Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Метод генерації тестів для програмного забезпечення на основі пошуку для певних дій
Назва теми

Рівень вищої освіти Другий (магістерський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ. 200175.01.01.ПЗ

Виконав студент 2 курсу, група ІПЗм-22-1 _____ Костянтин ДУДА
Підпис Ім'я, ПРІЗВИЩЕ

Керівник канд. техн.наук, доцент _____ Оксана ЯШИНА
Науковий ступінь, звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. пед.наук, доцент _____ Наталія ПРАВОРСЬКА
Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри
інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ
Дата

Хмельницький 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Другий (магістерський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
01.09.2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дуда Костянтин Михайлович
Прізвище, ім'я, по батькові здобувача

1. Тема роботи: Метод генерації тестів для програмного забезпечення на основі пошуку для певних дій

Керівник роботи Яшина Оксана Миколаївна. канд. техн.наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 15.08.2023 р. № 30
2. Строк подання студентом роботи на кафедру 01.12.2023 р.
3. Вихідні дані до роботи Матеріали науково-дослідної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
1 Дослідження предметної області та постановка задачі
2 Концепції, моделі та методи вирішення задачі
3 Технології вирішення задачі
4 Реалізація та тестування програмного засобу
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди)

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Антиплагіат	Форкун Ю. В., доцент	23.11.2023 р.	01.12.2023 р.
Нормоконтроль	Праворська Н.І., доцент	23.11.2023 р.	01.12.2023 р.

7. Дата видачі завдання «_01_» __вересня__ 2023 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів проєкту (роботи)	Примітка
1 Вивчення предметної області; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження; формування логістичної структури кваліфікаційної роботи	01.09-10.09.2023	
2 Робота над розділом 1 кваліфікаційної роботи – вивчення літературнихта Інтернет-джерел; аналіз відомих моделей, методів та засобів за темою роботи; визначення методологічних підходів до вирішення задачі; висновки до розділу та постановка задач дослідження	11.09-25.09.2023	
3 Робота над розділом 2 кваліфікаційної роботи – розробка моделей,методів та алгоритмів вирішення задачі; висновки до розділу	26.09-10.10.2023	
4 Робота над науковими статтями	11.10-30.10.2023	
5 Робота над розділом 3 кваліфікаційної роботи – розробка інформаційноїтехнології вирішення задачі (аналіз вимог до програмного засобу та його проектування, аналіз та вибір засобів реалізації програмного засобу тощо); висновки до розділу	11.10-26.10.2023	
6 Робота над розділом 4 кваліфікаційної роботи – програмна реалізація спроектованих рішень, результати експериментів та їх аналіз; дослідження ефективності запропонованих рішень; висновки до розділу	27.10-17.11.2023	
7 Попередній захист кваліфікаційної роботи	Листопад (згідно графіка)	
8 Узгодження постановки задачі, отриманих результатів та висновків; написання вступу, загальних висновків, оформлення джерел посилання та додатків; оформлення пояснювальної записки та графічних матеріалів згідно вимог чинних стандартів	18.11-30.11.2023	
9 Перевірка роботи на наявність плагіату; нормоконтроль; брошурування пояснювальної записки; підготовка супровідних документів	01.12-04.12.2023	
10 Підготовка до захисту кваліфікаційної роботи	з 01.12.2023 р.	

Студент _____ Костянтин ДУДА
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Оксана Яшина
Підпис Ім'я, ПРІЗВИЩЕ

РЕФЕРАТ

Тема дипломної роботи: «Метод генерації тестів для програмного забезпечення на основі пошуку для певних дій».

Автор роботи: Дуда Костянтин Михайлович.

Керівник роботи: Яшина Оксана Миколаївна.

Пояснювальна записка: 93 с., 27 рис., 3 табл., 4 дод., 20 джерел.

ТЕСТУВАННЯ, ГЕНЕРАЦІЯ, АЛГОРИТМ ГЕНЕРАЦІЇ, КОДОГЕНЕРАЦІЯ, ГЕНЕТИЧНИЙ АЛГОРИТМ.

Мета даної магістерської роботи полягає в удосконаленні методу генерації тестів для програмного забезпечення, спрямованого на покращення ефективності тестування і виявлення вразливостей програм.

Об'єктом дослідження є процес генерації тестів для програмного забезпечення.

Предметом дослідження є методи та алгоритми генерації тестів з для програмного забезпечення.

У роботі використані методи дослідження: спостереження, експеримент, абстрагування, аналіз та синтез, формалізація.

Під час виконання даної кваліфікаційної роботи було проаналізовано методи та засоби тестування програмного забезпечення, визначено основні проблеми процесу тестування в галузі розробки програмного забезпечення та їх вирішення. На основі дослідження проблематики тестування удосконалено метод генерації тестів з пошуком дій та виконано програмну реалізацію даного методу.

Даний метод генерації тестів з пошуком дій полегшить процес тестування програмного забезпечення, робить його значно продуктивнішим для розробників адже етап тестування в життєвому циклі програмного забезпечення є дуже відповідальним для подальшої розробки та реалізації програмного забезпечення.

Розроблений метод тестування забезпечує генерацію тестів чорної скриньки для фреймворку Laravel. Він враховує параметри, які вказує користувач,

забезпечуючи високу продуктивність тестування програмного забезпечення, знижуючи хибні значення тест-кейсів, що тим самим збільшить якість реалізації тестів.

