НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

«На правах рукопису»
УДК 004.054

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
«___»_____2024 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Система автоматизації оновлення тестових сценаріїв ПЗ»

Виконав (-ла):
студент (-ка) ІІ курсу, групи ІП-32мп
Ярощенко Назар Вікторович _____

Керівник:
Доцент, к.ф.-м.н, доц.,
Поперешняк Світлана Володимирівна _____

Рецензент:
Заступник завідувача відділу АСОУ
Істітут програмних систем НАНУ, к.т.н., с.н.с.,
Ігнатенко Петро Петрович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2024 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення інформаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Едуард ЖАРІКОВ

«___»_____2024р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Ярощенко Назар Вікторович

1. Тема дисертації «Система автоматизації оновлення тестових сценаріїв», науковий керівник дисертації Поперешняк Світлана Володимирівна, к.ф.-м.н, доц., затверджені наказом по університету від «08» листопада 2024 р. № 5016-с

2. Термін подання студентом дисертації «9» грудня 2024 р.

3. Об’єкт дослідження – процеси тестування програмного забезпечення, зокрема управління тестовими сценаріями та їх підтримка в актуальному стані.

4. Предмет дослідження – методи та інструменти автоматичного оновлення тестових сценаріїв у процесі розробки програмного забезпечення.

5. Перелік завдань, які потрібно розробити – аналіз проблеми та існуючих рішень; розробка архітектури програмного забезпечення; розробка прототипу; оцінка ефективності за критеріями точності та швидкості оновлення тестів.

6. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7.Дата видачі завдання «1» вересня 2024 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Ознайомлення з предметною галуззю	1.09.2024	
2	Визначення структури магістерської дисертації; пошук та вивчення літератури	15.09.2024	
3	Робота над першим розділом магістерської дисертації	1.10.2024	
4	Робота над другим розділом магістерської дисертації	15.10.2024	
5	Робота над третім розділом магістерської дисертації	1.11.2024	
6	Робота над третім розділом магістерської дисертації	10.11.2024	
7	Виконання експериментальних досліджень	15.11.2024	
8	Оформлення пояснювальної записки	20.11.2024	
9	Подання дисертації на попередній захист	22.11.2024	
10	Подання дисертації на захист	16.12.2024	

Студент Назар ЯРОЩЕНКО

Науковий керівник Світлана ПОПЕРЕШНЯК

РЕФЕРАТ

Розмір пояснювальної записки – 117 аркушів, містить 10 ілюстрацій, 27 таблиць, 2 додатки, 18 посилань на джерела.

Актуальність теми. У роботі розглянуто проблему в автоматизації тестування програмного забезпечення, зокрема, у контексті забезпечення якості та продуктивності тестування складних систем. Показано основні особливості існуючих рішень, їх переваги, такі як забезпечення своєчасної адаптації тестів до змін у коді, зниження ризику пропуску дефектів і підвищення продуктивності розробки, а також недоліки, зокрема високу початкову вартість впровадження. Виявлено потребу в удосконаленні підходів до автоматизації, що дозволить ефективніше покривати складні системи, зменшувати витрати на підтримку тестового середовища і підвищувати економічну ефективність у довгостроковій перспективі.

Мета дослідження. Основною метою є скоротити час необхідний для оновлення тестових сценаріїв автоматизучи цей процес шляхом розробки методів і засобів автоматичного виявлення змін у програмному коді та відповідного коригування тестових сценаріїв.

Об’єкт дослідження: процеси тестування програмного забезпечення, зокрема управління тестовими сценаріями та їх підтримка в актуальному стані.

Предмет дослідження: методи та інструменти автоматичного оновлення тестових сценаріїв у процесі розробки програмного забезпечення.

Для реалізації поставленої мети **сформульовані наступні завдання**:

- Аналіз існуючих інструментів;
- Розробити архітектуру системи;
- Реалізувати прототип системи;
- Оцінити ефективність за критеріями точності та швидкості оновлення тестів.

Наукова новизна результатів магістерської дисертації полягає в тому, що запропоновано архітектурне рішення для автоматизації тестування програмного забезпечення, яке, на відміну від існуючих підходів, забезпечує своєчасну

адаптацію тестів до змін у коді, знижує ризик пропуску дефектів та підвищує продуктивність команд розробки. Результат досягнутий шляхом розробки модернізованого алгоритму, що оптимізує процес оновлення тестів, мінімізуючи витрати часу та зусиль, необхідних для підтримки актуальності тестового середовища.

Практичне значення отриманих результатів полягає в тому, що запропоновані методи автоматизації тестування інтегровані в межах єдиної системи, яка забезпечує простоту використання для користувача. Розроблена система може бути використана в проєктах з розробки складного програмного забезпечення, де потрібна висока гнучкість, адаптивність тестового середовища та зниження витрат на підтримку якості.

Зв'язок з науковими програмами, планами, темами. Робота виконувалась на кафедрі інформатики та програмної інженерії Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського".

Дослідження виконувалося в рамках НДР "Теоретичні та практичні аспекти технології Internet of Everything" з державним реєстраційним номером: 0123U104930

Апробація. Наукові положення дисертації пройшли апробацію на II міжнародній науково-практичній конференції молодих вчених та студентів, 19 - 21 грудня 2024 року, м. Київ, Державний університет інформаційно-комунікаційних технологій.

Публікації. Наукові положення дисертації опубліковані в:

1) Ярощенко Н. В., Система автоматизації оновлення тестових сценаріїв / Н.В. Ярощенко, С.В. Поперешняк // Матеріали II міжнародної науково-практичної конференції «Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії»

Ключові слова: ТЕСТУВАННЯ, ОНОВЛЕННЯ, АВТОМАТИЗАЦІЯ.

ABSTRACT

Explanatory note size – 117 pages, contains 10 illustrations, 27 tables, 2 applications, 18 references.

Topicality. Examines the problem of software testing automation, particularly in ensuring quality and productivity in testing complex systems. The main features of existing solutions, their advantages, such as timely adaptation of tests to code changes, reduction of defect omission risks, and increased development productivity, as well as disadvantages, such as high initial implementation costs, are analyzed. The need to improve automation approaches to effectively cover complex systems, reduce maintenance costs for the test environment, and enhance long-term economic efficiency is identified.

The aim of the study. The main target is to reduce the time required for updating test scenarios by automating the process through developing methods and tools for automatic detection of changes in program code and the corresponding adjustment of test scenarios.

The object of research: software testing processes, particularly the management and maintenance of test scenarios in an up-to-date state.

The subject of research: methods and tools for automatic updating of test scenarios during the software development process.

To achieve this goal, the **following tasks** were formulated:

- analyze existing tools;
- develop the system architecture;
- implement a system prototype;
- Evaluate efficiency based on criteria such as accuracy and speed of test updates.

The scientific novelty of the results of the master's dissertation is the proposed architectural solution for software testing automation, which, unlike existing approaches, ensures timely adaptation of tests to code changes, reduces the risk of defect omission, and enhances the productivity of development teams. The result is achieved through the development of a modernized algorithm that optimizes the

process of updating tests, minimizing time and effort required for maintaining the test environment's relevance.

The practical value of the obtained results is that the proposed methods of testing automation are integrated into a single system that ensures ease of use for the user. The developed system can be applied in projects involving the development of complex software, requiring high flexibility, adaptability of the test environment, and reduced maintenance costs.

Relationship with working with scientific programs, plans, topics. Work was performed at the Department of Informatics and Software Engineering of the National Technical University of Ukraine «Kyiv Polytechnic Institute. Igor Sikorsky».

The research was conducted within the framework of the R&D project "Theoretical and Practical Aspects of Internet of Everything Technology" with the state registration number: 0123U104930.

Approbation. The scientific propositions of the dissertation were tested at the II International Scientific and Practical Conference of Young Scientists and Students, held on December 19-21, 2024, in Kyiv, at the State University of Telecommunications.

Publications. The scientific provisions of the dissertation were published in:

1) Yaroshenko N. V., System for Automation of Test Scenario Updates / N. V. Yaroshenko, S. V. Popereshnyak // Proceedings of the II International Scientific and Practical Conference "Modern Aspects of Digitalization and Informatization in Software and Computer Engineering."

Keywords: TESTING, AUTOMATION, CODE UPDATES, EFFICIENCY.

Пояснювальна записка

до магістерської дисертації

на тему: ***Система автоматизації оновлення тестових сценаріїв***

виконав студент групи ІП-32мп

Ярощенко Назар Вікторович

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення;

API (application programming interface) – програмний інтерфейс;

AST (Abstract Syntax Tree) – абстрактні синтаксичні дерева;

ML (Machine Learning) – машинне навчання;

AI (Artificial intelligence) – штучний інтелект;

ОП – оперативна пам’ять;

ЦП – центральний процесор.

ВСТУП

Зростання складності процесів забезпечення якості програмного забезпечення у сучасних умовах стрімкого розвитку інформаційних технологій вимагає впровадження новітніх підходів до оновлення тестових сценаріїв. Тестові сценарії відіграють центральну роль у процесі перевірки програмного забезпечення, адже їх актуальність є важливим фактором для забезпечення стабільності та якості ПЗ як на етапі розробки, так і після випуску продукту. Постійні зміни у програмному коді, обумовлені оновленнями функціоналу, виправленням дефектів та адаптацією до нових вимог користувачів, створюють значний виклик для команд тестування. Саме тому необхідно впроваджувати сучасні системи для автоматизації оновлення тестів.

Традиційні методи підтримки тестових сценаріїв, засновані на ручному оновленні або використанні статичних тестових фреймворків, часто не враховують усю динаміку змін у програмному коді та потребують значних ресурсів для підтримки актуальності. У зв'язку з цим виникає потреба в новітніх аналітичних підходах, які здатні автоматично адаптувати тестові сценарії до змін і забезпечувати їх релевантність навіть у швидкозмінному середовищі. Сучасні інформаційні технології, зокрема методи статичного аналізу коду та алгоритми автоматизації, пропонують більш ефективні рішення, що дозволяють автоматизувати оновлення тестів, знижуючи витрати часу та людських ресурсів.

Алгоритми автоматизації є потужним інструментом для адаптації тестової бази завдяки своїй здатності знаходити залежності між змінами в коді та відповідними тестовими сценаріями. Вони дозволяють автоматично виявляти шаблони змін та оновлювати тести на основі цих закономірностей. Це робить автоматизацію особливо привабливою для роботи з великими проектами, де зміни коду можуть впливати на десятки або сотні тестів, що потребують коригування.

Метою цієї роботи є скорочення часу необхідного для оновлення тестових сценаріїв автоматизучи цей процес шляхом розробки методів і засобів автоматичного виявлення змін у програмному коді та відповідного коригування

тестових сценаріїв. У ході дослідження аналізуються існуючі інструменти автоматизації, розробляється архітектура системи автоматичного оновлення тестів, реалізується її прототип та оцінюється ефективність за критеріями точності оновлення та часу, необхідного для коригування тестів.

У процесі виконання роботи використовувалися наукові методи, зокрема статичний аналіз коду, автоматичне виявлення змін у програмному забезпеченні, побудова залежностей між кодом і тестами, а також проектування програмного забезпечення. Розроблений прототип включає модуль аналізу коду, модуль оновлення тестових сценаріїв та інтерфейс для візуалізації результатів і валідації змін.

Об’єктом дослідження є процеси тестування програмного забезпечення, зокрема управління тестовими сценаріями та їх підтримка в актуальному стані.

Предметом достідження є методи та інструменти автоматичного оновлення тестових сценаріїв у процесі розробки програмного забезпечення.

Структура роботи передбачає послідовний розгляд теоретичних засад автоматизації тестування, аналіз існуючих інструментів і методів оновлення тестових сценаріїв, проектування архітектури системи, її практичну реалізацію та оцінку ефективності на реальних прикладах змін у програмному забезпеченні.

1 АНАЛІЗ ІСНУЮЧИХ ІНСТРУМЕНТІВ ТА ПІДХОДІВ ДО ОНОВЛЕННЯ ТЕСТОВИХ СЦЕНАРІЇВ

У сучасних умовах швидкої еволюції програмного забезпечення та впровадження методологій забезпечення якості ПЗ відіграє ключову роль у розробці. Однією з найбільш складних задач у цьому процесі є підтримка актуальності тестових сценаріїв, особливо у великих проєктах із великою кількістю змін у коді. Оновлення тестів часто стає вузьким місцем через значні витрати часу, складність та ризик виникнення помилок у ручному режимі.

Існуючі інструменти автоматизації тестування здебільшого спрямовані на виконання тестів, проте їхній функціонал для оновлення тестових сценаріїв є обмеженим. У більшості випадків оновлення тестів покладається на ручну працю, що ускладнює масштабування процесу тестування та впливає на швидкість доставки програмного забезпечення.

Метою цього розділу є аналіз існуючих інструментів автоматизації тестування та підходів до оновлення тестових сценаріїв. Це дозволить виявити їхні переваги, обмеження та сформулювати вимоги до розробки системи автоматизації, яка здатна ефективно оновлювати тестові сценарії, адаптуючись до змін у коді. У розділі також будуть досліджені сучасні методи автоматизації та виявлені основні недоліки традиційних підходів, що визначить напрям подальшого розвитку в цій галузі.

1.1 Сучасні методи автоматизації тестування

Зі збільшенням складності програмних систем і частотою оновлень ефективні автоматизовані методи тестування є ключем до підтримання стабільності та якості. Основними напрямами сучасної автоматизації тестування є застосування методів машинного навчання, хмарних обчислень, аналізу змін коду та технологій самовідновлення тестів:

— динамічний аналіз коду: одним із найбільш перспективних методів є динамічний аналіз коду[1], який дозволяє автоматично виявляти змінені компоненти програмного забезпечення та зв’язані з ними тестові сценарії. Цей підхід базується на побудові залежностей між кодом і тестами, що дозволяє

ефективно виділяти лише ті тести, які потребують оновлення. Наприклад, використання техніки "Trace Analysis" дозволяє відстежувати виклики методів і модулів, пов’язаних із конкретною функціональністю;

— хмарні технології: марні технології[2] в автоматизації тестування дозволяють масштабувати процеси тестування, забезпечуючи виконання тестів у паралельному режимі на різних конфігураціях. Використання хмарних платформ знижує витрати на утримання інфраструктури, оскільки ресурси надаються за запитом. Це особливо корисно для проєктів із різними середовищами розгортання. На графіку нижче представлено порівняння традиційного локального тестування та хмарного, яке демонструє переваги в масштабованості й економічності;

— самовідновлення тестів: методи самовідновлення тестів[3] базуються на алгоритмах машинного навчання, які дозволяють автоматично адаптувати тестові сценарії до змін у коді. Наприклад, у разі зміни структури DOM-елементів веб-додатку ці алгоритми можуть автоматично ідентифікувати нові елементи і коригувати відповідні тестові сценарії. Це значно знижує необхідність у ручному втручанні. Нижче наведено діаграму, яка ілюструє процес самовідновлення тестів від моменту виявлення зміни до оновлення тестового сценарію.

Для порівняння сучасних і традиційних методів тестування наведено таблицю ключових характеристи.

Таблиця 1.1 – Порівняння традиційних та сучасгих методів тестування

Критерій	Традиційні методи	Сучасні методи
Адаптивність до змін	Вимагають ручного оновлення тестів після змін у коді.	Автоматично адаптують тести за допомогою методів аналізу змін та самовідновлення.

Продовження таблиці 1.1

Критерій	Традиційні методи	Сучасні методи
Швидкість тестування	Обмежена локальною інфраструктурою.	Використання хмарних технологій прискорює виконання тестів.
Витрати на підтримку	Високі витрати через необхідність постійного ручного оновлення.	Зниження витрат завдяки автоматизації оновлення.
Точність виявлення помилок	Залежить від людського фактора.	Використання алгоритмів машинного навчання підвищує точність і швидкість.
Масштабованість	Обмежена локальними ресурсами.	Хмарні платформи забезпечують тестування на різних середовищах і пристроях.

1.2 Огляд інструментів автоматизації тестування

Автоматизація тестування програмного забезпечення включає широкий спектр інструментів, кожен із яких орієнтований на виконання певних завдань. У контексті підтримки актуальності тестових сценаріїв важливим є не лише здатність виконувати автоматизовані тести, а й можливість ефективного оновлення тестів у разі змін у програмному коді. У цьому розділі проведено аналіз ключових інструментів автоматизації тестування із зазначенням їхніх можливостей та обмежень у контексті оновлення тестів.

Selenium[4] є одним із найбільш поширених інструментів для автоматизації веб-додатків. Він підтримує багато мов програмування (Java, Python, C#) і дозволяє створювати тести для різних браузерів. Основною перевагою Selenium є його відкритий код, що робить інструмент доступним для

широкого кола користувачів. Проте оновлення тестових сценаріїв у Selenium вимагає значних ручних зусиль, а також не передбачає автоматичного обґрунтування змін у тестах, що обмежує його використання у великих динамічних проєктах.

Cypress[5] орієнтований на тестування фронтенд-додатків і надає інтегроване середовище для розробки тестів. Його особливістю є швидке виконання сценаріїв і зручний інтерфейс для розробників, який забезпечує візуалізацію результатів тестування у реальному часі. Проте, подібно до Selenium, Cypress не має функцій автоматичного оновлення тестів чи обґрунтування внесених змін, що може створювати труднощі для проєктів із частими оновленнями коду.

TestComplete[6] пропонує функціонал для тестування різних типів додатків: десктопних, мобільних і веб. Основною перевагою цього інструмента є функція запису та відтворення тестів, яка дозволяє швидко створювати сценарії. Інструмент також підтримує візуальне тестування, що спрощує перевірку графічних елементів додатків. Однак TestComplete не підтримує автоматичного оновлення тестів, а його висока вартість ліцензії може бути бар’єром для малих проєктів.

Appium[7] спеціалізується на автоматизації мобільного тестування для платформ iOS та Android. Цей інструмент підтримує кросплатформенну розробку тестів, що робить його зручним для багатоплатформових проєктів. Проте Appium, як і Selenium, покладається на ручне оновлення тестів, що обмежує його ефективність у проєктах із динамічними змінами. Відсутність функції обґрунтування оновлень також є недоліком цього інструмента.

Для узагальнення функціональних можливостей розглянутих інструментів наведено таблицю.

Таблиця 1.2 – Можливості сучасних рішень

Функція	Selenium	Cypress	TestComplete	Appium
Автоматизація тестування веб-додатків	+	+	+	-
Автоматизація мобільних тестів	-	-	+	+
Запис і відтворення тестів	-	-	+	-
Самовідновлення тестів	-	-	-	-
Тестування фронтенду	+	+	+	-
Тестування десктопних додатків	-	-	+	-
Використання штучного інтелекту	-	-	-	-
Адаптація тестів до змін у коді	-	-	-	-
Обґрунтування оновлених тестів	-	-	-	-

1.3 Виявлення обмежень існуючих підходів

Попри значний прогрес у сфері автоматизації тестування, існуючі підходи мають низку суттєвих обмежень, які знижують їх ефективність у підтримці великих і динамічних проєктів. Аналіз існуючих інструментів та методів показує, що більшість з популярних рішень не забезпечує повного циклу аналізу змін у програмному коді, автоматичного оновлення тестових сценаріїв та їх обґрунтування. Ці недоліки створюють значні труднощі для команд тестування, особливо у випадках, коли зміни в коді є частими та масштабними.

Одним із ключових обмежень є відсутність автоматичного аналізу залежностей між змінами в коді та тестами. Більшість сучасних інструментів вимагають ручного оновлення тестових сценаріїв після внесення змін у програмний код, що є трудомістким і ресурсозатратним процесом. Наприклад,

при зміні методів або модулів тестувальникам доводиться вручну визначати, які тести стали неактуальними або потребують коригування.

