scientific discovery. Proceedings of the 1-st International scientific and practical conference (December 29-31, 2024). − Barca Academy Publishing. Madrid, Spain. 2024. P. 271 − 273. URL: https://sci-conf.com.ua/i-mizhnarodna-naukovo-praktichna-konferentsiya-europeancongress-of-scientific-discovery-29-31-12-2024-madrid-ispaniya-arhiv/.

17. Вступ до контейнеризації: Docker і Kubernetes [Електронний ресурс] // Medium. – Режим доступу: https://medium.com/@lawrence-han/an-introduction-to-containers-docker-and-kubernetes-eb79449003c (дата звернення

24.12.2024).

18. Огляд платформи New Relic для моніторингу додатків [Електронний ресурс] // New Relic. – Режим доступу: https://docs.newrelic.com/docs/new-relic-solutions/get-started/intro-new-relic/ (дата звернення 25.12.2024).

19. Керівництво зі створення тестів у TestRail [Електронний ресурс] // TestRail. – Режим доступу: https://www.testrail.com/blog/test-design-guide/ (дата звернення 26.12.2024).

20. Навчальний посібник з використання SpecFlow [Електронний ресурс] // ToolsQA. – Режим доступу: https://www.toolsqa.com/specflow/specflow-tutorial/ (дата звернення 27.12.2024)

21. Методологія Agile: основи та огляд [Електронний ресурс] // Zenkit. – Режим доступу: https://zenkit.com/en/blog/agile-methodology-an-overview/ (дата звернення 28.12.2024).

22. Що таке Scratch: основи програмування для початківців [Електронний ресурс] // Create Learn. – Режим доступу: https://www.create-learn.us/blog/what-is-scratch-programming/ (дата звернення 29.12.2024).

23. Методологія IDEF: моделювання бізнес-процесів [Електронний ресурс] // Max Zosim. – Режим доступу: https://www.maxzosim.com/idef/ (дата звернення 30.12.2024).