Даний метод був реалізований такими програмними засобами як Python в якому прописаний сам алгоритм генерації тестів та фреймворк Laravel для створення інтерфейсу користувача, API сервіс для відправки та отримання запитів для згенерованих тестів і також даний фреймворк використовувався для перевірки згенерованих даних.

Практична значимість отриманих результатів полягає в тому, що даний метод генерацій тестів може застосовуватись в сфері розробки проєктів на Laravel фреймворку, відповідно це IT-компанії, php-розробники, які бажають спростити процес тестування програмного забезпечення і також даний метод можна застосувати як комерційний проєкт у вигляді веб-ресурсу.

Підпис

07.12.2023
Дата

ABSTRACT

Topic of thesis: «Method of test generation for software based on search for specific actions.»

The author of the work: Duda Kostyantyn Mykhailovych.

Head of work: Yashina Oksana Mykolaivna.

Explanatory note: 93 p., 27 figures, 3 tables, 4 appendices, 20 sources.

TESTING, GENERATION, GENERATION ALGORITHM.

The object of research is an algorithm for generating tests for software with the search for certain actions.

The purpose of the research is to create an algorithm for generating tests for software with the search for certain actions that will simplify and speed up the process of software testing.

The following research methods and equipment were used in the work:

– observation, experiment, abstraction, analysis and synthesis, formalization;

– means of implementing design, programming and testing;

– personal computer.

During the performance of this qualification work, the field of software testing and means of testing were investigated, the main problems of the testing process in the field of software development and their solutions were determined. Based on the research of testing problems, the method of generating tests with action search was developed and the software implementation of this method was carried out.

This method of generating tests with action search will facilitate the software testing process, make it much more productive and less "scary" for developers, because the testing stage in the software life cycle is very responsible for the further development and implementation of the software.

The developed testing method ensures the generation of black box tests for the Laravel framework, taking into account the parameters specified by the user, ensuring

high performance of software testing, reducing the false values of test cases, thereby increasing the quality of test implementation.

This method was implemented with such software tools as Python, in which the test generation algorithm itself and the Laravel framework for creating a user interface, an API service for sending and receiving requests for generated tests, and this framework was used to verify the generated data.

The practical significance of the obtained results is that this proposed test generation method can be applied in the field of project development on the Laravel framework, respectively, IT companies, php developers who want to simplify the process of software testing, and this method can also be applied as a commercial project in the form of a web -resource.

Signature

07.12.2023
Date

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ – програмне забезпечення

БД – база даних

МФПД – модуль формування початкових даних

ПС – програмна система

ГА – генетичний алгоритм

ВСТУП

У сучасному світі, розвиток програмного забезпечення відіграє ключову роль у різних галузях, починаючи від інформаційних технологій та закінчуючи медициною і авіацією. При цьому надійність і безпека програмного забезпечення стають критичними аспектами, які впливають на функціонування систем і безпеку користувачів. Сфера розробки програмного забезпечення є однією з найшвидше зростаючих галузей в інформаційній та технологічній сферах. Зростання сфери розробки ПЗ може бути визначено наступними чинниками [1]:

а) зростання попиту: практично у всіх сферах життя, включаючи бізнес, охорону здоров'я, освіту, розваги, автомобільну промисловість та інші, збільшується потреба в програмному забезпеченні для автоматизації процесів та розвитку нових продуктів і послуг;

б) технологічний прогрес: постійні інновації та нові технології сприяють зростанню галузі розробки ПЗ. Це включає в себе розвиток штучного інтелекту, блокчейн та інших сучасних технологій;

в) глобалізація: компанії в усьому світі шукають талановитих розробників програмного забезпечення, що створює конкуренцію і стимулює попит на розробників;

г) освіта: зростання кількості освітніх програм і курсів з розробки ПЗ сприяє підготовці нових фахівців.

За оцінками деяких джерел, галузь розробки програмного забезпечення може зростати на 5-10% річно [2]. Проте, точні цифри можуть відрізнятися в залежності від регіону та конкретного сегменту ринку. Для актуальної статистики рекомендується звертатися до організацій, що спеціалізуються на зборі та аналізі даних у галузі інформаційних технологій.

Одним з важливих етапів в життєвому циклі розробки програмного забезпечення є тестування. Тестування програм дозволяє виявляти помилки та вразливості, перш ніж вони можуть призвести до негативних наслідків. Проте, процес тестування може бути трудомістким і витратним завданням, особливо в

великих та складних системах. У зв'язку з цим, виникає потреба в розробці ефективних методів генерації тестів, які спростять та прискорять цей процес.

Розробка методу генерації тестів з використанням кодогенерації, зокрема методу чорної скриньки, має ключове значення для різних галузей, таких як розробка програмного забезпечення, тестування і якість продукту. Ось деякі з підстав, чому цей метод є важливим і актуальним:

– покращення ефективності тестування. Генерація тестів з використанням кодогенерації дозволяє автоматизувати процес створення тестових випробувань, що робить тестування ефективнішим і швидшим;

– виявлення помилок і вразливостей. Метод чорної скриньки дозволяє тестувати програмне забезпечення з точки зору зовнішньої поведінки, допомагаючи виявити помилки, вразливості та недоліки в програмах, які можуть залишитися непоміченими при ручному тестуванні;

– збільшення тестового покриття. Автоматична генерація тестів може забезпечити більше широке тестове покриття, охоплюючи різні аспекти програми, що допомагає виявити більше помилок;

– підтримка неперервної інтеграції. Методи генерації тестів можуть бути використані для автоматичного тестування під час неперервної інтеграції, що дозволяє виявляти помилки в реальному часі;

– витрати та час. Автоматизована генерація тестів може значно зменшити витрати і час, потрібні для тестування програмного забезпечення, оскільки вона здійснюється швидше та ефективніше.