Друге суттєве обмеження полягає у відсутності функції класифікації тестів. Сучасні рішення не дозволяють автоматично визначати, які тести залишаються релевантними, а які потребують оновлення. Це призводить до зайвого витрачання часу на аналіз усієї тестової бази, навіть якщо більшість тестів не зазнали впливу змін у коді.

Ще одним важливим недоліком є неможливість автоматичного генерування оновлених тестів із поясненням причин внесення змін. У багатьох інструментах, навіть якщо передбачена функція запису та відтворення, ці оновлення не включають пояснень, що ускладнює аудит тестів і перевірку їхньої релевантності. Це особливо важливо для великих команд, де необхідно забезпечити прозорість процесу тестування.

Іншим значущим обмеженням є відсутність комплексного підходу, який об'єднує аналіз коду, оновлення тестів та їх обґрунтування. Сучасні інструменти зазвичай орієнтовані на виконання лише однієї з цих задач, залишаючи значну частину роботи для ручного виконання. Це створює розрив між процесами розробки та тестування, що знижує ефективність усього циклу розробки програмного забезпечення.

Таким чином, існуючі підходи до автоматизації тестування виявляються недостатніми для забезпечення ефективності у проєктах із частими змінами. Проблема актуалізації тестової бази залишається однією з найбільших викликів, що потребує комплексного вирішення.

1.4 Постанока задачі

На основі проведеного аналізу сучасних методів та інструментів автоматизації тестування було сформульовано такі задачі, вирішення яких є необхідним для досягнення мети роботи:

— розробка методу автоматичного аналізу змін у програмному коді. Важливість цієї задачі зумовлена потребою в автоматизації визначення залежностей між змінами у коді та відповідними тестами. У сучасних підходах цей процес зазвичай здійснюється вручну, що є тривалим і ресурсозатратним, особливо в умовах великих проєктів із частими оновленнями. Результатом виконання цієї задачі має стати метод, який забезпечить автоматичний аналіз залежностей між модулями та тестовими сценаріями. Для реалізації цього методу планується використання статичного й динамічного аналізу коду, що дозволить побудувати точні зв’язки між компонентами системи. Це, у свою чергу, зменшить витрати часу та зусиль, необхідних для ідентифікації релевантних тестів;

— реалізація системи класифікації тестів. Актуальність цієї задачі визначається необхідністю оптимізації процесу тестування. У відсутності автоматичної класифікації тестів команди змушені переглядати весь тестовий набір, навіть якщо більшість сценаріїв залишаються релевантними. Очікуваним результатом є створення системи, яка автоматично класифікуватиме тести на релевантні та ті, що потребують оновлення. Для цього будуть використані алгоритми класифікації та аналіз залежностей між кодом і тестами. Виконання цієї задачі дозволить знизити навантаження на тестувальників і підвищити продуктивність процесу тестування;

— інтеграція механізму автоматичного оновлення тестових сценаріїв. Ручне оновлення тестів є одним із найбільших викликів для команд тестування, особливо у випадках масштабних змін у програмному коді. Метою цієї задачі є створення механізму, який забезпечить автоматичне генерування оновлених тестів відповідно до виявлених змін. Унікальність запропонованого підходу полягає у використанні API OpenAI[8] для генерації оновлених тестових сценаріїв на основі аналізу коду. Це рішення дозволить зменшити витрати на підтримку тестової бази та підвищити ефективність автоматизації тестування;

— забезпечення прозорості процесу оновлення тестів. Пояснення внесених змін у тестах є важливим для забезпечення прозорості та можливості

аудиту тестової бази. Сучасні інструменти не пропонують автоматичного обґрунтування змін, що ускладнює перевірку актуальності тестів. Завдання полягає у створенні системи, яка забезпечуватиме генерацію пояснень для кожного оновленого тесту, включаючи причини внесення змін та їхній вплив на результати тестування. Це сприятиме підвищенню довіри до автоматизації та забезпеченню прозорості процесу тестування;

— оцінка ефективності розробленої системи. Ефективність запропонованої системи має бути підтверджена на основі реальних даних. Метою цієї задачі є оцінка точності, швидкості оновлення тестів і зниження витрат на їх підтримку. Для цього планується тестування системи на реальних та симуляційних проєктах різного масштабу. Проведення порівняльного аналізу до і після впровадження системи дозволить оцінити її вплив на продуктивність процесу тестування.

Вирішення зазначених задач дозволить створити комплексну систему автоматизації оновлення тестових сценаріїв, яка не лише оптимізує процес тестування, але й забезпечує його прозорість та ефективність. Використання інноваційних підходів, таких як штучний інтелект та автоматичний аналіз залежностей, сприятиме підвищенню якості та швидкості розробки програмного забезпечення.

1.5 Взаємозв'язок з іншими роботами

Розробка системи автоматизації оновлення тестових сценаріїв програмного забезпечення є логічним продовженням та вдосконаленням існуючих досліджень та практик у сфері тестування ПЗ. Ця робота базується на попередніх дослідженнях, які вивчали автоматизацію тестування, використання скриптів та інтелектуальних систем для управління тестовими сценаріями.

Схожі дослідження:

— "Software Testing Resource Scheduling based on Artificial Intelligence"[9] – дослідження, яке фокусується на методах автоматизації підтримки тестів в Agile

проектах. Воно аналізує існуючі підходи та пропонує нові методи для підвищення ефективності оновлення тестових сценарії;

— "Machine Learning for Test Case Prioritization and Selection"[10] – дослідження, яке вивчає застосування алгоритмів машинного навчання для пріоритезації та вибору тестових сценаріїв. Воно демонструє, як ML може допомогти оптимізувати процес тестування, зменшуючи час та ресурси, необхідні для виконання тестів.

Відмінності та новизна роботи: на відміну від попередніх досліджень, ця робота спрямована на створення комплексної системи, яка поєднує простоту скриптів з інтелектуальними можливостями штучного інтелекту для автоматичного оновлення тестових сценаріїв.

Інтеграція з існуючими дослідженнями: ця робота інтегрує методи автоматизації тестування з сучасними технологіями штучного інтелекту, пропонуючи новий підхід до управління тестовими сценаріями. Вона спирається на існуючі дослідження з автоматизації тестів та використання ML для подальшого вдосконалення процесів тестування, забезпечуючи більш гнучкий та ефективний підхід до управління тестовими сценаріями в різних проектах.

Таблиця 1.3 – Відмінності та новизна роботи порівняно з попередніми дослідженнями

Дослідження	Основний фокус	Відмінності/Новизна
Automated Test Maintenance in Agile	Автоматизація підтримки тестів в Agile проектах	Комбінування скриптів з AI для інтелектуального оновлення тестів
Machine Learning for Test Case Prioritization	Використання ML для пріоритезації тестів	Використання OpenAI API для генерації оновлених тестів

Висновки

У цьому розділі було здійснено аналіз сучасних методів і інструментів автоматизації тестування, визначено їхні обмеження та запропоновано обґрунтування актуальності розробки системи автоматизації оновлення тестових сценаріїв.

Аналіз існуючих методів автоматизації показав, що попри широкий вибір технологій, більшість із них зосереджені на виконанні тестів, але не забезпечують ефективної підтримки їх актуальності. Методи, засновані на ручному оновленні тестових сценаріїв, виявилися недостатніми для великих і динамічних проєктів. Відсутність функцій автоматичного аналізу змін у коді, класифікації тестів та автоматизованого обґрунтування внесених змін створює значні виклики для сучасних підходів.

Розглянуті інструменти, такі як Selenium, Cypress, TestComplete та Appium, демонструють сильні сторони в різних аспектах автоматизації тестування. Однак жоден із них не забезпечує повного циклу оновлення тестів із врахуванням аналізу змін у коді та автоматичної генерації оновлених сценаріїв. Це підкреслює необхідність розробки комплексного рішення, яке інтегрує інтелектуальні методи аналізу й автоматизації оновлення тестів.

На основі проведеного аналізу було виявлено, що існуючі підходи обмежені в таких аспектах:

— відсутність автоматизованого аналізу залежностей між змінами в коді та тестами;

— відсутність класифікації тестів на релевантні та такі, що потребують оновлення;

— неможливість автоматичного генерування оновлених тестів із поясненням змін;

— відсутність прозорого процесу обґрунтування оновлень.

Таким чином, результати цього розділу обґрунтовують необхідність і актуальність розробки комплексної системи, яка інтегрує сучасні підходи до автоматизації тестування з новітніми технологіями штучного інтелекту.

2 РОЗРОБКА АРХІТЕКТУРИ

Проєктування системи автоматизації оновлення тестових сценаріїв є важливим етапом роботи, який закладає основу для її подальшої реалізації. У цьому розділі здійснюється розробка концептуальної моделі системи, визначаються основні компоненти її архітектури та описуються ключові алгоритми роботи.

Зважаючи на виявлені обмеження існуючих підходів і потреби динамічних проєктів, запропонована система базується на принципах модульності, прозорості, масштабованості та ефективності. Особливу увагу приділено інтеграції механізмів аналізу змін у коді, класифікації тестів на релевантні та такі, що потребують оновлення, а також автоматичної генерації оновлених тестів із поясненням внесених змін.

У рамках розділу детально описується архітектура системи, проєктуються алгоритми її роботи та моделюються основні процеси. Особливий акцент зроблено на використанні сучасних підходів до інженерії програмного забезпечення, таких як розробка діаграм компонентів, потоків даних і алгоритмів. Це забезпечує чітке уявлення про функціонування системи та дозволяє створити основу для її подальшої реалізації.

2.1 Принципи проєктування системи

Проєктування системи автоматизації оновлення тестових сценаріїв базується на п’яти ключових принципах[11]: модульності, прозорості, масштабованості, простоти та ефективності. Ці принципи спрямовані на створення гнучкого, адаптивного та зручного програмного забезпечення, яке забезпечує автоматизацію складних процесів і підвищення ефективності роботи з тестовими сценаріями.

2.1.1 Модульність

Модульність є основним принципом, який забезпечує структурований підхід до проєктування системи. Поділ системи на окремі модулі дає змогу:

— ізолювати функціональність кожного компонента;
— зменшити вплив змін в одному модулі на інші;
— спростити підтримку та модернізацію системи.

У контексті системи автоматизації оновлення тестів модульність забезпечує локалізацію основних функцій, таких як аналіз змін у коді, класифікація тестів та генерація оновлених сценаріїв. Це дозволяє швидко інтегрувати нові функції або адаптувати систему до змін у вимогах.

2.1.2 Прозорість

Прозорість є важливим принципом, який забезпечує зрозумілість процесів оновлення тестів. Система автоматично генерує пояснення для кожного оновленого тесту, що включає:

— опис змін у програмному коді;
— причину, через яку тест потребував оновлення;
— логічне обґрунтування внесених змін.

Це спрощує аудит тестів і сприяє кращому розумінню роботи системи розробниками та тестувальниками. Прозорість також підвищує довіру до результатів автоматизації та дозволяє уникнути суперечок у командній роботі.

2.1.3 Масштабованість

Масштабованість забезпечує здатність системи адаптуватися до роботи з великими обсягами даних. Для цього застосовуються оптимізовані алгоритми, які дозволяють обробляти значні набори тестових сценаріїв із високою швидкістю. Масштабованість системи дозволяє:

— адаптуватися до змін у розмірах тестових наборів;

— працювати з різними проєктами незалежно від їхнього масштабу;
— забезпечувати високу продуктивність навіть у великих інфраструктурах.

Ефективна структура даних і оптимізація алгоритмів роблять систему придатною для використання в умовах високих навантажень.

2.1.4 Простота та зручність

Простота та зручність є критично важливими для забезпечення ефективного використання системи. Цей принцип реалізується через:

— інтуїтивно зрозумілий інтерфейс, який дозволяє користувачам швидко освоїти функціонал;
— автоматизацію складних процесів, таких як класифікація тестів та аналіз змін;
— документацію, яка допомагає користувачам із різним рівнем технічної підготовки працювати із системою.

Це робить систему доступною для широкого кола користувачів і сприяє її швидкому впровадженню в проєкти різного масштабу.

2.1.5 Ефективність

Ефективність забезпечує скорочення часу та ресурсів, необхідних для підтримки тестової бази. Основними аспектами ефективності є:

— використання алгоритмів для автоматичного аналізу змін у коді;
— автоматична класифікація тестів, що дозволяє зосередитися на релевантних оновленнях;
— генерація оновлених сценаріїв, що знижує обсяг ручної роботи.

Це дозволяє оптимізувати процеси тестування та забезпечити високу продуктивність роботи системи.

2.2 Вибір архітектури системи

Вибір архітектури для системи автоматизації оновлення тестових сценаріїв є важливим етапом, що визначає загальну структуру програми та спосіб взаємодії між її компонентами. Для даної системи було обрано монолітну архітектуру, яка є оптимальним варіантом з огляду на простоту реалізації, ефективність та зручність у підтримці. Монолітна архітектура дозволяє інтегрувати всі компоненти в єдину програму, що зменшує складність і забезпечує високу продуктивність при роботі з тестовими сценаріями на одному комп'ютері користувача.

2.2.1 Обґрунтування вибору монолітної архітектури

Монолітна архітектура[12] була вибрана на основі кількох важливих факторів:

— простота реалізації та підтримки: всі компоненти системи інтегровані в одну програму, що спрощує процес розробки, тестування та подальшого обслуговування. Для даної системи, що виконується на одному комп'ютері, монолітна архітектура дозволяє уникнути необхідності в складних налаштуваннях розподілених сервісів чи серверної інфраструктури;

— продуктивність: оскільки система працює локально на комп’ютері користувача, відсутність необхідності в комунікації між різними сервісами чи серверами забезпечує високу швидкість обробки даних і мінімізацію часу на виконання операцій. Це важливо для забезпечення ефективної роботи системи без затримок;

— легкість в управлінні та інтеграції: використання монолітного підходу дозволяє зберегти всю логіку обробки даних і взаємодії між модулями в єдиному середовищі, що спрощує інтеграцію функціональності і знижує ймовірність виникнення помилок при взаємодії компонентів. Це важливо для зручності розробки і подальших змін у системі;

— масштабованість: хоча монолітна архітектура традиційно не має такої гнучкості, як мікросервісна, для цієї конкретної системи, де всі процеси відбуваються на одному комп’ютері користувача, вибір моноліту є оптимальним. Система здатна ефективно працювати з середнім обсягом даних і легко адаптується до змін.

2.2.2 Структура архітектури системи

Архітектура монолітної системи складається з кількох ключових компонентів, які взаємодіють в межах єдиного середовища. Ось основні елементи архітектури:

— модуль аналізу змін у коді: виявляє зміни, які були внесені до програмного коду, і визначає відповідні частини тестової бази, що потребують оновлення. Модуль здійснює статичний і динамічний аналіз коду, що дозволяє ефективно виявляти зміни;

— модуль класифікації тестів: отримує тести, що потребують оновлення, від модуля аналізу і класифікує їх за принципом актуальності. Використовуються алгоритми машинного навчання, для визначення, чи потребує тест оновлення;

— модуль генерації оновлених тестів: генерує нові тестові сценарії на основі класифікації тестів та виявлених змін у коді. Модуль також автоматично створює пояснення до кожного тесту, що дозволяє зрозуміти, чому та як були внесені зміни;

— центральна логіка обробки даних: забезпечує координацію між усіма модулями системи. Вона відповідає за обробку та зберігання результатів, а також керує взаємодією компонентів, що дозволяє забезпечити безперервний потік даних у межах системи.

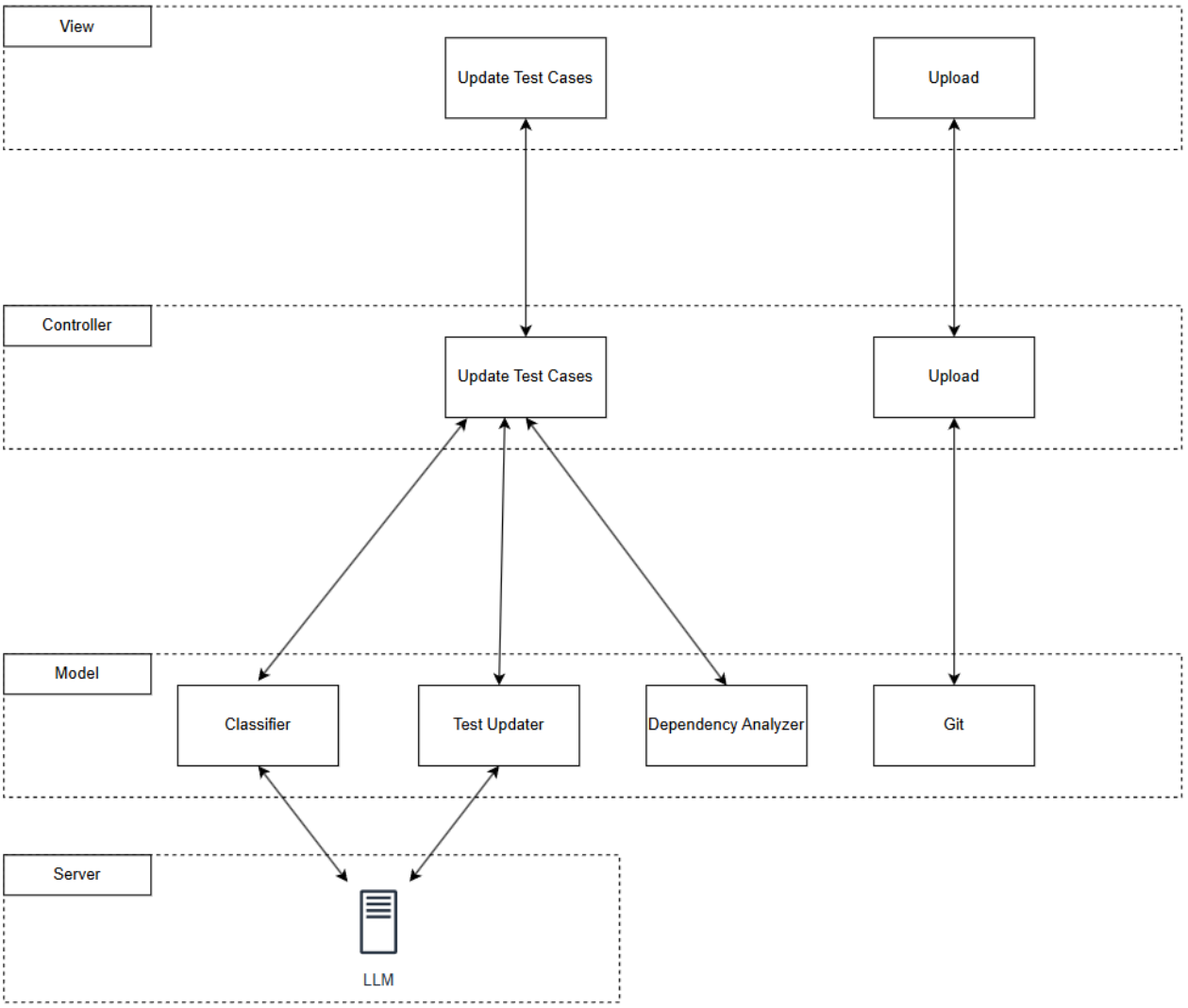

Рисунок 2.1 – Архітектура монолітної системи автоматизації оновлення тестів

2.3 Взаємодія компонентів

У рамках монолітної архітектури системи автоматизації оновлення тестових сценаріїв, взаємодія між основними компонентами є важливою складовою для забезпечення ефективної та безперебійної роботи системи. Кожен компонент виконує свою функцію, і їхня взаємодія повинна бути чітко організована для того, щоб забезпечити коректне оновлення тестових сценаріїв на основі змін у коді.

Центральним елементом взаємодії є центральна логіка обробки даних, яка координаціює роботу всіх модулів. Всі інші компоненти — модуль аналізу змін

у коді, модуль класифікації тестів і модуль генерації оновлених тестів — є підсистемами, що виконують окремі етапи обробки, але без чіткої синхронізації між ними система не зможе працювати належним чином.

2.3.1 Алгоритм взаємодії компонентів

Взаємодія між компонентами відбувається через певні алгоритми, які можна розподілити на кілька етапів:

а) аналіз змін у коді:

1) першим етапом є обробка змін у програмному коді. модуль аналізу змін отримує дані про змінений код (наприклад, за допомогою інструментів для порівняння версій чи арі для інтеграції з системами контролю версій) і виявляє частини коду, що зазнали змін;

2) після цього визначаються відповідні тести, які повинні бути актуалізовані, або нові, які повинні бути створені;

б) класифікація тестів:

1) всі тести, виявлені на попередньому етапі, передаються в модуль класифікації тестів, де кожен тест оцінюється на предмет його актуальності;

2) класифікація здійснюється з використанням мовних моделей або інших подібних систем, які на основі контексту змін визначають, чи є тест застарілим або він все ще відповідає актуальним вимогам коду;

3) модуль класифікації генерує два основні результати: тести, що потребують оновлення та тести, що залишаються актуальними і не потребують змін;

в) генерація оновлених тестів:

1) тести, що потребують оновлення, передаються в модуль генерації оновлених тестів. цей модуль на основі змін у коді і виявлених застарілих тестів генерує нові версії тестових сценаріїв;

2) генерація тестів відбувається автоматично, а також додається пояснення до кожного оновленого тесту, що допомагає зрозуміти, які саме зміни були внесені;

3) пояснення є важливим елементом для документування змін і забезпечує прозорість для розробників і тестувальників;

г) збереження та передача результатів:

1) центральна логіка обробки даних відповідає за збереження результатів аналізу, класифікації та генерації тестів;

2) всі результати передаються в загальну базу даних або систему зберігання для подальшого використання або для подальшої обробки іншими модулями.

2.3.2 Комунікація між компонентами

У монолітній архітектурі всі компоненти працюють в одному середовищі, що дозволяє забезпечити ефективну та швидку комунікацію між ними. Комунікація між компонентами відбувається через функціональні виклики та передачу даних між модулями, без необхідності використання складних інтерфейсів або протоколів обміну даними, характерних для розподілених архітектур. Це дозволяє значно зменшити час на обробку запитів і покращити загальну продуктивність системи:

— інтерфейси між модулями: кожен модуль має визначений інтерфейс для взаємодії з іншими частинами системи, що дозволяє забезпечити стандартизовану передачу даних і дозволяє знизити складність інтеграції;

— обробка даних у центральному компоненті: центральна логіка виступає як посередник між модулями, отримуючи дані від аналізу змін, передаючи їх на класифікацію, а потім до генерації оновлень. Всі дані, які надходять від модулів, обробляються в єдиному форматі, що дозволяє зберегти узгодженість та стандартизацію в обробці інформації.

2.3.3 Візуалізація взаємодії компонентів

У разі необхідності для більш детального розуміння взаємодії між компонентами, можна побудувати діаграму взаємодії (sequence diagram), що наочно відображає передачу даних між модулями та кроки, які виконуються на кожному етапі:

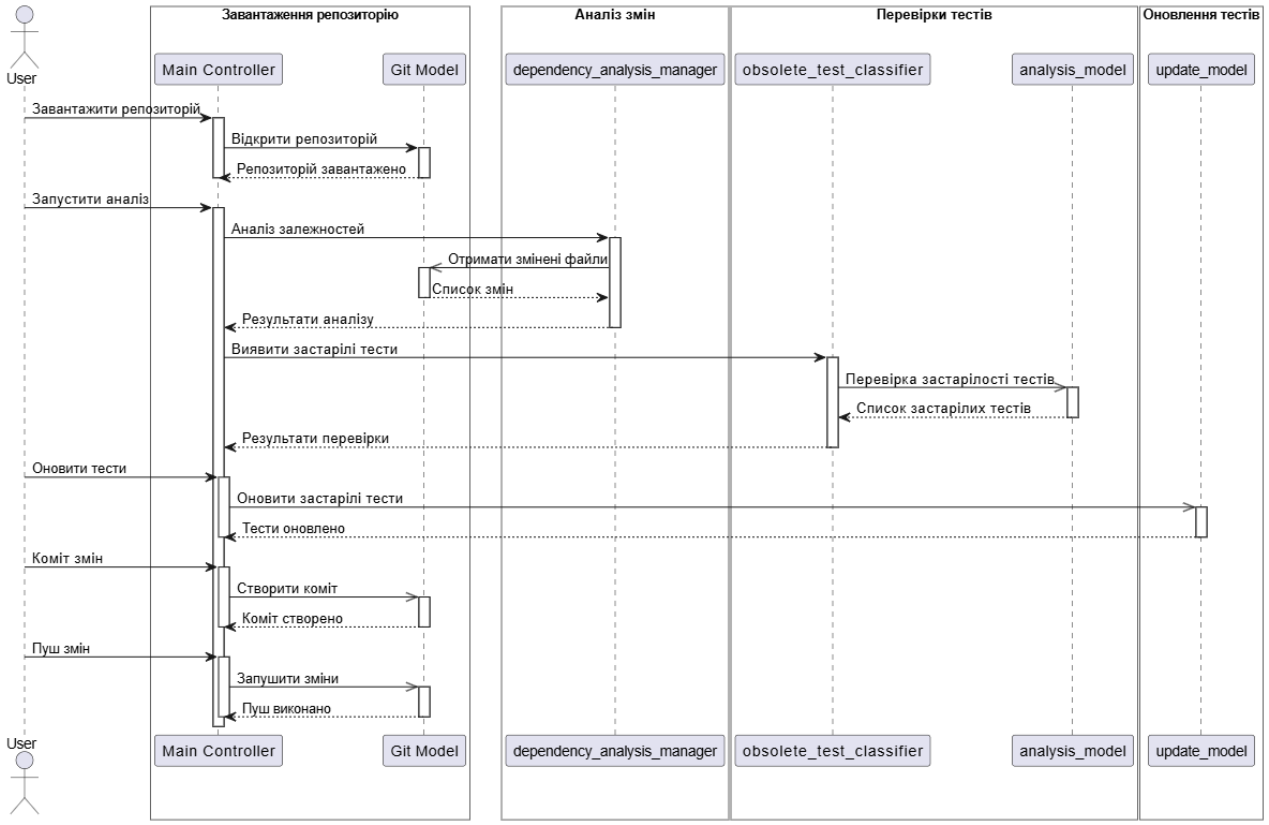

Рисунок 2.2 – Діаграма взаємодії компонентів системи автоматизації оновлення тестів

Висновки

У розділі було детально розглянуто архітектуру системи автоматизації оновлення тестових сценаріїв, зокрема обґрунтовано вибір монолітної архітектури як найбільш відповідної для цієї системи. Вибір монолітного підходу був мотивований простотою реалізації, високою продуктивністю при роботі в одному середовищі, а також зручністю в обслуговуванні та підтримці. Монолітна архітектура дозволяє централізовано інтегрувати всі основні функції

системи, що значно спрощує комунікацію між компонентами та забезпечує ефективність роботи системи в межах одного комп’ютера користувача.

Далі було описано взаємодію компонентів, що є ключовою для безперебійної роботи системи. Кожен компонент, від аналізу змін у коді до генерації оновлених тестів, виконує визначену функцію і взаємодіє з іншими через чітко визначену центральну логіку обробки даних. Завдяки такій організації компоненти ефективно обробляють змінений код і актуалізують тестові сценарії, що відповідають вимогам сучасних розробок.

У результаті системна архітектура була організована таким чином, щоб забезпечити високий рівень інтеграції та продуктивності при збереженні гнучкості та простоти обслуговування. Це дозволяє ефективно управляти процесами оновлення тестових сценаріїв, що є основним завданням цієї системи.

3 РОЗРОБКА І РЕАЛІЗАЦІЯ СИСТЕМИ

Розділ 3 присвячений детальному опису процесу розробки та реалізації системи автоматизації оновлення тестових сценаріїв. У цьому розділі буде розглянуто, як обрані технології та інструменти використовуються для створення ключових компонентів системи, як вони взаємодіють між собою, а також які методи були застосовані для забезпечення ефективності та точності роботи системи. Описаний підхід ґрунтується на попередніх принципах проєктування і виборі монолітної архітектури, що забезпечує інтеграцію всіх компонентів у єдине програмне середовище.

У даному розділі також розглядаються основні етапи реалізації: від вибору технологій та інструментів, що відповідають вимогам до ефективності, гнучкості та прозорості, до інтеграції всіх функцій системи в одну цілісну структуру. Оскільки система повинна автоматизувати процес оновлення тестів на основі змін у програмному коді, велика увага приділяється розробці алгоритмів для аналізу змін, класифікації тестів і генерації їх оновлених варіантів.

Метою цього розділу є детальне пояснення всіх етапів розробки, вибору технологій та інтеграції компонентів системи для досягнення ефективного і точного оновлення тестових сценаріїв в рамках заданої архітектури.

3.1 Вибір технологій та їх обґрунтування

3.1.1 Python

Python[13] — це високорівнева мова програмування, яка була розроблена в кінці 1980-х років Гвідо ван Россумом і випущена в 1991 році. Python став однією з найпопулярніших мов завдяки своїй простоті, гнучкості та широким можливостям для розробки програмного забезпечення. Мова наслідує багато принципів і конструкцій від C та C++, але при цьому вона має значно вищий рівень абстракції і спрощену модель управління пам’яттю. Python підтримує

об'єктно-орієнтоване, процедурне та функціональне програмування, що робить його універсальним інструментом для розробки різноманітних програмних рішень.

Основні переваги Python для розробки системи автоматизації оновлення тестових сценаріїв включають:

— простота у використанні: Python має інтуїтивно зрозумілий синтаксис, що дозволяє швидко розробляти ефективні програми без необхідності глибокого вивчення складних концепцій, таких як вказівники чи перевантаження операторів, що є в інших мовах, таких як C++;

— масштабованість та гнучкість: Python дозволяє розробляти як маленькі скрипти для автоматизації задач, так і великі розподілені системи. Його використання в рамках системи автоматизації тестів дозволяє легко масштабувати та доповнювати функціональність у міру потреби;

— велика екосистема бібліотек: Python має багатий набір бібліотек для роботи з даними, виконання статичного та динамічного аналізу коду, інтеграції з Git[14] та API для взаємодії з іншими системами. Це дозволяє реалізувати складні алгоритми для аналізу змін у коді та генерації тестів без необхідності створення всіх інструментів з нуля;

— простота інтеграції з іншими системами: Python дозволяє інтегруватися з різними технологіями та платформами, що є важливою перевагою при розробці інструментів для автоматизації тестування, які можуть взаємодіяти з існуючими системами контролю версій та базами даних.

Для нашої системи, Python є ідеальним вибором завдяки своїм перевагам у швидкості розробки, простоті інтеграції з іншими інструментами та здатності до обробки великих обсягів даних. Мова забезпечує необхідну продуктивність для розв'язання поставлених завдань без необхідності застосування складних технічних рішень.

3.1.2 PyQt5

PyQt5[15] — це бібліотека для створення графічних інтерфейсів на Python, що базується на популярному фреймворку Qt. PyQt5 дозволяє швидко розробляти кросплатформенні програми з інтуїтивно зрозумілим інтерфейсом користувача, що робить її ідеальним вибором для реалізації інтерфейсу нашої системи автоматизації оновлення тестів. Використання PyQt5 забезпечує:

— кросплатформенність: програма, розроблена за допомогою PyQt5, може бути запущена на різних операційних системах, таких як Windows, Linux та macOS, що дозволяє системі працювати на будь-якому комп’ютері користувача без додаткових налаштувань;

— гнучкість у проектуванні інтерфейсу: PyQt5 надає велику кількість віджетів і можливостей для створення складних візуальних елементів, таких як панелі, кнопки, вікна повідомлень та інші компоненти інтерфейсу, що значно підвищує зручність взаємодії з користувачем;

— простота інтеграції: оскільки PyQt5 працює на Python, всі компоненти програми можуть бути інтегровані в одному середовищі, що спрощує розробку і тестування.

3.1.3 Git

Git — це система контролю версій, що дозволяє ефективно управляти змінами в програмному коді та забезпечує зручну роботу з репозиторіями. Для інтеграції з Git у нашій системі було обрано бібліотеку GitPython. Git дозволяє відслідковувати зміни у коді, що важливо для автоматичного виявлення тих частин коду, які потребують актуалізації тестів. Важливі переваги Git:

— контроль версій: Git дозволяє відслідковувати історію змін у коді, що забезпечує зручне управління версіями та допомагає виявити, які частини коду були змінені і які тести необхідно оновити;

— інтеграція з іншими інструментами: завдяки Git можна інтегрувати систему з іншими інструментами для аналізу коду та тестування, такими як CI/CD платформи або автоматизовані системи тестування.

3.1.4 OpenAI

ChatGPT — мовна модель, що використовує технології машинного навчання для аналізу текстових даних і генерування відповідей на основі заданого контексту. У нашій системі ChatGPT використовується для автоматичної класифікації тестів та генерації їх оновлених версій. Це дає змогу:

— автоматизувати процес класифікації: ChatGPT здатен аналізувати текст тестових сценаріїв і визначати, які з них потребують оновлення на основі змін у коді;

— генерація пояснень до тестів: мовна модель дозволяє створювати текстові пояснення для кожного тесту, що підвищує зрозумілість і прозорість змін для кінцевого користувача.

3.2 Опис реалізації основних компонентів

3.2.1 Модуль аналізу змін у коді

Модуль використовує механізм AST (Abstract Syntax Tree)[16] для аналізу коду Python та виявлення залежностей між функціями. Він є основним компонентом для витягання інформації про структуру коду, виявлення змін і побудови графу залежностей, що потім буде використано для оновлення тестових сценаріїв.

Алгоритм роботи**:**

а) витягнення функцій з коду: Модуль використовує бібліотеку ast, яка дозволяє перетворити текстовий код на абстрактне синтаксичне дерево, що відображає його структуру. За допомогою методу extract_functions, система витягує всі функції з коду. Це дозволяє отримати наступну інформацію для кожної функції:

1) назва функції;
2) перелік параметрів та їх типів (якщо вони визначені);

3) структура тіла функції, включаючи умовні конструкції, цикли та виклики інших функцій;

б) Виявлення залежностей: Для побудови графа залежностей між функціями, модуль використовує метод extract_dependencies. Цей метод знаходить всі виклики функцій в коді і визначає, які функції викликають інші. Зібрані дані дозволяють побудувати повний граф залежностей, де кожна функція має список функцій, які вона викликає;

в) аналіз структури тіла функцій: Для кращого розуміння функціональності кожної функції, метод analyze_body_structure аналізує основні конструкції тіла функцій, такі як умови (if), цикли (for, while), виклики інших функцій та інші оператори;

г) визначення залежностей між функціями та тестами: Для пошуку тестів, що стосуються змінених функцій, модуль використовує методи пошуку тестових файлів і аналізу залежностей між функціями та тестами за допомогою методів find_test_files та find_related_tests. Цей процес включає пошук усіх тестових файлів у репозиторії і перевірку, чи залежать тестові функції від змінених функцій.

Ключові критерії для аналізу:

— тип змін: визначення, чи були зміни в функціях, методах чи класах;

— залежності: оцінка залежностей між функціями, що дозволяє створити граф викликів;

— актуальність тестів: Визначення тестів, які повинні бути оновлені після змін у коді.

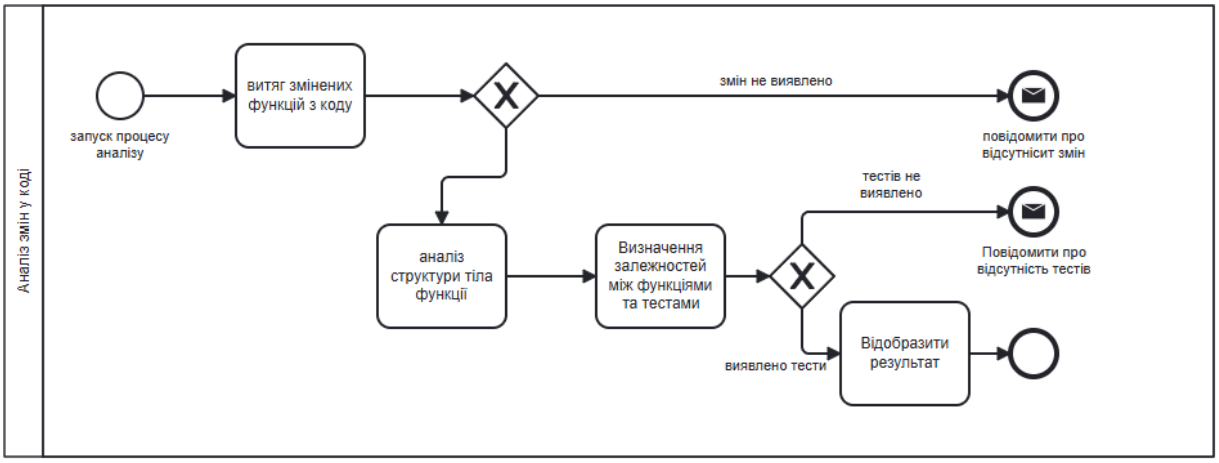

Рисунок 3.1 – Схема структурна діяльності процесу аналізу змін у коді

3.2.2 Модуль класифікації тестів

Модуль виконує функцію класифікації тестів, визначаючи, чи стали вони застарілими через зміни в програмному коді. Цей модуль є важливим етапом у процесі автоматизації оновлення тестів, оскільки дозволяє визначити, які з тестів потребують корекцій або повного оновлення після внесення змін у функції чи методи.

Алгоритм роботи:

— отримання тестового коду: першим кроком у процесі класифікації є витягнення тестового коду з відповідних тестових файлів. Для цього використовується метод _get_test_code, який завантажує і парсить тестові файли, виявляючи конкретні рядки коду, що стосуються кожного тесту;

— створення запиту до моделі OpenAI: на основі змін у коді та витягнутого тестового коду, модуль формує запит до моделі OpenAI, який складається з двох частин: системного повідомлення, яке пояснює роль моделі, і повідомлення від користувача, що містить інформацію про зміни у коді та пов'язану з ними інформацію про тести. Системне повідомлення містить чіткі вказівки для моделі, як визначати застарілі тести, орієнтуючись на такі критерії, як зміни у параметрах, логіці або зовнішніх ефектах функцій;

— аналіз відповіді від моделі: модель OpenAI на основі отриманого запиту оцінює, чи є тести застарілими. Відповідь від моделі надається у форматі JSON, де для кожного тесту зазначено, чи потребує він оновлення, а також пояснення (якщо потрібно) про причину застарілості;

— формування результатів: після отримання відповіді, метод _parse_response аналізує JSON-відповідь моделі та формує список тестів, що потребують оновлення. У цей список додається інформація про тест, стару та нову версію коду, а також причину необхідності оновлення.