Актуальність теми полягає в тому, що з ростом складності програмного забезпечення і збільшенням обсягів даних, тестування стає все більш важливим процесом розробки ПЗ. Для підтримання високої якості програм та забезпечення їх надійності, необхідно розробляти та впроваджувати нові методи генерації тестів, які враховують специфіку програм і призводять до покращення ефективності тестування.

Мета даної магістерської роботи полягає в удосконаленні методу генерації тестів для програмного забезпечення, спрямованого на покращення ефективності тестування і виявлення вразливостей програм.

Об'єктом дослідження є процес генерації тестів для програмного забезпечення.

Предметом дослідження є методи та алгоритми генерації тестів для програмного забезпечення.

Для досягнення мети використано теоретичні та емпіричні та методи дослідження, а саме:

а) теоретичні методи:

1) абстрагування – один з важливих методів, який дозволяє відкинути несуттєві параметри (від абстрагування напряму залежить ефективність моделі);

2) аналіз та синтез – декомпозиція моделі на прості складові, виявлення зав’язків між компонентами і, відповідно, синтез цих структурних елементів у єдине ціле;

3) формалізація – представлення моделі у вигляді програмного коду;

б) емпіричні методи:

1) спостереження (темою роботи є генерація тестів, але для того щоб виділити корисні ознаки, які мають бути імплементовані у розроблюваних рішеннях, слід провести спостереження існуючих рішень, визначити властивості та зв’язки між ними);

2) експеримент (на етапі дослідження існуючих аналогів слід відтворити певні умови, які потрібні для аналізу імплементованих алгоритмів; пізніше цей же метод використовується для аналізу ефективності результативного алгоритму, який розроблено та імплементовано у ході роботи).

Новизна даної теми полягає в поєднанні методів генерації тестів із застосуванням алгоритму кодогенерації для виявлення вразливостей у програмному забезпеченні. Розроблені методи та алгоритми можуть сприяти автоматизації процесу тестування і покращити його ефективність.

Практична цінність даного методу дуже велика і охоплює різні сфери, такі як розробка програмного забезпечення, тестування, якість продукту та інші. Ось декілька аспектів практичної цінності цієї теми:

– покращення якості ПЗ. Автоматизована генерація тестів допомагає виявляти помилки та вразливості в програмах, що призводить до покращення їх якості та надійності;

– ефективність тестування. Методи генерації тестів з використанням штучного інтелекту дозволяють прискорити процес тестування, зменшити тестові витрати та забезпечити більше широке тестове покриття;

– автоматизація неперервної інтеграції: Використання автоматичних методів генерації тестів в неперервній інтеграції допомагає виявляти проблеми в реальному часі, що забезпечує вчасне виправлення помилок;

– зменшення ризиків та витрат. Генерація тестів допомагає знизити ризики виникнення критичних помилок у програмах, а також зменшити витрати на тестування та розробку;

– підтримка розробки апаратного та вбудованого ПЗ. Методи генерації тестів можуть бути застосовані до розробки апаратного та вбудованого програмного забезпечення, що робить їх популярними в сферах, де вимагається висока надійність і безпека.

Отже, дана тема має практичну цінність в різних галузях і допомагає покращувати якість програмного забезпечення, зменшувати ризики та ефективно використовувати ресурси для тестування та розробки ПЗ.

Згідно з результатами досліджень було опубліковано тези доповіді на Всеукраїнській науково-практичній конференції «Актуальні проблеми комп’ютерних наук АПКН-2023» [20]. Копії наукових публікацій подані у додатку В.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОТСАНОВКА ЗАВДАННЯ

1.1 Аналіз предметної області, останніх досліджень та джерел

Тестування програмного забезпечення (ПЗ) є важливим етапом у розробці програмних продуктів. Воно спрямоване на перевірку функціональності, надійності та безпеки ПЗ перед його впровадженням у реальне середовище. Правильно розроблені методи тестування можуть допомогти виявити та усунути помилки ще на ранніх стадіях розробки, що зменшує витрати та ризики для підприємств. Переваги та мінуси процесу тестування [3].

Переваги:

а) виправлення помилок. Виявлення та виправлення помилок на ранніх стадіях розробки дозволяє економити ресурси та час;

б) підвищення якості. Тестування допомагає покращити якість ПЗ та забезпечити задоволення користувачів;

в) зменшення ризиків. Виявлення вразливостей та потенційних проблем допомагає зменшити ризики для бізнесу;

г) вдосконалення процесу. Автоматизовані тести та гнучкі методології сприяють швидкому впровадженню змін.

Мінуси:

а) витрати. Тестування може бути часово- та ресурсомістким процесом;

б) не повний огляд. Неможливість вичерпного тестування всіх можливих сценаріїв;

в) суб'єктивність. Деякі види тестування, такі як юзабіліті, можуть бути суб'єктивними;

г) спеціалізація. Деякі види тестування вимагають спеціалізованих знань та інструментів.

У підсумку, різні типи тестування мають свої переваги та недоліки, і вибір методів [3].