Модуль класифікації тестів визначає, чи став тест застарілим, виходячи з таких критеріїв:

— зміни в параметрах або типах даних: якщо зміни у функціях включають зміну типів параметрів або їх додавання/видалення;

— зміни в бізнес-логіці: якщо змінена логіка функцій або методів, що змінює результати тестів;

— зміни у побічних ефектах: якщо зміни у коді викликають нові побічні ефекти, наприклад, зміни в стані програми або зовнішніх системах;

— зміни в структурі функцій: якщо функція була перейменована або змінена таким чином, що тест більше не може її правильно викликати.

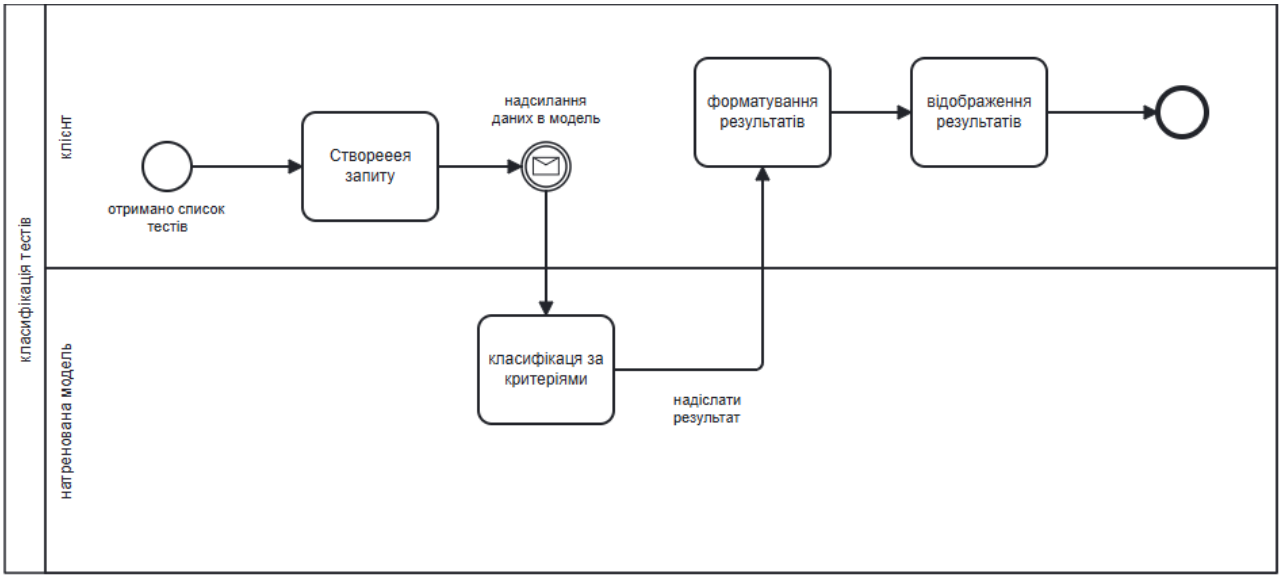

Рисунок 3.2 – Схема структурна діяльності процесу класифікації тестів

3.2.3 Модуль генерації оновлених тестів із поясненнями

Модуль генерації оновлених тестів із поясненнями виконує ключову роль в автоматизації оновлення тестових сценаріїв після внесення змін у програмний код. Цей модуль базується на глибокому аналізі функцій та їхніх залежностей за допомогою AST (Abstract Syntax Tree) і застосуванні інструментів штучного інтелекту, зокрема OpenAI для генерації відповідних тестів і пояснень до них.

Алгоритм роботи:

— аналіз змін у коді: модуль використовує функцію find_related_tests для пошуку тестів, що залежать від змінених функцій. Зміни в коді аналізуються через DependencyAnalyzer, який побудує граф залежностей між функціями та виявить ті з них, що були змінені;

— витягнення тестового коду: після виявлення змін у функціях модуль find_related_tests відшукує тестові файли в репозиторії, що мають відношення до цих змін. За допомогою методу extract_test_code_from_file зчитується код тестів, який далі передається на оцінку;

— генерація запиту до моделі OpenAI: зібрані дані (зміни в коді, старі та нові версії функцій) формують запит до OpenAI для створення оновлених тестів. Запит включає в себе конкретні інструкції для моделі, щоб визначити, чи тест потребує оновлення. Модель також генерує пояснення, чому конкретний тест більше не є актуальним, або чому його слід оновити;

— аналіз результатів відповіді: після того, як модель надає відповідь у форматі JSON, модуль _parse_response здійснює розбір цього JSON і зберігає для кожного тесту результат: чи потрібно оновлювати тест і чому;

— оновлення тестів: на основі відповідей моделі, система автоматично генерує оновлені версії тестів, доповнюючи або змінюючи їх відповідно до нових вимог функцій та їхніх залежностей. Тести, що потребують змін, надаються разом із поясненнями, які допомагають зрозуміти, чому ці зміни були необхідні.

Модуль класифікації оновлень тестів базується на кількох ключових критеріях, що допомагають визначити, чи тест потребує оновлення:

— зміни в параметрах: якщо змінилися вхідні або вихідні параметри функції (наприклад, додавання нових параметрів або зміна типів даних);

— зміни в бізнес-логіці: якщо змінився основний алгоритм або логіка роботи функції, що безпосередньо впливає на те, як повинні перевірятися результати;

— зміни у побічних ефектах: якщо функція тепер має нові побічні ефекти (наприклад, пише в базу даних або змінює глобальні змінні);

— перейменування функцій або класів: якщо тест більше не актуальний через зміну імені функції чи класу;

— неістотні зміни: зміни, що стосуються лише внутрішньої реалізації, без впливу на вхідні/вихідні дані або побічні ефекти (наприклад, рефакторинг коду).

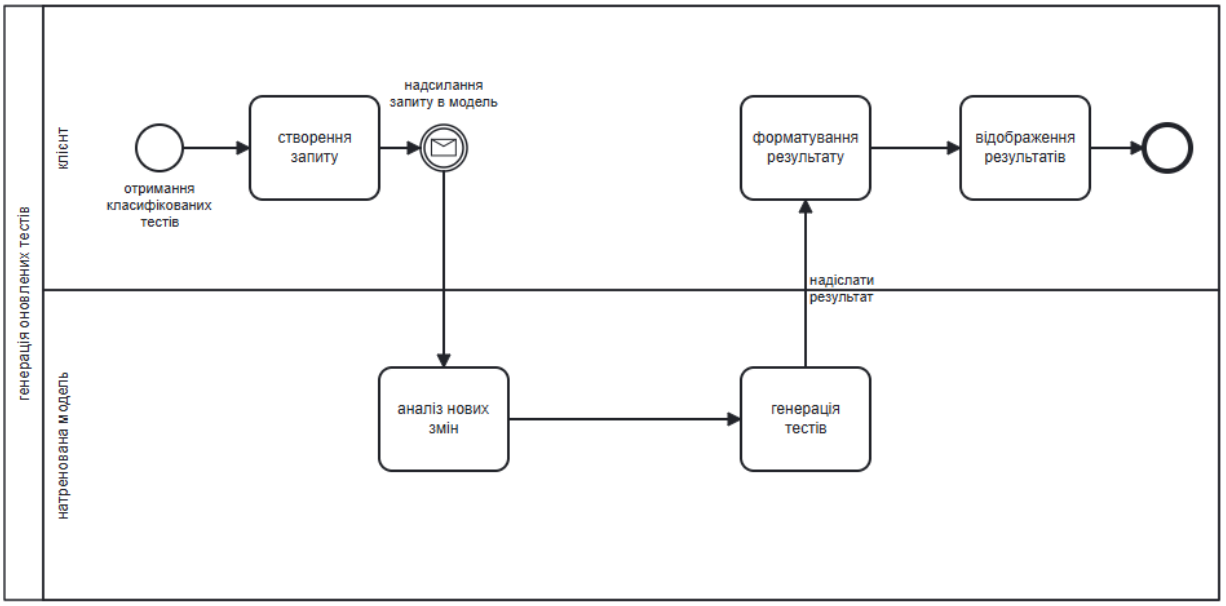

Рисунок 3.3 – Схема структурна діяльності процесу генерації тестів

3.2.4 Модуль центральної логіки обробки даних

Модуль центральної логіки обробки даних є серцевиною системи автоматизації оновлення тестових сценаріїв, оскільки він забезпечує взаємодію

між різними модулями системи і координує всі основні процеси. Його основною функцією є управління даними, які надходять від різних моделей, та контроль за виконанням процесів тестування і оновлення тестів. Основний компонент цього модуля реалізовано через MainController, який інтерфейсує з користувацьким інтерфейсом, організовує процеси аналізу коду, класифікації застарілих тестів та їх оновлення.

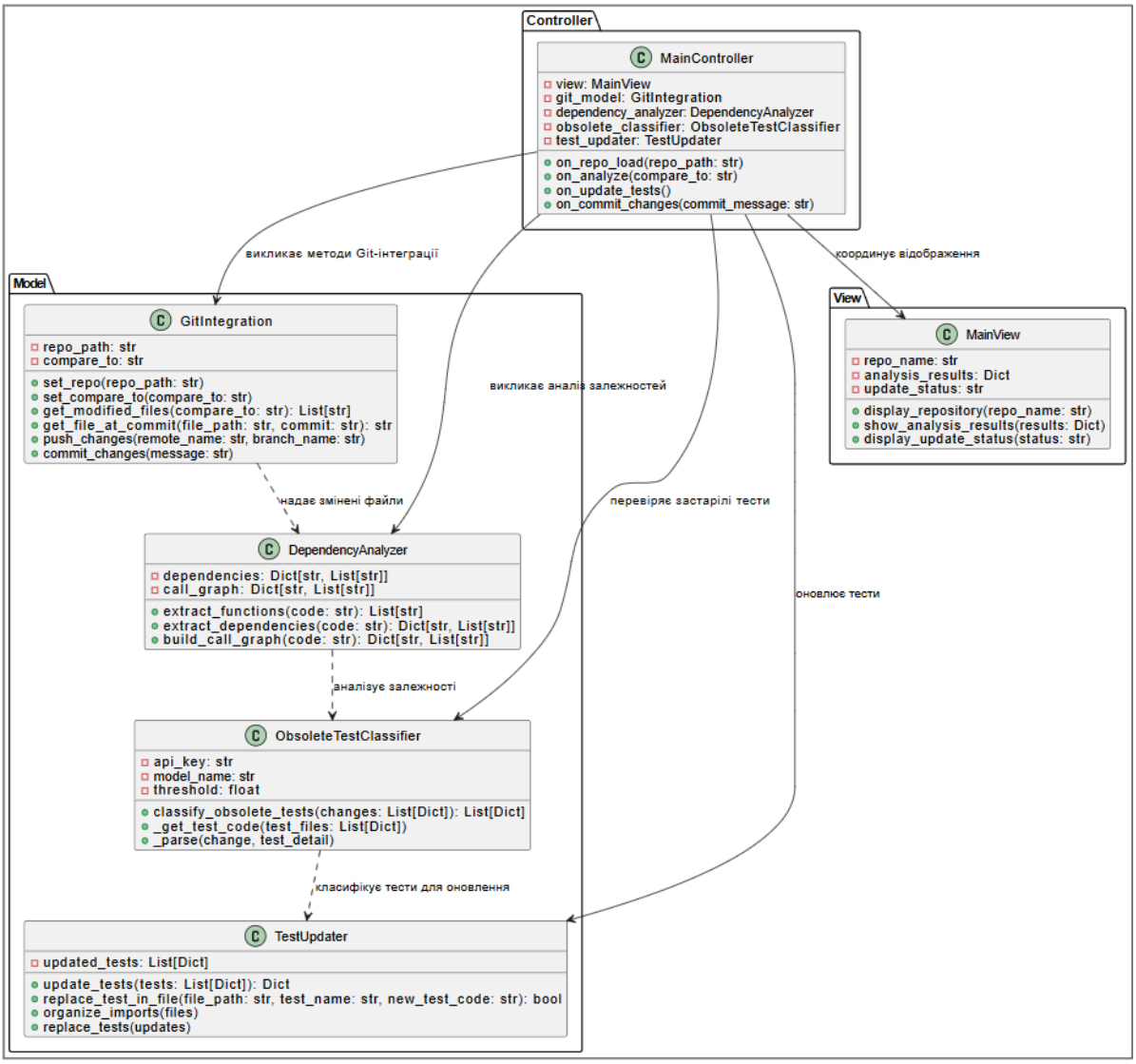

Рисунок 3.4 – Діаграма класів взаємодії компоненті системи

Алгоритм роботи:

а) ініціалізація і підключення до моделей:

1) клас MainController ініціалізує всі необхідні моделі: GitModel для роботи з репозиторіями, AnalysisModel для аналізу змін у коді, ClassificationModel для класифікації застарілих тестів, та UpdateModel для оновлення тестів;

2) кожна модель взаємодіє з іншими компонентами, надаючи функціональність для отримання і обробки даних, що дозволяє координувати дії системи;

б) Взаємодія з інтерфейсом користувача:

1) MainController також відповідає за взаємодію з користувацьким інтерфейсом через MainView. Кнопки інтерфейсу (наприклад, для завантаження репозиторіїв, запуску аналізу або оновлення тестів) підключені до відповідних методів контролера через сигнал-слот механізм PyQt5;

2) окремо обробляються взаємодії, пов’язані з комітами та пушами змін у репозиторій;

в) завантаження репозиторіїв і налаштування інтерфейсу:

1) за допомогою методу sow_repo_loader, користувач може вказати шлях до репозиторію, після чого система виконує базове налаштування через метод on_repo_loaded, що включає завантаження репозиторію в модель GitModel і оновлення відображення інформації в інтерфейсі;

2) налаштовуються також кнопки для комітів та пушів змін до репозиторію;

г) аналіз змін і класифікація застарілих тестів:

1) коли користувач ініціює процес аналізу через кнопку analyze_button, викликається метод analyze_repository, який обробляє останні зміни у коді, знаходить змінені файли та визначає тести, що можуть бути застарілими. Використовується метод load_commit_changes для завантаження змін, а потім застосовуються методи classify_and_display_obsolete_tests для визначення застарілих тестів;

д) оновлення тестів:

1) якщо система виявляє застарілі тести, викликається метод update_tests, що оновлює ці тести в коді. Оновлені тести після цього відображаються у відповідному інтерфейсі за допомогою методу display_updated_tests;

е) коміт і пуш змін:

1) MainController також відповідає за коміт і пуш змін у репозиторій через методи commit_changes, push_changes та commit_and_push. Ці методи дозволяють користувачу здійснювати зміни в коді, що безпосередньо стосуються оновлення тестових сценаріїв, а потім відправляти їх у віддалений репозиторій.

Ключові аспекти роботи контролера:

— обробка змін у коді: центральна логіка аналізує зміни, що були внесені до репозиторію, і використовує відповідні моделі для оцінки, які тести потребують оновлення;

— координація між компонентами: контролер об'єднує модулі аналізу, класифікації та оновлення тестів, що забезпечує плавну взаємодію між усіма етапами процесу;

— взаємодія з користувачем: всі дії у системі (анализ, оновлення тестів, коміти) пов'язані з користувацьким інтерфейсом, що дозволяє користувачам зручно управляти процесом оновлення тестових сценаріїв.

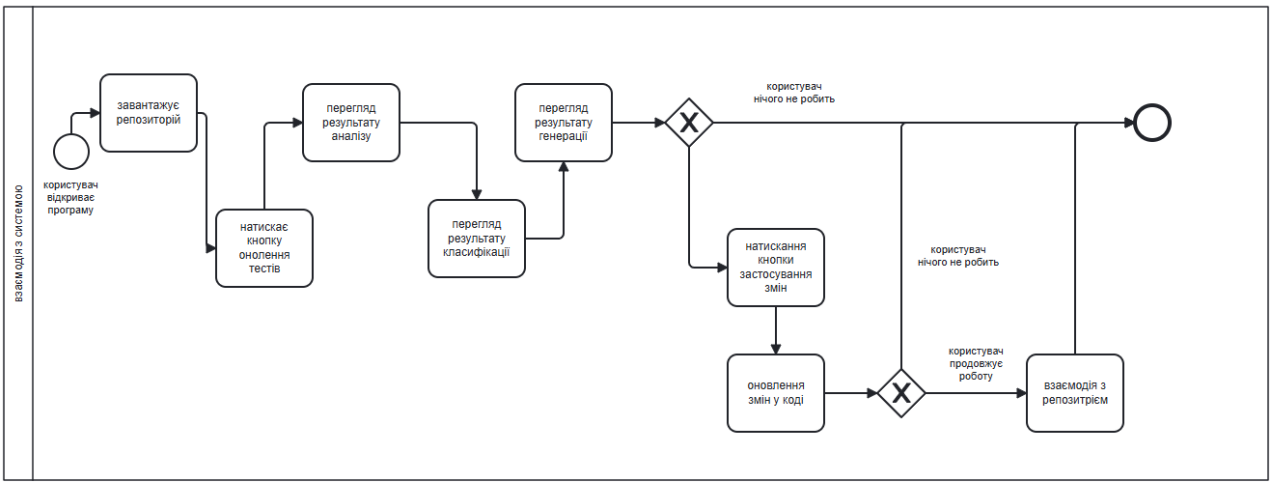

Рисунок 3.5 – Схема структурна діяльності процесу взаємодії з системою

3.3. Розробка та опис інтерфейсу користувача

Інтерфейс користувача є важливим елементом, що забезпечує зручність взаємодії з системою автоматизації оновлення тестів. Метою інтерфейсу є створення простого, інтуїтивно зрозумілого та ефективного способу доступу до всіх функцій системи, що дозволяє користувачам з мінімальними зусиллями управляти процесами тестування та оновлення тестів. В інтерфейсі реалізовані основні елементи, які забезпечують плавну взаємодію між користувачем і програмними модулями.

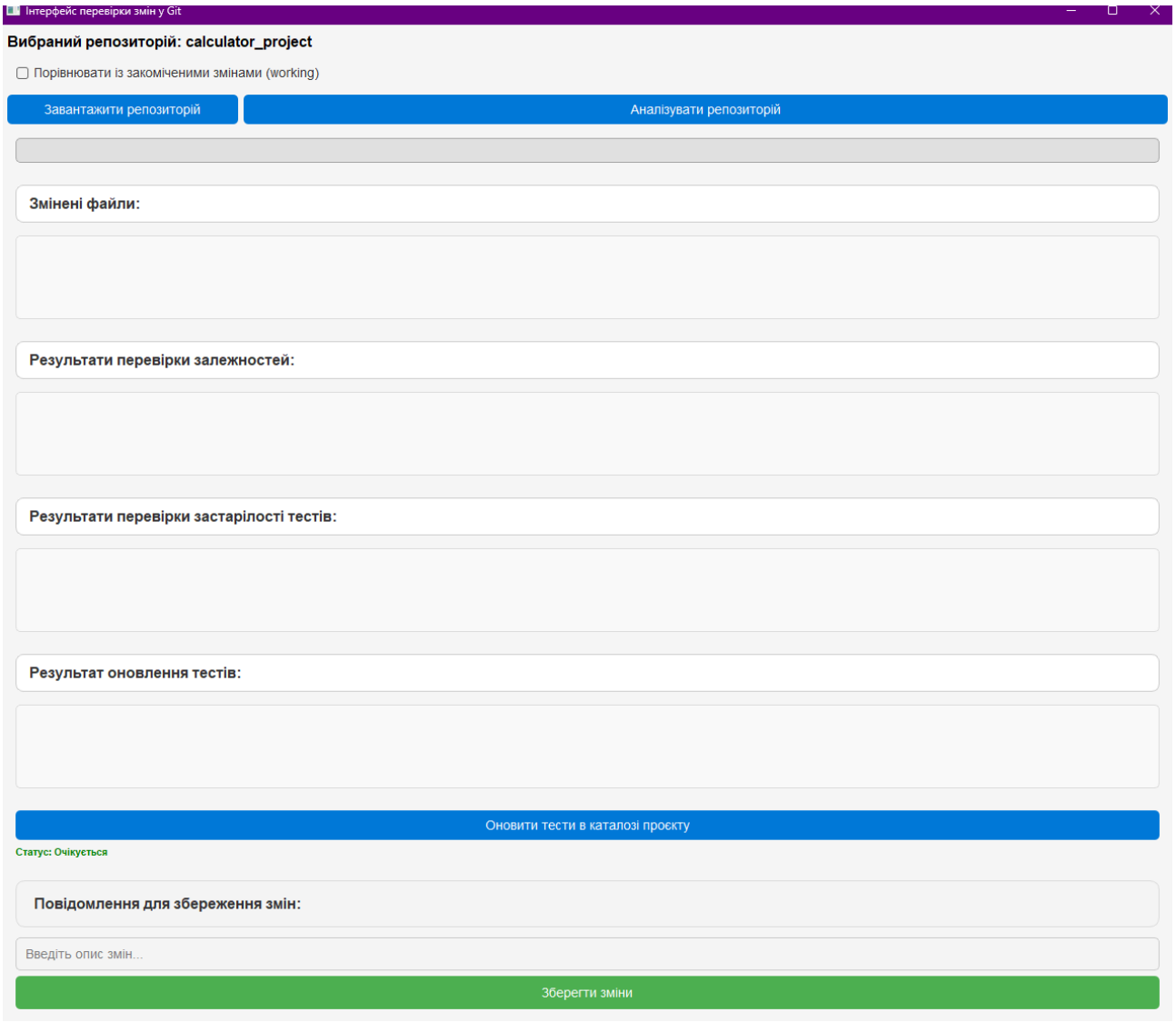

Рисунок 3.6 – Основне вікно керування

Основне вікно керування є центральним елементом інтерфейсу системи. Воно містить усі основні функціональні можливості, зокрема кнопки для завантаження репозиторію та аналізу змін у коді, а також кнопку для оновлення тестів, що ініціює автоматичне оновлення тестових сценаріїв відповідно до виявлених змін у коді. Інформаційна панель відображає статус виконання задач, таких як аналіз коду та оновлення тестів. Це вікно має зрозумілу навігацію та забезпечує доступ до всіх функцій системи без необхідності перемикатися між кількома вікнами. Користувач може миттєво бачити статус виконаних операцій і отримувати зворотній зв'язок про процеси оновлення.

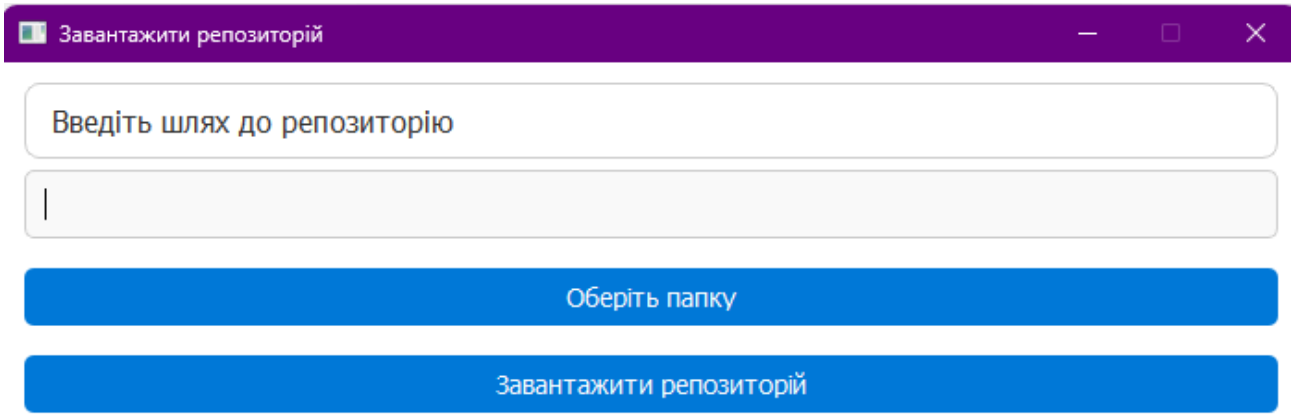

Рисунок 3.7 – Вікно завантаження репозиторію

Вікно завантаження репозиторію дозволяє користувачеві завантажити репозиторій для подальшого аналізу. У цьому вікні користувач має можливість ввести шлях до репозиторію, який необхідно завантажити, та активувати процес завантаження через кнопку. Після успішного завантаження репозиторію відображаються деталі репозиторію, включаючи його назву та основні метадані. Вікно також надає можливість налаштувати доступ до віддалених репозиторіїв, що забезпечує зручну взаємодію з системою контролю версій.

3.4. Вимоги до технічного забезпечення

— Процесор: 2 ГГц, 2-ядерний ЦП або кращий з можливістю розподілу ядер на потоки для ефективної обробки даних;

— ОП: не менше ніж 3072 МБ для забезпечення належної продуктивності при обробці великих обсягів даних;

— місце на диску: не менше ніж 1 ГБ для зберігання програмного забезпечення та тимчасових файлів;

— доступ до мережі Інтернет: для інтеграції з онлайн-сервісами та віддаленими репозиторіями.

Висновки

У третьому розділі було детально описано розробку та реалізацію основних компонентів системи автоматизації оновлення тестів. Розгляд

технологій та архітектури системи дозволив чітко визначити вибір інструментів, що забезпечують ефективне виконання завдань, таких як аналіз змін у коді, класифікація тестів, а також їхнє оновлення.

Розробка центральної логіки обробки даних, представлена, забезпечила взаємодію між усіма компонентами системи, включаючи модулі аналізу, класифікації та оновлення тестів. Взаємодія цих компонентів була описана через алгоритми, що визначають, як система обробляє зміни у коді, взаємодіє з репозиторієм, і автоматично оновлює тести відповідно до нових вимог.

Інтерфейс користувача був розроблений таким чином, щоб забезпечити максимальну зручність та інтуїтивно зрозумілу взаємодію з користувачем. Основне вікно керування, вікно завантаження репозиторію та верхня панель навігації створюють ефективний та простий у використанні інтерфейс для виконання основних операцій. Це дозволяє користувачеві з мінімальними зусиллями виконувати завдання, пов’язані з аналізом коду та оновленням тестів.

Загалом, третій розділ демонструє інтеграцію технологій, алгоритмів і зручного інтерфейсу, що забезпечують ефективну та автоматизовану систему для оновлення тестових сценаріїв у процесі розробки програмного забезпечення.

4 ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ЗАПРОПОНОВАНОГО РІШЕННЯ

4.1 Підготовка до тестування

4.1.1 Налаштування середовища для тестування

Для тестування системи було підготовлено програмне середовище на основі мови Python із використанням бібліотек pytest[17] для модульного тестування, а також PyQt5 для роботи з інтерфейсом користувача.

Для роботи із репозиторіями було реалізовано підтримку систем контролю версій, що дозволяє аналізувати змінені файли та пов'язані тести. Це дало змогу створити реалістичні сценарії для оцінки функціональності системи.

4.1.2 Підготовка тестових сценаріїв

Для створення тестових сценаріїв було імітовано різні типи змін у програмному коді:

- додавання нових функцій;
- зміна бізнес-логіки існуючих функцій;
- перейменування або видалення компонентів.

Кожен сценарій включав відповідні тести, що дозволяли перевірити:

- коректність аналізу змін у коді;
- адекватність класифікації тестів;
- точність і валідність генерації оновлених тестів.

4.1.3 Визначення критеріїв успішного тестування

Основними критеріями для оцінки ефективності системи стали:

— точність класифікації тестів: правильність визначення застарілих тестів.

— швидкість виконання операцій: аналіз змін, класифікація тестів і генерація оновлень;

— стабільність системи: здатність коректно обробляти сценарії з великим обсягом змін;

— зручність використання інтерфейсу: доступність усіх функцій для користувача.

4.2. Процес тестування

Тестування системи автоматизації оновлення тестових сценаріїв було організоване у кілька етапів, кожен із яких спрямований на перевірку функціональності окремих модулів та їх взаємодії. Під час тестування здійснювалася перевірка коректності роботи таких модулів:

— аналіз змін у коді;

— класифікація тестів;

— генерація оновлених тестів.

Для кожного модуля створювалися тестові сценарії, які перевіряли різні аспекти їх функціонування. Наприклад, у модулі аналізу змін у коді перевірялося, чи правильно система ідентифікує модифіковані функції. Усі тести виконувалися за допомогою бібліотеки pytest, що дозволило автоматизувати їх запуск та збір результатів.

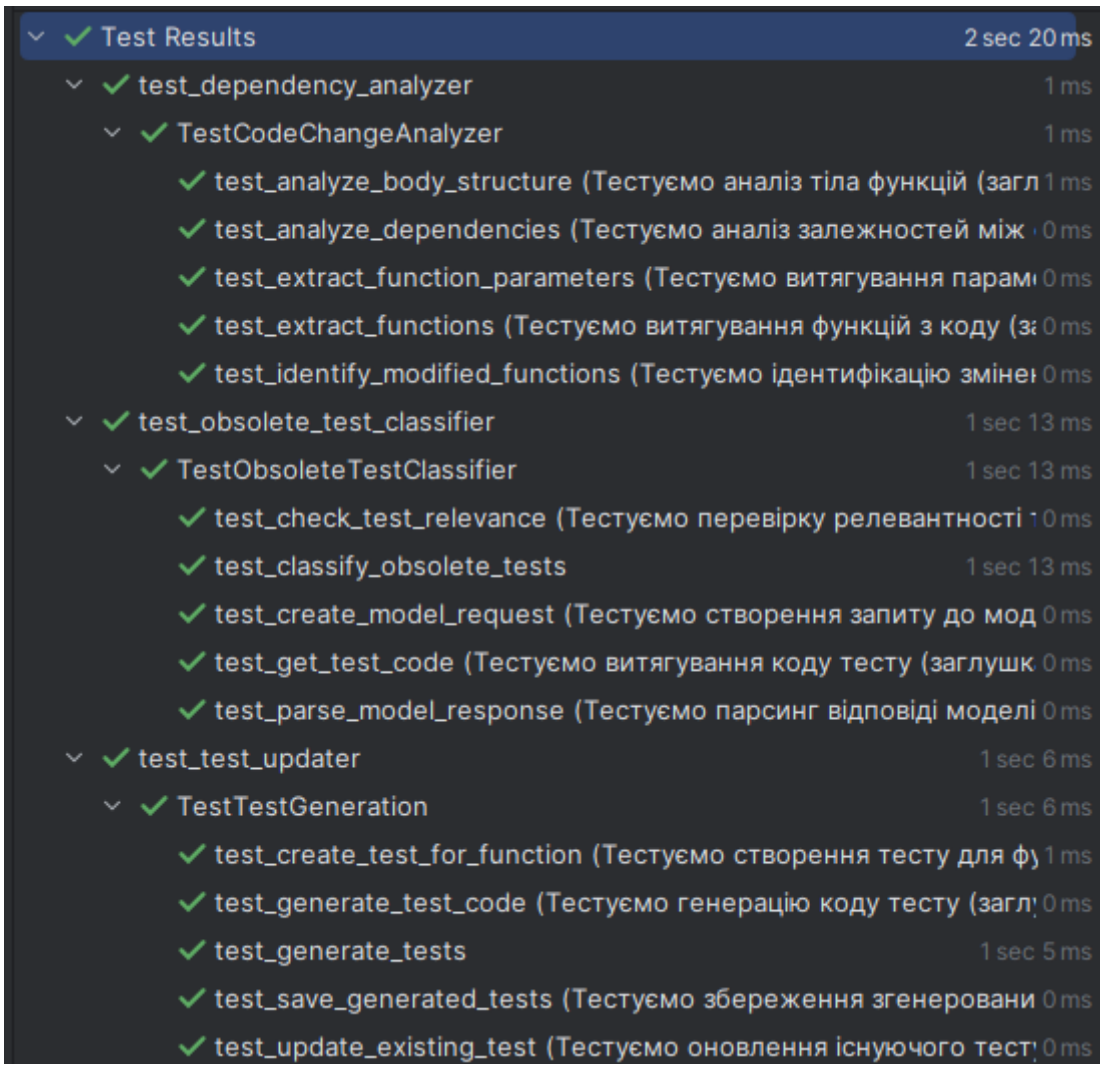

Рисунок 4.1 – Запуск тестів у середовищі PyCharm

Інтеграційне тестування[18] було спрямоване на перевірку коректності передачі даних між модулями. Для цього створювалися спеціальні симуляції, у яких передбачалися можливі сценарії роботи системи.

У ході тестування були виконані наступні сценарії:

а) завантаження репозиторію через інтерфейс користувача:

1) запустити модуль аналізу змін у коді з різними вхідними файлами;

2) переконатися, що модуль успішно витягує всі функції з коду і коректно визначає їх параметри та структуру;

б) аналіз змін у коді:

1) виконати аналіз змін у програмному коді для визначення необхідності оновлення тестових сценаріїв;

2) переконатися, що система коректно виявляє зміни та передає їх до відповідних модулів для подальшої обробки;

в) генерація тестів на основі змін:

1) запустити модуль генерації тестів після внесення змін у код;

2) переконатися, що створені тести відповідають змінам у функціях і охоплюють всі необхідні аспекти функціональності;

г) інтеграційне тестування:

1) перевірити взаємодію між модулями аналізу змін у коді, класифікації тестів та генерації тестів;

2) переконатися, що дані коректно передаються між модулями і що всі етапи виконуються без помилок;

д) тестування продуктивності при великій кількості функцій:

1) для оцінки швидкодії системи було створено великий обсяг функцій у коді, щоб перевірити, чи зберігається швидкість обробки та аналізу змін під високим навантаженням;

2) під час експерименту система продемонструвала стабільну роботу, забезпечуючи швидкий аналіз та оновлення тестів навіть при великій кількості функцій, що підтверджує ефективність використаних алгоритмів;

е) перевірка стабільності після багаторазових операцій:

1) для тестування надійності було проведено багаторазові цикли операцій аналізу та оновлення тестів;

2) очікувалося, що система залишатиметься стабільною після численних ітерацій роботи, без виникнення помилок або збоїв;

3) результати показали, що система зберігає стабільність і функціональність навіть після багатьох запусків, що свідчить про високу надійність розробленого рішення.

Система тестується на стабільність при виникненні помилок або винятків, щоб переконатися, що всі компоненти системи коректно обробляють такі ситуації. Це важливо для підвищення надійності та точності автоматичного оновлення тестових сценаріїв.

Проведені експерименти дозволили переконатися у функціональності, продуктивності та стабільності роботи системи автоматизації оновлення тестових сценаріїв, підтвердивши її готовність до інтеграції та подальшого використання. Виявлені проблеми були оперативно виправлені, що дозволило створити надійне програмне рішення для ефективної взаємодії з системою аналізу змін у коді та автоматичного оновлення тестів.

4.4. Оцінка ефективності

Оцінка ефективності запропонованої системи автоматизації оновлення тестових сценаріїв проводилася на основі кількох ключових показників, зокрема точності оновлення тестів, часу, необхідного для оновлення тестів до і після автоматизації, а також загальної продуктивності системи при обробці проектів різного масштабу. Для тестування використовувалися три репозиторії з різною кількістю тестів: малий (~250 тестів), середній (~750 тестів) та великий (~1350 тестів).

Точність оновлення тестів була визначена як відсоток правильно оновлених тестів від загальної кількості знайдених тестів, що залежать від змін у коді. Результати тестування показали різну точність для проектів різного масштабу, що вказує на складність задачі при збільшенні кількості тестів

Таблиця 4.1 – Точність оновлення тестів за розмірами проектів

Розмір проекту (к-ть тестів)	Кількість релевантно оновлених тестів	Знайдено тестів	Точність оновлення (%)
Малий (~250)	3	3	100

Продовження таблиці 4.1

Розмір проекту (к-ть тестів)	Кількість релевантно оновлених тестів	Знайдено тестів	Точність оновлення (%)
Середній (~750)	11	12	92
Великий (~1350)	21	25	84

Як видно з наведених результатів, точність оновлення тестів поступово зменшується з ростом кількості тестів у проекті. Для малих проектів точність оновлення становить 100%, в той час як для великих проектів цей показник знижується до 84%. Це свідчить про те, що з підвищенням розміру проекту і кількості тестів складність коректного оновлення зростає.

Одним із важливих аспектів ефективності є зниження часу, необхідного для оновлення тестів. Оцінка часу, витраченого на виконання операцій до та після впровадження автоматизації, дозволяє оцінити вплив автоматизації на загальну продуктивність процесу.

Таблиця 4.2 – Порівняння часу виконання операцій до та після автоматизації

Розмір проекту (к-ть тестів)	Час до автоматизації (хв)	Час після автоматизації (хв)	Точність оновлення (%)
Малий (~250)	5	1	80
Середній (~750)	35	6	82
Великий (~1350)	96	18	81

Результати демонструють значне зниження часу, необхідного для оновлення тестів після впровадження автоматизації. Для малих проектів час зменшується на 80%, для середніх – на 83%, а для великих – на 81%. Водночас, точність оновлення тестів не має суттєвих змін, що свідчить про стабільність системи навіть при значних скороченнях часу.

Висновки

У рамках розділу була проведена оцінка ефективності запропонованої системи автоматизації оновлення тестів, яка включала перевірку точності оновлення тестів, часу виконання операцій та загальної продуктивності системи для проектів різного розміру.

Результати тестування підтвердили, що система автоматизації значно скорочує час, необхідний для оновлення тестів, порівняно з традиційним ручним оновленням. Це особливо актуально для великих проектів, де автоматизація дозволяє знизити час виконання операцій на 80-83%.

Точність оновлення тестів також була на високому рівні, зокрема для малих і середніх проектів, де точність залишалася в межах 92-100%. Проте, для великих проектів спостерігався незначний спад точності (до 84%), що вказує на необхідність подальшого вдосконалення механізмів класифікації та генерації тестів для великих кодових баз.

Загалом, запропонована система продемонструвала високу ефективність в автоматизації процесу оновлення тестів, зокрема в зменшенні часу виконання та підтримці високої точності, що робить її практично корисною для широкого кола проектів, зокрема для середніх та малих за розміром. Водночас для великих проектів потребуються додаткові удосконалення алгоритмів, щоб досягти стабільно високої точності оновлення тестів.