Згідно проведеного дослідження вказано що тестування програмного забезпечення може займати від 30% до 50% від загального часу розробки проекту (рисунок 1.1) [4]. Це означає, що, наприклад, у дворічному проекті на розробку програмного продукту час, відведений на тестування, може становити від 7 до 12 місяців. Цей етап визнається ключовим для забезпечення якості продукту, адже вірно налаштоване тестування може виявити і виправити дефекти, сприяючи надійності та функціональності програми.

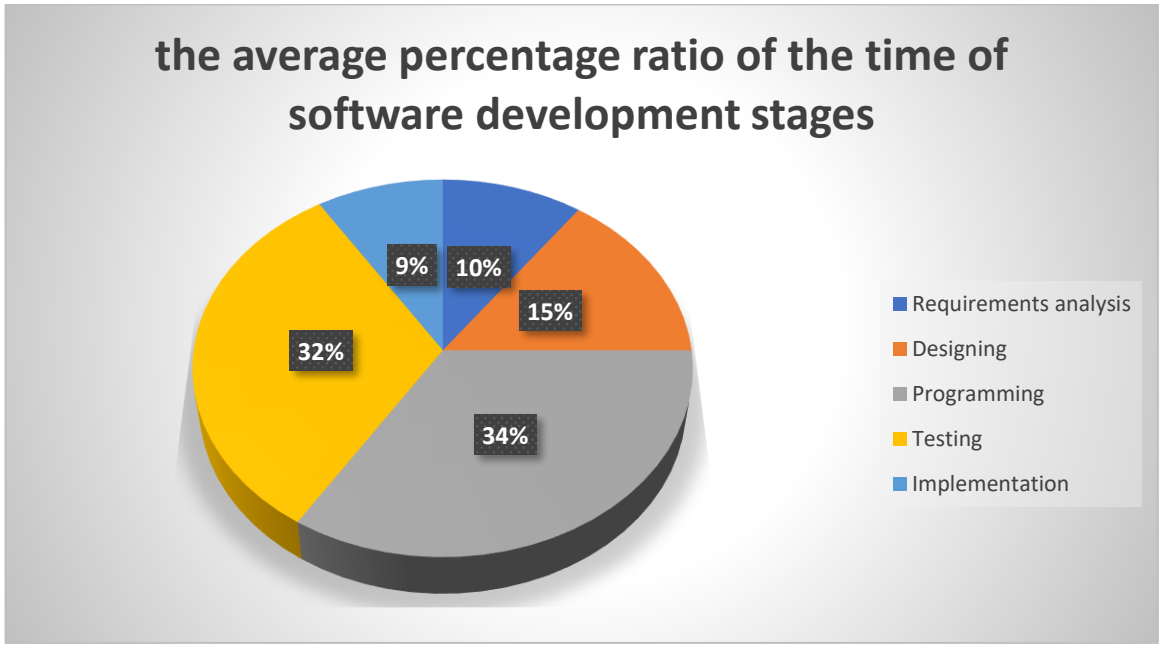


Рисунок 1.1 – Середнє процентне співвідношення часу етапів розробки ПЗ

Тестування ПЗ включає в себе різні види, такі як:

– модульне тестування. Перевірка окремих компонентів (функцій, методів) на коректність та відповідність специфікаціям. *Плюси:* Перевіряє окремі компоненти програми; виявляє локальні помилки; прискорює виправлення помилок. *Мінуси:* Не виявляє помилок взаємодії між компонентами;

– інтеграційне тестування**.** Перевірка взаємодії між компонентами та їхніх інтегрованих частин. *Плюси:* Перевіряє взаємодію між компонентами; виявляє

проблеми інтеграції. *Мінуси:* Складніше відлагоджування; може бути часово- та ресурсомістким;

– функціональне тестування. Перевірка функціональних властивостей програми відповідно до бізнес-вимог. Плюси: Перевіряє, чи відповідає програма бізнес-вимогам; ідентифікує проблеми функціональності. Мінуси*:* Не виявляє внутрішніх проблем; може бути повільним у великих системах;

– відмовостійкість (реліабельність) тестування. Оцінка, як система відновлюється після відмови та впливу аномалій. Плюси*:* Визначає стабільність системи при відмовах; допомагає у виявленні вразливостей. Мінуси*:* Може бути складним та вимагати спеціалізованих інструментів;

– юзабіліті тестування. Перевірка зручності та задоволення користувачів використання ПЗ. Плюси*:* Оцінює зручність користування; допомагає в покращенні інтерфейсу. Мінуси*:* Суб'єктивний; може бути важко автоматизувати;

– безпека тестування. Виявлення вразливостей та заходів для захисту ПЗ від загроз. Плюси*:* Виявляє вразливості та потенційні загрози; сприяє захисту від атак. Мінуси*:* Складний та спеціалізований; може бути часово- та ресурсомістким.

Останні дослідження в галузі тестування ПЗ спрямовані на вдосконалення і розширення існуючих методів. Однією з актуальних тем є автоматизація тестування, що дозволяє зменшити ручну працю та підвищити швидкість тестування. Інструменти автоматизації тестування, такі як Selenium, JUnit та TestNG, стали широко використовуваними [4].

Дослідження також фокусуються на адаптацію методів тестування до гнучких методологій розробки, таких як Scrum та Kanban. Це дозволяє впроваджувати тести на ранніх етапах розробки та прискорює процес виправлення помилок.

Сучасні інструменти тестування, такі як Jira, TestRail, та Jenkins, надають командам можливість керувати процесом тестування, створювати тестову документацію та виконувати тести автоматично.

Майбутнє тестування ПЗ обіцяє бути захопливим та інноваційним. Розвиток штучного інтелекту, машинного навчання та автоматизації допоможе вдосконалити методи тестування. Інтеграція тестування в розробку в реальному часі дозволить зменшити витрати та покращити якість ПЗ.

Усі ці фактори роблять галузь тестування ПЗ важливою та динамічною. Розуміння останніх досліджень і викликів в цій галузі важливо для розробників та фахівців з тестування, щоб підтримувати високу якість програмного забезпечення та задовольняти потреби користувачів.

Кодогенерація є потужним інструментом в сучасній розробці програмного забезпечення, спрямованим на автоматизацію процесу написання коду за допомогою попередньо визначених шаблонів та правил. Цей підхід дозволяє розробникам ефективно створювати стандартний код, зменшуючи тим самим трудомісткість розробки.

Вплив кодогенерації на сферу тестування ПЗ:

– швидкість розробки тестів. Кодогенерація може суттєво покращити швидкість написання тестів, дозволяючи автоматично створювати базові тестові випадки на основі визначених шаблонів та правил;

– уніфікація тестових сценаріїв. За допомогою кодогенерації можна впроваджувати стандарти для тестових сценаріїв, забезпечуючи, що всі тести створюються за однаковими правилами;

– мінімізація людського фактору. Автоматична генерація тестів дозволяє уникнути людських помилок та забезпечити стабільність та точність тестового покриття;

– підтримка континуальної інтеграції: Кодогенерація допомагає швидко створювати тести для нових функцій або змін у програмному коді, що є важливим для ефективної роботи в умовах континуальної інтеграції.

Виклики та переваги:

– необхідність ефективних шаблонів. Ефективна кодогенерація вимагає добре визначених шаблонів для тестування, які враховують всі можливі варіації тестових сценаріїв;

– підтримка динамічних змін. Кодогенерація повинна підтримувати зміни в програмному коді, забезпечуючи актуальність тестів при внесенні змін;

– важливість тестового покриття. Для ефективної генерації тестів, необхідно точно визначити області коду, які потребують тестування, і включити їх до процесу.

Кодогенерація може виявитися потужним інструментом для автоматизації генерації тестів, полегшуючи завдання розробників та забезпечуючи ефективність та стандартизацію в процесі тестування програмного забезпечення.

1.2 Аналіз існуючих методів та засобів забезпечення

Тестування програмного забезпечення відіграє надзвичайно важливу роль у розробці та підтримці сучасних програм і додатків. В процесі постійного зростання комп'ютеризації і залежності від програм, функціональність та надійність стали критичними факторами для задоволення потреб користувачів та досягнення бізнес-цілей.

На сьогоднішній день є два основних види тестувань [1]:

– ручне тестування;

– автоматизоване тестування.

Ручне тестування – це процес, в якому тестувальники вручну перевіряють програмне забезпечення на наявність помилок, відповідність вимогам та забезпечують якість продукту. Це включає в себе створення тестових сценаріїв, виконання їх, аналіз результатів та документування виявлених проблем. Ручне тестування використовується для перевірки аспектів, які важко автоматизувати, а також для тестування користувацького інтерфейсу та інших аспектів, які потребують людського інтуїтивного розуміння. Ручне тестування може бути

ефективним, але воно ресурсозатратне, і тому автоматизація тестування стає важливою для покращення ефективності та точності процесу тестування.

Автоматизоване тестування – це процес використання спеціальних програмних інструментів та сценаріїв для автоматичного виконання тестів програмного забезпечення. Його головна мета – зменшити ручне тестування, полегшити і прискорити процес виявлення помилок та забезпечити більшу надійність програмного продукту.

В основу принцип тестування полягає в тому щоб порівнювати фактичний та очікуваний результат певного функціоналу. Перед початком тестування складається тест-план в якому описується функціонал який тестується при певних умовах, очікуваний та фактичний результат після виконання функції (таблиця 1.1). Очікуваний результат – це результат який після виконання певної функції видає дані згідно сценарію. Фактичний результат – це той результат або стан, який фактично спостерігається після виконання тесту чи функції. Якщо фактичний і очікуваний результат співпадають – тест пройдений, якщо не співпадають – тест не пройдений [5-6].

Таблиця 1.1 – Приклад тест-плану.

№	Очікуваний результат	Фактичний результат	Статус
Тест 1	При натисканні «кнопки1» має завантажуватись текстовий файл.	При натисканні «кнопки1» текстовий файл НЕ завантажується.	Тест не пройдений
Тест 2	При натисканні «кнопки2» має закриватись «блок1».	При натисканні «кнопки2» закривається «блок1».	Тест пройдений

Загалом процес тестування є важким, масивним процесом життєвого циклу програмного забезпечення але в першу чергу є надзвичайно важливим для його розробки.

Для забезпечення самого процесу генерації тестів застосовується відповідно процес кодогенерації. Кодогенерація – це ефективний та інноваційний підхід до розробки програмного забезпечення, який дозволяє автоматизувати

процес створення програмного коду за допомогою визначених правил та шаблонів. Цей метод знайшов широке застосування в сучасному світі розробки, прискорюючи та спрощуючи процес написання коду в різних областях, від роботи з базами даних до розробки веб-застосунків.