5 МАРКЕТИНГОВИЙ АНАЛІЗ СТАРТАП-ПРОЄКТУ

5.1 Опис ідеї проєкту

Таблиця 5.1 — Опис ідеї стартап-проєкту

Зміст ідеї	Напрямки застосування	Вигоди для користучача
Розробка системи для автоматичного оновлення тестових сценаріїв на основі аналізу змін у коді та класифікації застарілих тестів за допомогою мовної моделі.	Автоматизація процесу тестування ПЗ	Зниження витрат часу на оновлення тестів, покращення точності тестування та зниження людських помилок.
	Підвищення ефективності розробки ПЗ	

Таблиця 5.2 — Опис ідеї стартап-проєкту

№	Техніко-економічні характеристики ідеї	Продукція конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проєкт	Selenium	Cypress	Appium			
1	Кросплатформність	Так	Так	Так	Так			+
2	Можливість автоматизації UI	Так	Ні	Так	Ні			+
3	Масштабованість	Так	Ні	Так	Ні	+		

Продовження таблиці 5.2

№	Техніко-економічні характеристики ідеї	Продукція конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проєкт	Selenium	Cypress	Appium			
4	Інтеграція з іншими інструментами	Так	Так	Ні	Ні		+	
5	Підтримка сучасних фреймворків	Так	Ні	Ні	Так			+
6	Швидкість виконання тестів	Так	Ні	Так	Так			+

5.2. Технологічний аудит ідеї проєкту

Таблиця 5.3 — Технологічна здійсненність ідеї проєкту

№	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Аналіз змін у коді та автоматизація оновлення тестів	Python	Наявна	Доступна
		C#	Наявна	Доступна
		Java	Наявна	Доступна
Обрана технологія реалізації ідеї проєкту: Python				
2	Інтерфейс користувача	Tkinter	Наявна	Доступна
		Kivy	Наявна	Доступна
		PyQt5	Наявна	Доступна
Обрана технологія реалізації ідеї проєкту: PyQt5				

Продовження таблиці 5.3

№	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
3	Моделі для генерації	OpenAI	Наявна	Доступна
		BERT	Наявна	Доступна
		Використання власної моделі на основі машиного навчання	Відсутня	Доступна
Обрана технологія реалізації ідеї проєкту: OpenAI				

5.3 Аналіз ринкових можливостей запуску стартап-проєкту

Таблиця 5.4 — Попередня характеристика потенційного ринку

№	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	5
2	Загальний обсяг продаж, грн./ум.од	1000000 ₴
3	Динаміка ринку	Зростаюча
4	Наявність обмежень для входу	Високі вимоги до технічної підготовки розробників та знань у сфері тестування, наявність конкурентів на ринку.
5	Специфічні вимоги до стандартизації та сертифікації	Необхідність дотримання стандартів ISO та сертифікації якості для певних ринків
6	Середня норма рентабельності в галузі або по ринку, %	20-25%.

Висновок: враховуючи кількість головних гравців по ринку, зростаючу динаміку ринку, невелику кількість конкурентів та середню норму рентабельності можна зробити висновок, що на даний момент, ринок для входження стартап-продукту є привабливим.

Таблиця 5.5 — Характеристика потенційних клієнтів стартап-проєкту

№	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці цільових груп клієнтів	Вимоги споживачів до товару
1	Потреба в автоматизації оновлення тестових сценаріїв	Великі компанії розробники ПЗ, стартапи в сфері технологій	Компанії, що мають масштабовані проекти, потребують ефективної автоматизації тестів, швидкого оновлення тестових сценаріїв	Споживачі очікують зручний веб-інтерфейс для налаштування та виконання автоматизованого оновлення тестів, з можливістю інтеграції через API для більш гнучкого використання та експорту даних.
2	Потреба в покращенні якості програмного забезпечення через актуальність тестів	Менші компанії та фрілансери, які займаються розробкою ПЗ	Для невеликих проектів важлива простота та низька вартість інструментів для автоматизації тестів.	
3	Потреба у зменшенні витрат часу на тестування через автоматизацію	ІТ-компанії, які мають численні розробки з високими темпами змін у коді	Для таких компаній важлива висока ефективність, автоматизація та інтеграція з іншими інструментами.	

Таблиця 5.6 — Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Конкуренція на ринку автоматизації тестування	Збільшення кількості конкурентів, які пропонують подібні рішення для автоматизації тестів	Покращення унікальних функцій продукту, акцент на інноваціях та ефективності
2	Технічні проблеми або збої у системі	Виникнення технічних помилок або збоїв, які можуть призвести до відмови в роботі продукту	Оперативне виправлення помилок, збереження високого рівня підтримки клієнтів
3	Зміни в вимогах до тестування ПЗ	Виникнення нових стандартів чи вимог щодо тестування програмного забезпечення в індустрії	Адаптація продукту до нових вимог, оновлення відповідно до змін у нормативних актах

Таблиця 5.7 — Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Новий продукт	Розробка нових функцій або версій продукту для задоволення нових потреб ринку	Оновлення або доповнення функціоналу, активна маркетингова кампанія
2	Вихід аналогу	Поява подібного продукту на ринку, що може посилити попит на автоматизацію тестування	Адаптація продукту до нових стандартів, вдосконалення функцій і зниження ціни

Продовження таблиці 5.7

№	Фактор	Зміст можливості	Можлива реакція компанії
3	Зворотний зв’язок від користувачів	Відгуки користувачів можуть допомогти виявити недоліки і пропозиції щодо поліпшення	Аналіз зворотного зв’язку, покращення продукту та підтримка користувачів
4	Грошова винагорода за рекламу	Можливість отримати фінансування або бонуси через партнерства з рекламними платформами	Впровадження рекламних кампаній, розширення партнерських відносин

Таблиця 5.8 — Ступеневий аналіз конкуренції на ринку

№	Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1	Тип конкуренції: монополістична	На ринку тестових інструментів домінують кілька великих гравців (наприклад, Selenium, TestComplete).	Компанія повинна пропонувати унікальні функції, такі як інтеграція мовних моделей для автоматичного оновлення тестів.
2	Рівень конкурентної боротьби: світовий	Світові компанії активно використовують передові технології, такі як штучний інтелект і машинне навчання.	Потрібно активно впроваджувати новітні технології, такі як використання мовних моделей для тестування.

Продовження таблиці 5.8

№	Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
3	Галузева ознака: внутрішньогалузева	Конкуренція серед інструментів для автоматизації тестування на основі змін у коді.	Важливо зосередитись на специфічних потребах ринку автоматизації тестів і надавати нові функції, які є відсутніми у конкурентів.
4	Конкуренція за видами товарів: товарно-видова	Пропозиція інструментів для автоматизації тестування, таких як різні фреймворки для тестування.	Потрібно удосконалювати продукт, щоб він міг інтегруватися з різними фреймворками і мовами програмування.
5	Характер конкурентних переваг: цінова та не цінова	Конкуренція як за допомогою ціни, так і через додаткові функціональні можливості (наприклад, інтеграція з AI).	Інвестувати в розробку інтелектуальних алгоритмів для полегшення оновлення тестів і зниження витрат на обслуговування.
6	За інтенсивністю: марочна	Створення брендів, що надають інноваційні рішення в автоматизації тестування, таких як інструменти з підтримкою AI.	Інвестувати у побудову сильного бренду та маркетинг з фокусом на інтелектуальну автоматизацію тестів і зручний інтерфейс.

Таблиця 5.9 — Аналіз конкуренції в галузі за М. Портером

Складовіаналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Компанії, що надають рішення для автоматизації тестування з використанням інтелектуальних систем, таких як Testim, Functionize.	Нові стартапи, що можуть змінювати підходи до автоматизації тестування.	Постачальники технологій для автоматизації тестування, зокрема мовні моделі (наприклад, OpenAI).	Компанії, що потребують швидкої адаптації тестів до змін у коді, зокрема стартапи та середні компанії в галузі ПЗ.	Інші інструменти автоматизації тестування, такі як Selenium, Cypress.
Висновки	Висновки	Прогнозуються важливі гравці, але є можливість створити перевагу через новітні технології, зокрема використання мовних моделей для класифікації тестів.	Потенційні конкуренти можуть з'явитись завдяки новим технологіям, зокрема в машинному навчанні.	Необхідні надійні постачальники для високої ефективності оновлення тестів.	Цільова аудиторія — компанії, що потребують автоматизованих рішень для постійного оновлення тестів.

Проаналізувавши можливості роботи на ринку з огляду на конкурентну ситуацію можна зробити висновок: оскільки кожний з існуючих продуктів не впливає у великій мірі на поточну ситуацію на ринку в цілому, кожний з

існуючих продуктів має свою специфічну сферу використання та свої позитивні та негативні сторони щодо рішення певних типів задач, то робота та вихід на даний ринок є можливою і реалізованою задачею.

Для виходу на ринок продукт повинен мати функціонал що відсутній у продуктів-аналогів, повинен задовольняти потреби користувачів, мати необхідний та достатній функціонал з конфігурування, підтримку зі сторони розробників та можливість розробки спеціального функціоналу за відповідною ліцензією.

Таблиця 5.10 — Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування
1	Використання мовних моделей	Інноваційний підхід до автоматизації оновлення тестів за допомогою аналізу змін у коді та класифікації тестів.
2	Висока швидкість обробки	Система забезпечує автоматичний аналіз і оновлення тестів з мінімальними затримками.
3	Зручний інтерфейс користувача	Інтуїтивно зрозумілий UI дозволяє користувачам легко взаємодіяти з системою без спеціальних знань.
4	Модульність системи	Компонентна архітектура дозволяє легко інтегрувати та адаптувати систему до інших рішень.
5	Зниження витрат на підтримку тестів	Автоматизація дозволяє скоротити витрати на ручну перевірку та оновлення тестових сценаріїв.

Таблиця 5.11 — Порівняльний аналіз сильних та слабких сторін системи кешування мало змінних даних

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з запропонованим						
			-3	-2	-1	0	+1	+2	+3
1	Актуальність товару	20	+						
2	Відкритий вихідний код	16				+			
3	Висока готовність до кооперації	15			+				
4	Простота використання	15			+				
5	Можливість інтеграції з іншими програмними засобами	16				+			
6	Кросплатформеність	11						+	

Таблиця 5.12 — SWOT аналіз стартап-проєкту

Сильні сторони (S): – Використання сучасних мовних моделей для автоматизації оновлення тестів. – Інтуїтивно зрозумілий інтерфейс для користувачів.	Слабкі сторони (W): – Висока залежність від зовнішніх API мовної моделі (наприклад, ChatGPT). – Потреба у значних обчислювальних ресурсах для масштабування.
Можливості (O): – Розширення функціоналу для інших мов програмування. – Інтеграція з CI/CD системами для забезпечення безперервного тестування.	Загрози (T): – Конкуренція зі схожими продуктами, такими як Selenium чи Cypress. – - Ризики, пов’язані зі змінами в ліцензіях на використання мовної моделі.

Таблиця 5.13 — Альтернативи ринкового впровадження стартап-проєкту

№	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Безкоштовне надання певного функціоналу у користування споживачам на обмежений термін	Головний ресурс – люди, даний ресурс - наявний	2-3 місяці
2	Реклама	Залучення власних коштів для реклами товару	1-2 місяці
3	Написання статей та опис товару на відомих ресурсах	Головний ресурс – час, даний ресурс - наявний	3-4 тижні
4	Презентація товару на хакатонах й інших ІТ заходах	Ресурс – час та гроші для участі, наявні	1-3 місяці

5.4 Розроблення ринкової стратегії проєкту

Таблиця 5.14 — Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Розробники програмного забезпечення в малих і середніх компаніях, які регулярно оновлюють функціонал своїх продуктів	Висока, оскільки потребують автоматизації процесів оновлення тестів для економії часу	Високий, через поширення практики тестування в більшості компаній	Помірна, на ринку є декілька інших рішень для автоматизації тестів	Середня, потребує налаштування інтерфейсу та інтеграції з існуючими інструментами

Продовження таблиці 5.14

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
2	Великі компанії та IT-консалтингові фірми, що працюють з численними проектами	Висока, зважаючи на складність тестування в проектах великого масштабу	Середній, оскільки більшість великих компаній вже використовують вбудовані інструменти	Висока, через домінування великих компаній і наявність комплексних рішень	Низька, високі вимоги до сумісності з існуючими процесами
3	Стартапи, які створюють нові програмні продукти і шукають оптимальні рішення для тестування	Помірна, оскільки стартапи можуть бути обережні щодо нових інструментів	Середній, залежно від специфіки стартапів і їх фінансування	Низька, оскільки стартапи часто готові до нових рішень	Середня, потребує простоти налаштування та швидкої інтеграції
4	Академічні установи та навчальні платформи, що вивчають методи автоматизації тестування	Помірна, оскільки інтерес до нових технологій в навчальному середовищі зростає	Низький, оскільки попит обмежений навчальними цілями та невеликими проектами	Низька, конкуренція з іншими навчальними платформами	Висока, оскільки для академічних цілей досить низьких вимог до функціональності
Які цільові групи обрано: Розробники програмного забезпечення в малих і середніх компаніях, великі компанії та IT-консалтингові фірми, стартапи, академічні установи та навчальні платформи					

Відповідно до проведеного аналізу можна зробити висновок, що підходящою цільовою групою для розповсюдження даного програмного продукту є працівники IT сфери, IT компанії в цілому та будь-які підприємства

котрі використовують програмні продукти побудовані на мові програмування Python, та використовують реляційні бази даних. Відповідно до стратегії охоплення ринку збуту товару обрано стратегію масового маркетингу, оскільки для підприємств, IT працівників та IT компаній у цілому надається стандартизований продукт з можливістю розширення функціональності за домовленістю (відповідно до ліцензії).

Таблиця 5.15 — Визначення базової стратегії розвитку

Обрана альтернатива розвитку проєкту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Надання функціональності що відсутня у товарів-замінників, підтримка клієнтів	Проведення реклами, освітлення унікальної функціональності через інтернет ресурси та інші канали, контакт напряму з споживачами; формування лояльності і прихильності споживачів	Зниження ступеню замінності товару; Прихильність клієнтів; Відмітні властивості товару; Відмітні характеристики товару;	Стратегія диференціації

Таблиця 5.16 — Визначення базової стратегії конкурентної поведінки

Чи є проєкт «першопрохідцем» на ринку	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, які?	Стратегія диференціації
Ні, оскільки є товари-заміннники, але дані товари замінники не мають деякого необхідного функціоналу	Так, ціль компанії знайти нових споживачів та, частково, забрати існуючих у конкурентів задля задоволення потреб останніх	Компанія частково копіює характеристики товару конкурента, основна ціль компанії розробка нового унікального функціоналу, з підтримкою основного функціоналу конкурентів	Стратегія заняття конкурентної ніші

Таблиця 5.17 — Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможн і позиції власного стартап-проєкту	Вибір асоціацій, які мають сформувати комплексну позицію власного проєкту
1	Можливість заміни різних платформ, що використовує система	Стратегії диверсифіковано го зростання	Система буде розроблена з позиції принципів проектування, що дозволять клієнтам застосовувати різні платформи для інтеграції вже з існуючими продуктами	Кросплатформеніс ть, економічна вигода, залучення більшої кількості клієнтів

Продовження таблиці 5.17

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможн і позиції власного стартап-проєкту	Вибір асоціацій, які мають сформувати комплексну позицію власного проєкту
2	Багатозадачність	Стратегії концентрованого зростання	З самого початку система буде спрямована на виконання широкого спектру функцій, який може бути легко доповнений новими	Виконання різних функцій, ефективність, актуальність
3	Здатність системи до розширювання	Стратегії диверсифіковано го зростання	Система буде побудована модульним способом, що дозволить легко додавати нові модулі без зміни існуючих	Модульність, відкритість до змін, гнучкість

Відповідно до проведеного аналізу можна зробити висновок, що стартап-компанія вибирає як базову стратегію розвитку – стратегію диференціації, як базову стратегію конкурентної поведінки – стратегію заняття конкурентної ніші.

5.5 Розроблення маркетингової програми стартап-проєкту

Таблиця 5.18 — Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Підвищення ефективності тестування	Швидке і автоматичне оновлення тестових сценаріїв при змінах у коді	Використання мовних моделей для класифікації тестів і автоматичне оновлення без ручного втручання

Продовження таблиці 5.18

2	Зменшення витрат часу на підтримку тестів	Автоматичне визначення застарілих тестів і оновлення їх відповідно до змін у коді	Інтеграція аналізу змін у коді з автоматичним оновленням тестів без потреби ручної перевірки

Таблиця 5.19 — Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
1. Товар за задумом	Програмне забезпечення для автоматизації оновлення тестових сценаріїв, яке аналізує зміни в коді та відповідно коригує тести. Можливість інтеграції з існуючими репозиторіями та використання моделей для класифікації застарілих тестів і їх генерації. Передбачено зручний інтерфейс для взаємодії користувача з системою та підтримка автоматичного оновлення тестів без необхідності ручного втручання.		
2. Товар у реальному виконанні	Автоматичне оновлення тестів	М	Тл, Тх
	Інтеграція з репозиторіями	Нм	Тл, Тх
	Підтримка мовної моделі для класифікації	Нм	Тх, Тл
	Зручний графічний інтерфейс	М	Е, Ор
	Генерація тестів на основі змін у коді	М	Тх, Тл
	Підтримка імпорту даних	Нм	Тл, Тх
	Висока швидкість обробки змін	М	Вр, Тх

Продовження таблиці 5.19

Рівні товару	Сутність та складові		
	Автоматичне оновлення тестів	М	Тл, Тх
	Якість: експериментально доведена відмовостійкість та ефективність		
	Пакування: Характеристики продукту будуть оформлено в офіційній документації на сайті сервісу, а також на відкритому репозиторії в GitHub		
3. Товар із підкріпленням	До продажу: наявна повна документація, акції на придбання декількох ліцензій, знижки для певних сегментів на покупку товару		
	Після продажу: додаткова підтримка спеціалістів налаштування, підтримка з боку розробника		
За рахунок чого потенційний товар буде захищено від копіювання: захист інтелектуальної власності, патент			

Таблиця 5.20 — Визначення меж встановлення ціни

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
Безкоштовні	0 – 150$	Середній для товару з відкритим вихідним кодом і високий для можливої комерційної версії продукту	30 – 45$ для комерційної версії

Таблиця 5.21 — Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Цільові клієнти можуть купити підписку на використання вебзастосунку	- аналіз ринку; - дослідження ринків для розширення аналіз законодавчих норм різних країн	Багатоканальний розподіл	Міжнародні торгівельні площадки

Таблиця 5.22 — Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Малі та середні підприємства, стартапи, які потребують автоматизації тестування, знижуючи витрати на оновлення тестів.	Технічні блоги, форуми (Stack Overflow), YouTube-канали, email-розсилки.	Інноваційність, автоматизація, зниження витрат на підтримку тестів.	Показати, як автоматизація оновлення тестових сценаріїв дозволяє зекономити час та ресурси для малих і середніх підприємств.	Інвестуйте у якість і економте на витратах: автоматизуйте оновлення тестів

Продовження таблиці 5.22

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
2	Розробники ПЗ та QA інженери, які займаються тестуванням і шукають інструменти для підвищення ефективності роботи.		Підвищення ефективності, точність, автоматизація процесів.	Донести до користувачів, що їхні тести можуть бути оновлені автоматично, що скоротить час на підтримку і полегшить роботу.	

Як результат було створено ринкову (маркетингову) програму, що включає в себе визначення ключових переваг концепції потенційного товару, опис моделі товару, визначення меж встановлення ціни, формування системи збуту та концепцію маркетингових комунікацій.

Висновки

У п’ятому розділі детально розглянуто основні аспекти створення стартап-проєкту, зокрема стратегії розробки, оцінку потенційного попиту, динаміку ринку та його рентабельність. Зроблено висновок про реальну можливість комерціалізації проєкту на ринку. Проведено дослідження цільових груп клієнтів, потенційних перешкод для входу на ринок, аналіз конкурентного середовища та рівня конкурентоспроможності проєкту, що підтвердило його перспективність.

У рамках розділу також було детально проаналізовано сильні та слабкі сторони продукту, можливості й загрози, що можуть вплинути на його впровадження та довгострокову підтримку. Визначено цільові групи споживачів, серед яких національні аутсорсингові компанії, закордонні виробники технологічних продуктів, університети та дослідницькі установи. Розроблено

стратегії розвитку, конкурентної поведінки та позиціонування продукту на ринку з урахуванням його унікальних переваг.

У підсумку проведеного аналізу обґрунтовано доцільність подальшої реалізації проєкту з огляду на його переваги, ринкові можливості та потенційні ризики. Визначені в розділі аспекти формують основу для розробки конкретних стратегій і планів дій, спрямованих на успішну комерціалізацію стартап-проєкту.

ВИСНОВКИ

У ході виконання магістерської дисертації було розглянуто питання автоматизації оновлення тестових сценаріїв програмного забезпечення, зокрема аналізу змін у коді, класифікації тестів та генерації оновлених тестів. Основною задачею було розробити систему, що автоматично адаптує тестові сценарії до змін у коді, знижуючи витрати часу та ресурсів на підтримку тестової бази.

Для розробки системи було використано Python і популярні бібліотеки, такі як ast для аналізу коду та pytest для тестування. Це дозволило створити ефективне та гнучке рішення для автоматизації оновлення тестів, яке застосовується до проектів з часто змінюваним кодом.

Розроблене рішенні автоматичного оновлення тестів демонструє високу точність класифікації застарілих тестів та значне скорочення часу на їх оновлення. Рішення дозволяє значно ефективність тестування програмного забезпечення та підтримки тестових сценаріїв у актуальному стані.

Розроблене програмне забезпечення успішно протестоване на реальних прикладах змін у коді, що підтверджує його ефективність у реальних умовах. Практична значимість роботи полягає в розробці інструменту для автоматизації процесу підтримки актуальності тестових сценаріїв.

Робота пройшла апробацію на II міжнародній науково-практичній конференції молодих вчених та студентів, 19 - 21 грудня 2024 року, м. Київ, Державний університет інформаційно-комунікаційних технологій.

Наукова новизна: Набуло подальшого розвитку підхід до автоматизації оновлення тестових сценаріїв через інтеграцію аналізу змін у коді та використання мовних моделей для класифікації застарілих тестів, що дозволяє значно підвищити точність та адаптивність тестової бази до змін у програмному забезпеченні.

Практична значимість: Розробка системи для автоматизації оновлення тестів дозволяє підвищити ефективність і точність підтримки тестових сценаріїв у великих програмних проєктах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Ball T. The Concept of Dynamic Analysis // Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering. — 1999. — 216-227 с.

2) Sriram I., Khajeh-Hosseini A. Research Agenda in Cloud Technologies // Proceedings of the 10th International Conference on Cloud Computing and Services Science. — 2021. — 65-75 с.

3) Nama, P., Reddy, P., Pattanayak, S. K. Artificial Intelligence for Self-Healing Automation Testing Frameworks: Real-Time Fault Prediction and Recovery // Cineforum. – 2024. – 118 с.

4) Selenium [Електронний ресурс] Режим доступу: https://www.selenium.dev/

5) Cypress [Електронний ресурс] Режим доступу: https://www.cypress.io/

6) TestComplete [Електронний ресурс] Режим доступу: https://smartbear.com/product/testcomplete/overview/

7) Appium [Електронний ресурс] Режим доступу: http://appium.io/

8) OpenAI [Електронний ресурс] Режим доступу: https://openai.com/

9) Software Testing Resource Scheduling based on Artificial Intelligence [Електронний ресурс] / C. Song, Y. Jiecong. – 2022. – Режим доступу до ресурсу: https://www.diva-portal.org/smash/get/diva2:1697322/FULLTEXT01.pdf.

10) Software Testing Resource Scheduling based on Artificial Intelligence [Електронний ресурс] / R.Pan, M. Bagherzadeh, T. A. Ghaleb, L. Briand // Test Case Selection and Prioritization Using Machine Learning: A Systematic Literature Review. – 2022. – Режим доступу до ресурсу: https://arxiv.org/abs/2106.13891

11) Saltzer J. H., Kaashoek M. F. Principles of Computer System Design: An Introduction. Elsevier Science & Technology, – 2009. – 146 с.

12) Mosleh M., Dalili K., Heydari B. Distributed or Monolithic? A Computational Architecture Decision Framework. IEEE Systems Journal. 2018. Т. 12, № 1. 125–136 с.

13) Python Programming Language [Електронний ресурс] Режим доступу: https://www.python.org/

14) Git [Електронний ресурс] – Режим доступу: https://git-scm.com/

15) PyQt5 [Електронний ресурс] – Режим доступу: https://riverbankcomputing.com/software/pyqt/intro

16) AST Module in Python [Електронний ресурс] – Режим доступу: https://docs.python.org/3/library/ast.html

17) pytest [Електронний ресурс] – Режим доступу: https://pytest.org/

18) Myers, G. J., Sandler, C., Badgett, T. The Art of Software Testing. – 3rd ed. – Hoboken: Wiley, 2011. – 256 с.