Застосування в розробці ПЗ. Системи управління базами даних. У роботі з базами даних кодогенерація дозволяє автоматично створювати моделі даних, запити та з'єднання на основі визначених сутностей та їх взаємозв'язків. Наприклад, Hibernate для Java дозволяє генерувати код для взаємодії з базою даних на основі об'єктно-реляційного відображення.

У веб-розробці інструменти, такі як Swagger Codegen, допомагають автоматично створювати клієнтський та серверний код на основі специфікації API. Це спрощує інтеграцію клієнтів та серверів, зменшуючи кількість ручного коду.

Інструменти розробки.

Кодогенерація використовується у багатьох інструментах розробки, таких як Yeoman, який генерує стартові файли та структури проектів на основі шаблонів. Це дозволяє розробникам швидко створювати основу проекту і приступати до написання специфічного коду.

Принципи кодогенерації:

– метамоделювання**.** Визначення метамоделі, яка описує структуру та правила генерації коду;

– шаблони. Використання шаблонів для опису вигляду та структури генерованого коду;

– автоматизація. Використання інструментів, що автоматично генерують код на основі вхідних даних та шаблонів;

– конфігурація. Можливість налаштовувати правила генерації за допомогою конфігураційних файлів.

Переваги кодогенерації:

– швидкість розробки. Зменшує час, необхідний для написання стандартного коду;

– стандартизація. Забезпечує використання єдиної структури та підходів у всьому проекті;

– менша ймовірність помилок. Зменшує кількість помилок, пов'язаних з ручним написанням.

На основі проведеного дослідження у сфері інформаційних технологій можна зробити висновок, що тенденція до автоматизації процесів стає все більш прогресивною. Автоматична генерація тестів для програмного забезпечення виходить на передній план як важливий етап розробки. За таких умов виникає необхідність глибокого вивчення та реалізації даної теми для підтримки швидкого та ефективного тестування програмних продуктів.

1.3 Методологічні підходи до вирішення задачі генерації тестів.

Згідно проведеного аналізу все більше проектів тестуються завдяки автоматизованому тестуванню. Це обумовлено зростанням розмірів програмних продуктів і вимог до їх якості. Автоматизовані тести дозволяють виконувати тестування швидше та ефективніше, сприяючи виявленню дефектів на ранніх етапах розробки та зменшенню часу регресійного тестування під час змін в коді.

В ідеалі, ефективна стратегія тестування може включати в себе як ручні, так і автоматизовані тести. Ручні тести можуть бути ефективними для виявлення неочікуваних проблем, тоді як автоматизовані тести можуть забезпечити швидке виявлення регресійних дефектів. Але згідно кваліфікаційної роботи буде йти мова про генерацію тестів для автоматизованого тестування, адже ручне тестування не підходить до даної юрисдикції. Тому буде акцентовано увагу на генерацію тестів для автоматизованого тестування, оскільки ручне тестування не є оптимальним

методом для вирішення завдань, пов'язаних із значним обсягом коду чи різноманітністю сценаріїв взаємодії.

Грамотне проєктування алгоритму генерації тестів дозволить створити надійну систему із застосуванням кодогенерації. Перед початком необхідно визначитись які проблеми та завдання може вирішувати даний метод.

Перше питання яке має вирішувати даний метод це значення покриття коду. Покриття коду в тестуванні є метрикою, яка визначає, яка частина програмного коду була використана або протестована під час виконання тестів. Ця метрика вказує на те, наскільки великою є кількість коду, яка була охоплена тестами, і може слугувати показником ефективності тестування, зазвичай вимірюється у відсотках. Покриття коду допомагає виявити непротестовані або малопротестовані частини програми, що може бути корисним для підвищення якості програмного забезпечення та виявлення потенційних вразливостей чи помилок. Покриття коду визначається за формулою 1 [7].

$$\text{Покриття коду (\%)} = \left[\frac{\text{Кількість відзначених (протестованих) рядків коду}}{\text{Загальні кількість рядків коду}}\right] \cdot 100\% \qquad (1)$$

Ця формула визначає відсоток рядків коду, які були використані в тестах, у порівнянні з загальною кількістю рядків коду у програмі. Чим вищий відсоток, тим більше коду було покрито тестами.

Другий параметр – це визначення функціоналу для тестування або область застосування тестування. Область застосування тестування визначає межі, в яких тестові випробування будуть застосовані для перевірки функціональності або якості програмного забезпечення. Це охоплює різні аспекти від тестування окремих функцій до тестування великих систем та інтеграції між різними компонентами. Область застосування може включати в себе тестування різноманітних аспектів, таких як функціональність, продуктивність, безпека, сумісність та інші, залежно від конкретних вимог проекту. Задача тестування - переконатися, що програмне забезпечення відповідає очікуванням та працює

стабільно та ефективно в рамках визначеної області застосування. Основні види області тестування є наступні:

а) запити:

1) перевірка правильності структури та синтаксису запитів;

2) визначення часу виконання запитів.

б) форми:

1) перевірка обов'язкових полів у формі;

2) валідація введених даних;

3) взаємодія з різними типами полів (текстові, числові, випадаючі списки тощо);

4) тестування можливостей введення та редагування даних.

в) навігація:

1) перевірка правильності переходу між різними сторінками або екранами;

2) тестування внутрішньої системної навігації.

г) автентифікація та авторизація:

1) перевірка правильності введення логіну та пароля;

2) перевірка доступу користувачів до різних ресурсів згідно їх ролей.

д) безпека:

1) перевірка на вразливості, такі як SQL-ін'єкції та перехоплення сесій;

2) тестування обробки помилок та виключень.

е) функціональні опції:

1) перевірка роботи різних функціональних опцій та можливостей системи;

2) тестування алгоритмів та логіки системи.

ж) мобільність:

1) тестування відображення на різних розмірах екранів та пристроях;

2) перевірка функцій, специфічних для мобільних платформ.

з) інтеграція:

1) тестування взаємодії системи з іншими програмами або сервісами;
2) перевірка обміну даними між різними компонентами системи.

и) дані:
1) перевірка коректності зберігання, оновлення та вилучення даних;
2) тестування роботи з базами даних.

к) інтерфейс:
1) перевірка коректності відображення елементів інтерфейсу;
2) тестування взаємодії користувача з елементами інтерфейсу.

Вибір функціоналу для тестування є важливою складовою процесу розробки програмного забезпечення, оскільки від нього залежить ефективність та якість тестового покриття. Правильно обрані функціональні вимоги і сценарії тестування відображають реальне використання програмного продукту і дозволяють виявити потенційні проблеми або недоліки. Це сприяє створенню надійних, функціональних та відповідних вимогам продуктів, а також економії часу та ресурсів, які витрачаються на тестування.

Третій фактор, який впливатиме на розробку даного методу це відповідно це ефективність алгоритму який здійснює генерацію тестів. Обираючи правильний алгоритм, розробники можуть оптимізувати процес генерації тестів, забезпечуючи максимальну ефективність та покриття коду, оптимальної продуктивності для швидкого та ефективного тестування. Основна вимога до даного фактору – алгоритм повинен бути ефективним і здатним генерувати тести в обґрунтований час. Зрозуміло, що тут мається на увазі поняття про швидкість і ефективність роботи алгоритму.

Четвертий фактор, який буде впливати на визначення розробки алгоритму генерації тестів це можливість налаштування параметрів [8]. Налаштування параметрів для генерації тестів включає в себе виважену оптимізацію, щоб забезпечити оптимальний результат у вигляді ефективності та покриття. Ось деякі ключові параметри та їх можливі налаштування:

– кількість ітерацій алгоритму кодогенерації. Тут йде мова про налаштування та визначення кількість поколінь або ітерацій алгоритму та визначення оптимального балансу між забезпеченням достатньої кількості популяції тестів та обмеженням обчислювальних витрат;

– розмір популяції тестів. Тут необхідно враховувати зміни кількості тестів у популяції, яка використовується для генерації нових тестів. Більша кількість тестів може призвести до більшого різноманіття, але може також призвести до збільшення обчислювальних витрат;

– мутації та кросовер. Тут йде мова про визначення інтенсивності мутацій та кросоверу. Зміни в мутаціях і кросовері можуть впливати на рівень різноманітності та швидкість збіжності;

– функція придатності. Тут необхідно врахувати формування та оптимізація функції придатності для відбору тестів з врахування ключових властивостей програми, які потрібно випробувати. Адже завдяки оцінці придатності має визначатись який тест зі всіх згенерованих поколінь є найкращий та який буда враховуватись для застосування в тестуванні ПЗ. Принцип даного методу зображено на рисунку 1.2;