ДОДАТКИ

ДОДАТОК А

ЛІСТИНГ ПРОГРАМНОГО КОДУ

```
# controllers/main_controller.py
from PyQt5.QtCore import pyqtSlot

from ui.models.analysis_model import AnalysisModel
from ui.models.classification_model import ClassificationModel
from ui.models.git_model import GitModel
from ui.models.update_model import UpdateModel
from ui.repository_loader_widget import RepositoryLoaderWidget
from ui.views.main_view import MainView



class MainController:
    def __init__(self, api_key):

        self.git_model = GitModel()
        self.analysis_model = AnalysisModel(self.git_model)
        self.classification_model = ClassificationModel(api_key)
        self.update_model = UpdateModel(api_key)

        self.view = MainView()
        self.connect_signals()

    def connect_signals(self):
        self.view.load_repo_button.clicked.connect(self.show_repo_loader)

self.view.update_tests_widget.update_button.clicked.connect(self.replace_test_in_pro
ject)
```

```
    self.view.analyze_button.clicked.connect(self.analyze_repository)
    repo_name = self.git_model.get_repo_name()
    if not repo_name or repo_name == 'Final':
        self.view.commit_push_widget.hide()

def show_repo_loader(self):
    self.view.repo_loader = RepositoryLoaderWidget(self.on_repo_loaded)
    self.view.repo_loader.show()

def on_repo_loaded(self, repo_path):
    self.git_model.load_repository(repo_path)
    self.view.repo_loader.close()

    repo_name = self.git_model.get_repo_name()
    self.view.set_repo_name(repo_name)
    self.view.commit_push_widget.show()

    self.set_push_button_visibility(self.git_model.has_remote_repo())
    self.view.analyze_button.show()

def analyze_repository(self):
    changes = self.load_commit_changes()
    self.set_progress(20)

    self.classify_and_display_obsolete_tests(changes)
    self.set_progress(60)

    # Оновлення тестів у разі потреби
    self.update_tests()
    self.set_progress(100)
```

```
def set_push_button_visibility(self, value):
    if not value:
        self.view.commit_push_widget.push_button.hide()
        self.view.commit_push_widget.commit_and_push_button.hide()
    else:
        self.view.commit_push_widget.push_button.show()
        self.view.commit_push_widget.commit_and_push_button.show()

@pyqtSlot()
def commit_changes(self):
    message = self.view.commit_message_input.text()
    result = self.git_model.commit_changes(message)
    self.view.commit_message_input.clear()
    self.view.label.setText(result)

@pyqtSlot()
def push_changes(self):
    result = self.git_model.push_changes()
    self.view.label.setText(result)

@pyqtSlot()
def commit_and_push(self):
    message = self.view.commit_message_input.text()
    result = self.git_model.commit_and_push(message)
    self.view.commit_message_input.clear()
    self.view.label.setText(result)

def load_commit_changes(self):
    compare_to = "last_commit" if not
```

```
self.view.compare_to_working_checkbox.isChecked() else "working"
        modified_files = self.git_model.get_modified_files(compare_to)
        related_tests = self.analysis_model.analyze_changes(modified_files)

        changes_text = "\n".join(modified_files) if len(
            modified_files) > 0 else 'Немає змін у коді'
        self.view.display_git_changes(changes_text)

        result_text = "Тести, що можуть бути застарілими:\n" if len(related_tests) > 0
else 'Тести актуальні'
        for item in related_tests:
            result_text += f"\nЗмінений компонент: {item['old_name']}\n"
            for test_file_path, test_name, line_number in item['test_files']:
                result_text += f"- {test_name} (файл: {test_file_path}, рядок:
{line_number})\n"
        self.view.display_analysis_results(result_text)
        return related_tests

    def classify_and_display_obsolete_tests(self, changes):
        obsolete_tests = self.classification_model.classify_obsolete_tests(changes)
        results_text = "Застарілі тести:\n" if len(obsolete_tests) > 0 else "Застарілих
тестів не виявлено"
        for item in obsolete_tests:
            results_text += f"Файл: {item['file']}, Тест: {item['name']}, Причина:
{item['reason']}\n"
        self.view.display_obsolete_tests(results_text)
        self.classification_model.set_tests(obsolete_tests)

    def update_tests(self):
        try:
```

```
            tests_to_update = self.classification_model.tests
            updated_tests = self.update_model.update_tests(tests_to_update)
            self.update_model.set_tests(updated_tests)
            self.view.display_updated_tests(updated_tests)
        except Exception as e:
            self.view.display_error(f"Виникла помилка під час аналізу: {str(e)}")
            self.view.update_progress(0, "Помилка")

    def replace_test_in_project(self):
        try:
            if not self.update_model.tests:
                raise Exception('немає тестів')
            self.set_update_status('running')
            self.update_model.replace_multiple_tests_in_files(self.update_model.tests)
            self.set_update_status('done')
        except Exception as e:
            self.set_update_status(message=f'error {e}', error=True)

    def set_update_status(self, message, error=False):
        """
        Updates the status in UpdateTestsWidget.
        """
        self.view.update_tests_widget.set_status(message, error=error)

    def set_progress(self, value):
        self.view.progress_bar.update_progress(value)
# views/main_view.py
import os
```

```
from PyQt5.QtWidgets import QMainWindow, QVBoxLayout, QWidget, QCheckBox, QPushButton, QLabel, QHBoxLayout, QMessageBox

from ui.CommitPushWidget import CommitPushWidget
from ui.git_changes_viewer_widget import GitChangesViewerWidget
from ui.progress_bar_widget import ProgressBarWidget
from ui.results_viewer_widget import ResultsViewerWidget
from ui.obsolete_test_checker_widget import ObsoleteTestCheckerWidget
from ui.updated_test_results_widget import UpdatedTestResultsWidget
from ui.update_tests_widget import UpdateTestsWidget


class MainView(QMainWindow):
    def __init__(self):
        super().__init__()
        self.repo_loader = None
        self.setWindowTitle("Інтерфейс перевірки змін у Git")
        self.setGeometry(100, 100, 1280, 720)

        self.main_widget = QWidget()
        self.layout = QVBoxLayout(self.main_widget)

        button_layout = QHBoxLayout()

        self.repo_name_label = QLabel("Репозиторій не завантажений")
        self.repo_name_label.setStyleSheet("font-weight: bold; font-size: 16px;")
        self.layout.addWidget(self.repo_name_label)

        self.compare_to_working_checkbox = QCheckBox("Порівнювати із закоміченими змінами (working)")
```

```
self.layout.addWidget(self.compare_to_working_checkbox)

self.load_repo_button = QPushButton("Завантажити репозиторій")
button_layout.addWidget(self.load_repo_button, stretch=1)

self.analyze_button = QPushButton("Аналізувати репозиторій")
button_layout.addWidget(self.analyze_button, stretch=4)
self.analyze_button.hide()  # Приховуємо кнопку аналізу спершу

self.layout.addLayout(button_layout)

self.progress_bar = ProgressBarWidget()
self.layout.addWidget(self.progress_bar)

self.git_changes_viewer = GitChangesViewerWidget()
self.results_viewer = ResultsViewerWidget()
self.obsolete_test_checker_widget = ObsoleteTestCheckerWidget()
self.updated_test_results_widget = UpdatedTestResultsWidget()
self.update_tests_widget = UpdateTestsWidget()
self.commit_push_widget = CommitPushWidget()

self.layout.addWidget(self.git_changes_viewer)
self.layout.addWidget(self.results_viewer)
self.layout.addWidget(self.obsolete_test_checker_widget)
self.layout.addWidget(self.updated_test_results_widget)
self.layout.addWidget(self.update_tests_widget)
self.layout.addWidget(self.commit_push_widget)

self.setCentralWidget(self.main_widget)
```

```
        self.apply_stylesheet()

    def apply_stylesheet(self):
        css_path = os.path.join(os.path.dirname(__file__), '../resources/styles.css')
        with open(css_path, "r") as file:
            self.setStyleSheet(file.read())

    def set_repo_name(self, name):
        self.repo_name_label.setText(f"Вибраний репозиторій: {name}")

    def display_git_changes(self, changes_text):
        self.git_changes_viewer.display_changes(changes_text)

    def display_analysis_results(self, results_text):
        self.results_viewer.display_results(results_text)

    def display_obsolete_tests(self, results_text):
        self.obsolete_test_checker_widget.display_results(results_text)

    def display_updated_tests(self, results_text):
        self.updated_test_results_widget.display_results(results_text)

    def set_progress(self, value, message=""):
        self.progress_bar.update_progress(value)
        self.progress_bar.setToolTip(message)

    def display_error(self, message):
        error_box = QMessageBox(self)
        error_box.setIcon(QMessageBox.Critical)
        error_box.setWindowTitle("Помилка")
```

```
        error_box.setText(message)
        error_box.exec_()
# models/analysis_model.py
from analyzers.dependency_analysis_manager import DependencyAnalysisManager


class AnalysisModel:
    def __init__(self, git_model):
        self.analysis_manager =
DependencyAnalysisManager(git_model.git_integration)

    def analyze_changes(self, modified_files):
        return self.analysis_manager.analyze_changes(modified_files)
# models/classification_model.py
from analyzers.obsolete_test_classifier import ObsoleteTestClassifier


class ClassificationModel:
    def __init__(self, api_key):
        self.tests = None
        self.classifier = ObsoleteTestClassifier(api_key)

    def classify_obsolete_tests(self, changes):
        return self.classifier.classify_obsolete_tests(changes)

    def set_tests(self, data):
        self.tests = data
# models/git_model.py
from vcs.git_integration import GitIntegration
```

```
class GitModel:
    def __init__(self):
        self.git_integration = GitIntegration()

    def get_modified_files(self, compare_to="last_commit"):
        return self.git_integration.get_modified_files(compare_to)

    def load_repository(self, path):
        self.git_integration.set_repo(path)

    def get_repo_name(self):
        return self.git_integration.repo.working_dir.split('\\')[
            -1] if self.git_integration else "Репозиторій не завантажений"

    def commit_changes(self, message):
        return self.git_integration.commit_changes(message)

    def push_changes(self, remote_name='origin', branch_name='main'):
        return self.git_integration.push_changes(remote_name, branch_name)

    def commit_and_push(self, message, remote_name='origin', branch_name='main'):
        return self.git_integration.commit_and_push(message, remote_name,
branch_name)

    def has_remote_repo(self):
        try:
            remotes = len(self.git_integration.repo.remotes)
            return remotes != 0
        except:
```

```
            return False
# models/update_model.py
from analyzers.test_updater import TestUpdater


class UpdateModel:
    def __init__(self, api_key):
        self.updater = TestUpdater(api_key)
        self.tests = None

    def update_tests(self, tests_data):
        return self.updater.update_tests(tests_data)

    def replace_multiple_tests_in_files(self, generated_tests):
        self.updater.replace_multiple_tests_with_import_organization(generated_tests)

    def set_tests(self, tests):
        self.tests = tests
# managers/dependency_analysis_manager.py
import ast

from analyzers.dependency_analyzer_impl import DependencyAnalyzerImpl


class DependencyAnalysisManager:
    def __init__(self, git_integration):
        self.git_integration = git_integration
        self.dependency_analyzer = DependencyAnalyzerImpl()

    def get_changed_functions_and_classes(self, file_path, changed_lines):
```

```
        changed_elements = []

        # Читаємо стару версію коду файлу
        old_code = self.git_integration.get_file_at_commit(file_path)
        old_tree = ast.parse(old_code)

        # Читаємо нову версію коду файлу
        new_code = self.git_integration.read_file_from_repo(file_path)
        new_tree = ast.parse(new_code)
        zipp = zip(ast.walk(old_tree), ast.walk(new_tree))
        # Визначаємо змінені функції, методи чи класи
        for old_node, new_node in zipp:
            if isinstance(old_node, (ast.FunctionDef, ast.ClassDef)) and
isinstance(new_node,
                                                    (ast.FunctionDef, ast.ClassDef)):
                element_start = old_node.lineno
                element_end = getattr(old_node, 'end_lineno', None)

                # Перевіряємо, чи входять змінені рядки в межі цього елемента
                if any(element_start <= line <= (element_end or element_start) for line in
changed_lines):
                    # Зберігаємо інформацію про змінені елементи
                    old_name = old_node.name
                    old_element_code = ast.get_source_segment(old_code, old_node)
                    new_element_code = ast.get_source_segment(new_code, new_node)

                    changed_elements.append({
                        "old_name": old_name,
                        "old_code": old_element_code,
                        "new_code": new_element_code,
```

```
                "test_files": []  # Це оновимо далі в analyze_changes
            })

        return changed_elements

    def analyze_changes(self, modified_files):
        all_changes = []

        for file_path in modified_files:
            # Отримуємо список змінених рядків
            changed_lines = self.git_integration.get_changed_lines(file_path)

            # Визначаємо змінені елементи з їхньою старою і новою версіями
            changed_elements = self.get_changed_functions_and_classes(file_path, changed_lines)

            # Для кожного зміненого елемента шукаємо відповідні тести
            for element in changed_elements:
                element_name = element["old_name"]

                # Знаходимо відповідні тести
                repo_path = self.git_integration.repo_path
                related_tests = self.dependency_analyzer.find_related_tests([element_name], repo_path)

                # Додаємо інформацію про тести до об'єкта змін
                element["test_files"] = related_tests

                # Додаємо змінений елемент у загальний список змін
                all_changes.append({
```

```
                "old_name": element["old_name"],
                "old_code": element["old_code"],
                "new_code": element["new_code"],
                "test_files": element["test_files"]
            })

        return all_changes
# managers/dependency_analysis_manager.py
import ast

from analyzers.dependency_analyzer_impl import DependencyAnalyzerImpl


class DependencyAnalysisManager:
    def __init__(self, git_integration):
        self.git_integration = git_integration
        self.dependency_analyzer = DependencyAnalyzerImpl()

    def get_changed_functions_and_classes(self, file_path, changed_lines):
        changed_elements = []

        # Читаємо стару версію коду файлу
        old_code = self.git_integration.get_file_at_commit(file_path)
        old_tree = ast.parse(old_code)

        # Читаємо нову версію коду файлу
        new_code = self.git_integration.read_file_from_repo(file_path)
        new_tree = ast.parse(new_code)
        zipp = zip(ast.walk(old_tree), ast.walk(new_tree))
        # Визначаємо змінені функції, методи чи класи
```

```
        for old_node, new_node in zipp:
            if isinstance(old_node, (ast.FunctionDef, ast.ClassDef)) and isinstance(new_node,
                                                                                    (ast.FunctionDef, ast.ClassDef)):
                element_start = old_node.lineno
                element_end = getattr(old_node, 'end_lineno', None)

                # Перевіряємо, чи входять змінені рядки в межі цього елемента
                if any(element_start <= line <= (element_end or element_start) for line in changed_lines):
                    # Зберігаємо інформацію про змінені елементи
                    old_name = old_node.name
                    old_element_code = ast.get_source_segment(old_code, old_node)
                    new_element_code = ast.get_source_segment(new_code, new_node)

                    changed_elements.append({
                        "old_name": old_name,
                        "old_code": old_element_code,
                        "new_code": new_element_code,
                        "test_files": []  # Це оновимо далі в analyze_changes
                    })

        return changed_elements

    def analyze_changes(self, modified_files):
        all_changes = []

        for file_path in modified_files:
            # Отримуємо список змінених рядків
            changed_lines = self.git_integration.get_changed_lines(file_path)
```

```
            # Визначаємо змінені елементи з їхньою старою і новою версіями
            changed_elements = self.get_changed_functions_and_classes(file_path, changed_lines)

            # Для кожного зміненого елемента шукаємо відповідні тести
            for element in changed_elements:
                element_name = element["old_name"]

                # Знаходимо відповідні тести
                repo_path = self.git_integration.repo_path
                related_tests = self.dependency_analyzer.find_related_tests([element_name], repo_path)

                # Додаємо інформацію про тести до об'єкта змін
                element["test_files"] = related_tests

                # Додаємо змінений елемент у загальний список змін
                all_changes.append({
                    "old_name": element["old_name"],
                    "old_code": element["old_code"],
                    "new_code": element["new_code"],
                    "test_files": element["test_files"]
                })

        return all_changes
# analyzers/dependency_analyzer_impl.py
import ast
import os
from analyzers.dependency_analyzer import DependencyAnalyzer
```

```
class DependencyAnalyzerImpl(DependencyAnalyzer):
    def extract_functions(self, code):
        tree = ast.parse(code)
        functions = []

        for node in ast.walk(tree):
            if isinstance(node, ast.FunctionDef):
                function_info = {
                    "name": node.name,
                    "params": [arg.arg for arg in node.args.args],
                    "param_types": [self.get_annotation(arg.annotation) for arg in
node.args.args],
                    "body": self.analyze_body_structure(node.body)
                }
                functions.append(function_info)

        return functions

    def get_annotation(self, annotation):
        if annotation is None:
            return "Any"
        elif isinstance(annotation, ast.Name):
            return annotation.id
        elif isinstance(annotation, ast.Subscript):
            return f"{annotation.value.id}[{annotation.slice.id}]"
        return "Unknown"

    def analyze_body_structure(self, body):
```

```
        structure = []
        for stmt in body:
            if isinstance(stmt, ast.If):
                structure.append("if")
            elif isinstance(stmt, ast.For):
                structure.append("for")
            elif isinstance(stmt, ast.While):
                structure.append("while")
            elif isinstance(stmt, ast.Try):
                structure.append("try")
            elif isinstance(stmt, ast.With):
                structure.append("with")
            elif isinstance(stmt, ast.FunctionDef):
                # Додаємо вкладену функцію як окремий виклик
                structure.append(f"nested_function_{stmt.name}")
            elif isinstance(stmt, ast.Call):
                # Додаємо виклик функції, зберігаючи назву викликаної функції, якщо можливо
                called_function = stmt.func.id if isinstance(stmt.func, ast.Name) else 'unknown_call'
                structure.append(f"call_{called_function}")
            else:
                # Додаємо тип оператора як загальну інформацію про структуру
                structure.append(type(stmt).__name__)

        return structure

    def extract_dependencies(self, code):
        tree = ast.parse(code)
        dependencies = {}
```

```
        for node in ast.walk(tree):
            if isinstance(node, ast.FunctionDef):
                function_name = node.name
                calls = [n.func.id for n in ast.walk(node) if isinstance(n, ast.Call) and isinstance(n.func, ast.Name)]
                dependencies[function_name] = calls

        return dependencies

    def build_call_graph(self, code):
        return self.extract_dependencies(code)

    def find_test_files(self, repo_path):
        test_files = []
        for root, _, files in os.walk(repo_path):
            for file in files:
                if file.startswith("test_") and file.endswith(".py"):
                    test_files.append(os.path.join(root, file))
        return test_files

    def find_related_tests(self, changed_functions, repo_path):
        related_tests = []

        # Знаходимо всі тестові файли
        test_files = self.find_test_files(repo_path)

        for test_file in test_files:
            from utils.file_utils import read_file
            test_code = read_file(test_file)
```

```
            # Парсимо код тестового файлу
            tree = ast.parse(test_code)

            # Витягуємо залежності для тестового файлу
            test_dependencies = self.extract_dependencies(test_code)

            # Перевіряємо, чи залежить будь-який тест від змінених функцій
            for node in ast.walk(tree):
                if isinstance(node, ast.FunctionDef) and node.name.startswith("test_"):
                    # Визначаємо залежності для тестової функції
                    function_name = node.name
                    dependencies = test_dependencies.get(function_name, [])

                    # Перевіряємо, чи викликає тест змінену функцію
                    if any(func in dependencies for func in changed_functions):
                        # Додаємо шлях до файлу, назву тесту і рядок початку тесту
                        related_tests.append((test_file, function_name, node.lineno))

        return related_tests
# models/obsolete_test_classifier.py
import json
import os

from openai import OpenAI

from utils.file_utils import extract_test_code_from_file


class ObsoleteTestClassifier:
```

```
    def __init__(self, api_key=None, model_name="gpt-3.5-turbo", threshold=0.75, use_mock=False):
        self.api_key = api_key
        self.model_name = model_name
        self.threshold = threshold
        self.use_mock = use_mock
        self.client = OpenAI(api_key=api_key)

    def classify_obsolete_tests(self, changes):
        obsolete_tests = []

        for change in changes:
            test_details = self._get_test_code(change["test_files"])
            prompt = self._create_model_request(change, test_details)

            response = self._get_model_response(prompt)

            if response:
                obsolete_tests += self._parse_response(response, change, test_details)

        return obsolete_tests

    def _get_test_code(self, test_files):
        test_details = []
        for test_file, test_name, line in test_files:
            try:
                test_code = extract_test_code_from_file(test_file, line)
                test_details.append({
                    "file": test_file,
                    "name": test_name,
```

```
                "line": line,
                "code": test_code
            })
        except Exception as e:
            print(f"Помилка читання файлу {test_file}: {e}")
    return test_details

def _create_model_request(self, change, test_details):
    # Роль system пояснює завдання
    system_message = {
        "role": "system",
        "content": (
            "You are an assistant tasked with evaluating whether certain tests have become obsolete due to changes in a function or class. "
            "Only consider a test obsolete if the change directly affects the expected output or test behavior, not just internal implementation details.\n\n"

            "To determine if a test needs updating, follow these **objective criteria**:\n\n"

            "- **Changes to input/output:**\n"
            "  - If the change modifies the function's expected inputs or outputs (e.g., new parameters, removed parameters, changes in return types or values).\n"
            "  - Example: A function previously returned an integer but now returns a string.\n\n"

            "- **Changes to business logic:**\n"
            "  - If the core logic or purpose of the function/class has changed, directly impacting the results it produces.\n"
            "  - Example: A function previously calculated a sum but now calculates an
```

```
average.\n\n"

        "- **Side effect changes:**\n"
        "   - If the change introduces or modifies side effects (e.g., database writes, API calls, state changes).\n"
        "   - Example: A function now logs output to a file or modifies global state.\n\n"

        "- **Parameter validation changes:**\n"
        "   - If the change alters how input parameters are validated, such as new constraints or allowed values.\n"
        "   - Example: A function now rejects negative numbers but previously accepted them.\n\n"

        "- **Structural changes affecting test relevance:**\n"
        "   - If a function or class is renamed or moved, and the test no longer references the correct implementation.\n"
        "   - Example: A test calls a function by its old name, which has been renamed in the code.\n\n"

        "- **Implementation-only changes:**\n"
        "   - If the change only affects internal implementation details without changing inputs, outputs, or side effects, "
        "the test should remain relevant and should not be flagged as obsolete.\n"
        "   - Example: Reordering operations or optimizing logic without functional changes.\n\n"

        "Your response must be in JSON format with the following structure:\n"
        "```\n"
```

```
            "[\n"
            "    {\n"
            "        \"test_name\": \"<name of the test>\",\n"
            "        \"file\": \"<path to the test file>\",\n"
            "        \"need_update\": <true or false>,\n"
            "         \"reason\": \"<reason in Ukrainian, if need_update is true, otherwise leave empty>\"\n"
            "    },\n"
            "    ...\n"
            "]\n"
            "```\n\n"

            "Instructions:\n"
            "- `test_name`: The name of the test being evaluated.\n"
            "- `file`: Path to the test file.\n"
            "- `need_update`: Set to true only if the test's expected behavior is impacted by the change in functionality. If the test remains relevant, set it to false.\n"
            "- `reason`: If `need_update` is true, briefly explain in Ukrainian why the test is now obsolete or needs updates; leave blank if `need_update` is false.\n\n"

            "Ensure responses are precise, and avoid marking tests as obsolete due to minor, internal-only changes that do not affect the function's observable behavior."
        )
    }

    # Роль user надає деталі змін та тестів
    user_message = {
        "role": "user",
        "content": (
            f"Function/Class: {change['old_name']}\n\n"
```

```
                f"Old Version:\n{change['old_code']}\n\n"
                f"New Version:\n{change['new_code']}\n\n"
                "Associated tests:\n" +
                "\n".join(
                    f"- Test '{test['name']}' in file {test['file']}:\n{test['code']}" for test in
test_details)
            )
        }

        return [system_message, user_message]

    def _get_model_response(self, prompt):
        try:
            response = self.client.chat.completions.create(
                model=self.model_name,
                messages=prompt
            )
            return response.choices[0].message.content.strip()
        except Exception as e:
            print(f"Помилка при надсиланні запиту до OpenAI: {e}")
            return None

    def _parse_response(self, response, change, test_details):
        obsolete_tests = []

        # Parse the JSON response from the model
        try:
            response_data = json.loads(response)
        except json.JSONDecodeError:
            print("Error decoding JSON response")
```

```
            return obsolete_tests

        # Process each test in the response
        for test_info in response_data:
            if test_info.get("need_update", False):
                # Find the corresponding test detail in test_details for old_code and line information
                matching_test = next((test for test in test_details if test["name"] == test_info["test_name"]), None)
                if matching_test:
                    obsolete_tests.append({
                        "file": test_info["file"],
                        "name": test_info["test_name"],
                        "line": matching_test["line"],
                        "old_code": change["old_code"],
                        "new_code": change["new_code"],
                        "reason": test_info.get("reason", "")
                    })

        return obsolete_tests
# models/test_updater.py
import ast
import json
import os
import re

import isort
import subprocess

from utils.file_utils import extract_test_code_from_file, write_file, read_file
```

```
import openai
from openai import OpenAI



class TestUpdater:
    def __init__(self, api_key, model_name="gpt-4o"):
        self.api_key = api_key
        openai.api_key = api_key
        self.model_name = model_name
        self.client = OpenAI(api_key=api_key)

    def update_tests(self, test_data):
        updated_tests = []

        for test_info in test_data:
            # Extract existing test code
            test_code = extract_test_code_from_file(test_info["file"], test_info["line"])

            # Create a structured prompt with system and user instructions
            prompt = self._create_model_request(test_info, test_code)

            # Get updated test from OpenAI
            updated_test_code = self._get_updated_test_code(prompt)

            if updated_test_code:
                updated_tests.append({
                    "file": test_info["file"],
                    "line": test_info["line"],
                    "name": test_info["name"],
                    "updated_test": updated_test_code
```

```
            })

        return updated_tests

    def _create_model_request(self, test_info, test_code):
        # System message with guidelines for the assistant role
        system_message = {
            "role": "system",
            "content": (
                "You are an assistant for generating updated tests. Your role is to update existing tests based on "
                "changes in the function or method they test. Only make changes to the test if the function's updated "
                "behavior affects the test's expected outcomes or logic.\n\n"
                "Return your response as updated test code only, in Python syntax, without any additional explanations, "
                "comments, or descriptions. Respond in the following structure:\n\n"
                "```python\n"
                "{updated_test_code}\n"
                "```\n\n"
                "Important:\n"
                "1. If the logic, return type, or parameters in the function have changed, adjust the test code to reflect this.\n"
                "2. Do not change the test if the new function behavior does not affect the test's expected outcome.\n"
                "3. Ensure the returned code is properly formatted and adheres to Python syntax standards.\n"
                "4. Do not include any unnecessary explanations, comments, or unrelated text in the response.\n"
                "5. If  parameters count in the function have changed, adjust the test code to
```

```
reflect this."
            )
        }

        # User message providing specific data for updating the test
        user_message = {
            "role": "user",
            "content": (
                f"Please update the test based on the following information:\n\n"
                f"- **Old Function Code**:\n{test_info['old_code']}\n\n"
                f"- **New Function Code**:\n{test_info['new_code']}\n\n"
                f"- **Reason for Update**: {test_info['reason']}\n\n"
                f"- **Original Test Code**:\n{test_code}\n\n"
                "Update the test code to reflect any necessary adjustments based on the new function behavior. "
                "Return only the modified test code in Python syntax."
            )
        }

        return [system_message, user_message]

    def _get_updated_test_code(self, prompt):
        try:
            response = self.client.chat.completions.create(
                model=self.model_name,
                messages=prompt
            )
            # Extracts the updated test code from the response
            return response.choices[0].message.content.strip()
        except Exception as e:
```

```
            print(f"Error generating updated test code: {e}")
            return None

    def _find_test_function_code(self, file_content, test_name):
        tree = ast.parse(file_content)
        for node in ast.walk(tree):
            if isinstance(node, ast.FunctionDef) and node.name == test_name:
                # Extract function source code
                start_line = node.lineno - 1  # AST line numbers are 1-indexed
                end_line = node.end_lineno
                lines = file_content.splitlines()
                return "\n".join(lines[start_line:end_line])
        return None

    def replace_test_in_file(self, file_path: str, test_name: str, new_test_code: str) ->
bool:
        try:
            new_test_code = self.remove_python_wrapper(new_test_code)
            # Додаємо 4 пробіли до кожного рядка
            new_test_code = self.ensure_indentation(new_test_code)
            # Читаємо вміст файлу
            file_content = read_file(file_path)
            # Регулярний вираз для пошуку тестової функції за назвою
            test_pattern = re.compile(
                rf"(^\s*def {re.escape(test_name)}\s*\(.*?\):[\s\S]*?)(?=^\s*def |\Z)",
                re.MULTILINE
            )

            # Перевірка наявності тесту
            match = test_pattern.search(file_content)
```

```
            if not match:
                print(f"Тест з назвою '{test_name}' не знайдено у файлі.")
                return False

            # Замінюємо старий тест на новий код
            updated_content = test_pattern.sub(new_test_code + "\n", file_content)

            # Записуємо оновлений вміст у файл
            write_file(file_path, updated_content)
            return True
        except Exception as e:
            print(f"Помилка при заміні тесту у файлі {file_path}: {e}")
            return False

    def _find_test_function_range(self, lines, test_name):
        function_start_pattern = re.compile(rf"^\s*def\s+{test_name}\b")

        start_idx = None
        indentation = ""

        for i, line in enumerate(lines):
            if function_start_pattern.match(line):
                start_idx = i
                indentation = line[:len(line) - len(line.lstrip())]
                break

        if start_idx is None:
            return None, None, None  # Функцію не знайдено

        for j in range(start_idx + 1, len(lines)):
```

```
            if lines[j].strip() == "" or lines[j].startswith(indentation + " "):
                continue
            if len(lines[j]) - len(lines[j].lstrip()) <= len(indentation):
                return start_idx, j, indentation

        return start_idx, len(lines), indentation

    def replace_tests(self, updates):
        success_count = 0
        fail_count = 0
        processed_files = set()

        for update in updates:
            file_path = update["file"]
            test_name = update["name"]
            new_test_code = update["updated_test"]
            if self.replace_test_in_file(file_path, test_name, new_test_code):
                success_count += 1
                processed_files.add(file_path)
            else:
                fail_count += 1

        return {"success": success_count, "fail": fail_count}, processed_files

    def organize_imports(self, files):
        for file_path in files:
            try:
                absolute_file_path = os.path.abspath(file_path)
                isort.file(absolute_file_path)
```

```
            subprocess.run(["autoflake",  "--in-place",  "--remove-all-unused-imports",
absolute_file_path],
                           check=True)
        except FileNotFoundError as e:
            print(f"Помилка: файл {file_path} не знайдено: {e}")
        except Exception as e:
            print(f"Помилка при впорядкуванні імпортів у файлі {file_path}: {e}")

    def replace_multiple_tests_with_import_organization(self, updates):
        results, processed_files = self.replace_tests(updates)
        return results

    def remove_python_wrapper(self, new_test_code: str) -> str:
        if new_test_code.startswith("```python") and new_test_code.endswith("```"):
            return new_test_code[9:-3]
        return new_test_code.strip()

    def ensure_indentation(self,new_test_code):
        lines = new_test_code.splitlines()
        if lines and (len(lines[0]) - len(lines[0].lstrip())) >= 4:
            return new_test_code
        else:
            return "\n".join(f"    {line}" for line in lines)
```

ДОДАТОК Б

Результати перевірки роботи на співпадіння

StrikePlagiarism.com | National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute | Дата звіту 12/8/2024 | Дата редагування 12/8/2024 | Документ прийнятий

Звіт подібності

метадані

Заголовок
ІП-32мп_Ярощенко_ПЗ

Науковий керівник / Експерт
Автор **Поперешняк С.В.**

підрозділ
ФІОТ, К-а інформатики та програмної інженерії

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	2
Інтервали	A⟷	0
Мікропробіли	·	0
Білі знаки	ß	0
Парафрази (SmartMarks)	a	7

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

10	8345	65096
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	МКР Марценков КНУС-23 без економіки [redacted] 11/25/2024 National University "Lviv Politechnika" (NULP2)	22	0.26 %
2	МКР Марценков КНУС-23 без економіки [redacted] 11/25/2024 National University "Lviv Politechnika" (NULP2)	19	0.23 %
3	МКР Марценков КНУС-23 без економіки [redacted] 11/25/2024 National University "Lviv Politechnika" (NULP2)	12	0.14 %

4	МКР Марценков КНУС-23 без економіки [redacted] 11/25/2024 National University "Lviv Politechnika" (NULP2)	10	0.12 %
5	МКР Марценков КНУС-23 без економіки [redacted] 11/25/2024 National University "Lviv Politechnika" (NULP2)	10	0.12 %
6	МКР Марценков КНУС-23 без економіки [redacted] 11/25/2024 National University "Lviv Politechnika" (NULP2)	10	0.12 %
7	Аналіз ефективності чат-ботів у системах онлайн-підтримки: порівняння діалогових моделей та інтеграція з масовими онлайн сервісами [redacted] 12/4/2024 Kharkiv National University of Economics named after S.Kuznets (KNUE) (KNUE)	10	0.12 %
8	https://mate.academy/blog/python/learning-python-2023/	9	0.11 %
9	МКР Марценков КНУС-23 без економіки [redacted] 11/25/2024 National University "Lviv Politechnika" (NULP2)	8	0.10 %
10	МКР Марценков КНУС-23 без економіки [redacted] 11/25/2024 National University "Lviv Politechnika" (NULP2)	8	0.10 %

з домашньої бази даних (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з програми обміну базами даних (1.37 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	МКР Марценков КНУС-23 без економіки [redacted] 11/25/2024 **National University "Lviv Politechnika" (NULP2)**	**104 (9)**	**1.25 %**
2	Аналіз ефективності чат-ботів у системах онлайн-підтримки: порівняння діалогових моделей та інтеграція з масовими онлайн сервісами [redacted] 12/4/2024 **Kharkiv National University of Economics named after S.Kuznets (KNUE) (KNUE)**	**10 (1)**	**0.12 %**

з Інтернету (0.44 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	https://mate.academy/blog/python/learning-python-2023/	30 (4)	0.36 %
2	https://ela.kpi.ua/bitstream/123456789/32084/1/Demydov_magistr.pdf	7 (1)	0.08 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)