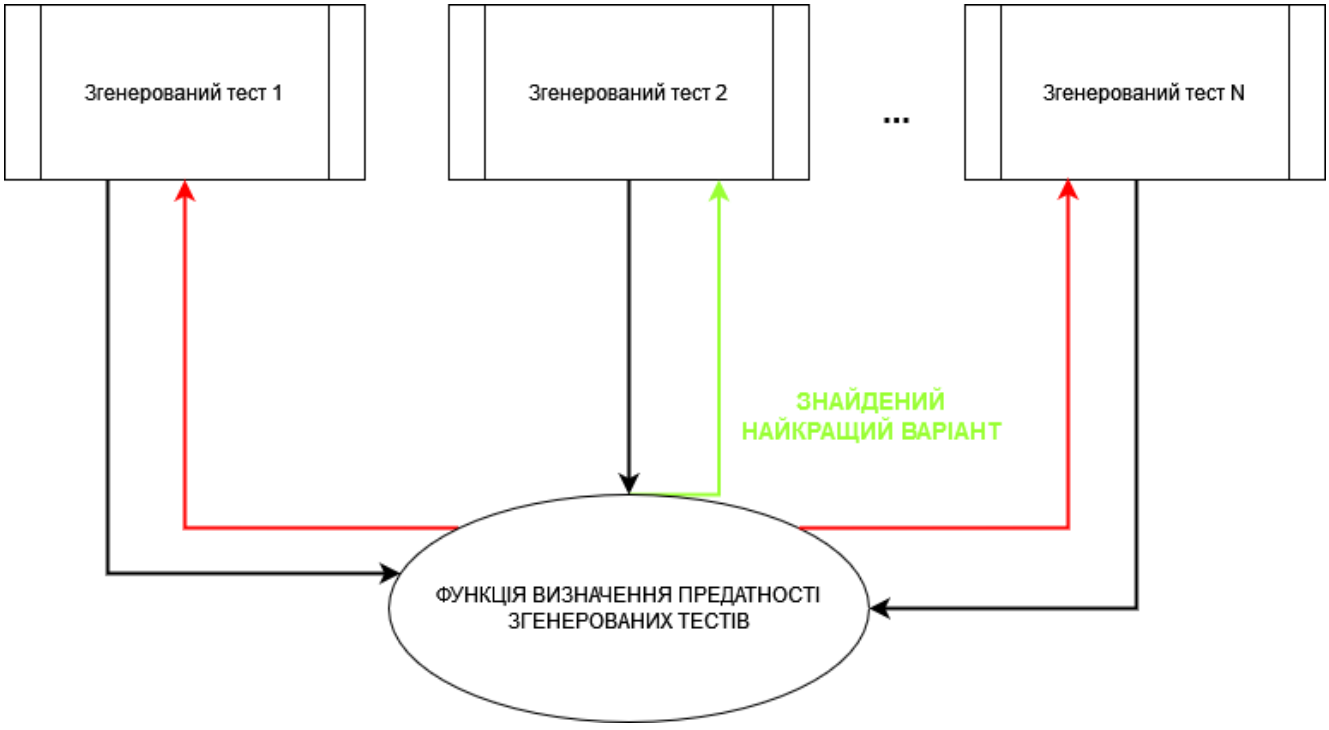


Рисунок 1.2 – Принцип визначення функції придатності.

– вагові коефіцієнти для різних видів тестів**.**

Тут йде мова про налаштування вагових коефіцієнтів для різних видів тестів (наприклад, випадкові тести, граничні умови). Вагові коефіцієнти для різних видів тестів можуть бути визначені для врахування важливості різних аспектів тестування. Такі коефіцієнти допомагають алгоритму генерації тестів приділяти більше чи менше уваги конкретним аспектам тестових випадків.

Останній параметр який необхідно врахувати це різноманітність тестових сценаріїв. Різноманітність тестових сценаріїв. Алгоритм повинен генерувати тести для різних сценаріїв та обставин. Це робиться з метою забезпечення розмаїття тестів для виявлення різних видів помилок та аномалій. Цей параметр важливий для забезпечення широкого покриття можливих сценаріїв взаємодії з програмним забезпеченням. Чим вище рівень різноманітності, тим більше різних умов та ситуацій враховується в тестових випробуваннях [9].

Визначення та контроль параметра різноманітності тестових сценаріїв може залежати від конкретного методу генерації тестів. Наприклад:

– генетичне програмування**:** у цьому випадку різноманітність може контролюватися параметрами селекції, кросоверу та мутації. Збереження різноманітності може сприяти тому, щоб алгоритм не ставав занадто специфічним або «застрягав» в певних сценаріях;

– символьне виконання. Тут різноманітність може контролюватися вибором символьних входів та умов для генерації тестів. Додавання нових символьних виразів чи зміна умов може впливати на різноманітність генерованих тестів;

– машинне навчання. У моделях машинного навчання для генерації тестів, різноманітність може контролюватися різними параметрами моделі, такими як глибина дерева рішень, кількість шарів у нейронних мережах тощо.

Визначення оптимального рівня різноманітності визначається вимогами до тестування, характеристиками програмного забезпечення та специфічними вимогами до тестових сценаріїв. Збалансоване використання різноманітності може покращити виявлення дефектів та забезпечити повне покриття функціональності програми.

Подальші дослідження у цьому напрямку полягають у проєктуванні та програмній реалізації відповідного методу генерації тестів з пошуком певних дій. Значна увага буде приділятися розробці алгоритмів, які дозволять ефективно визначати та враховувати потрібні дії для тестування в різних контекстах програмного забезпечення. Також, важливим напрямком буде вдосконалення методів автоматичного аналізу вихідних кодів програм для автоматичного виявлення можливих шляхів виконання програми та формування тестових сценаріїв. Під час подальших досліджень обговорюватиметься можливість використання методів кодогенерації для покращення ефективності та точності генерації тестів. Розглядатиметься можливість створення моделей, які здатні враховувати специфіку конкретного програмного продукту та виокремлювати ключові дії для тестування.

1.4 Висновки та постановка завдання.

У зв’язку розвитку сфери розробки ПЗ виникає потреба розробляти методи для вдосконалення тестування ПЗ. А це, у свою чергу, породжує потребу в розробці алгоритму генерації тестування ПЗ яка дане можливість підвищити швидкість, якість та продуктивність процесу тестування, зробить даний етап проєктування ПЗ значно легшим і не таким ресурсозатратним. Звичайне не можливо розробити метод який 100% спростить процес тестування, оскільки складність розробки тестів значною мірою залежить від різноманітності програмного забезпечення та його функціональних особливостей. Однак, можна

спробувати зменшити вартість і трудомісткість процесу шляхом застосування ефективних методів та інструментів.

Одним з ключових аспектів є використання методів генерації тестів, спрямованих на автоматизацію процесу тестування. При цьому важливо розробляти методи, які б не лише враховували різноманітні можливості програм, але й працювали ефективно в різних сценаріях використання.

Основними завдання даної кваліфікаційної роботи є наступні:

– визначення області тестування ПЗ;

– розробка самого алгоритму генерацій тестів з пошуком дій;

– здійснити аналіз роботи даного алгоритму і визначення його ефективності.

Актуальність роботи полягає в тому, що з ростом складності програмного забезпечення і збільшенням обсягів даних, тестування стає все більш важливим завданням. Для підтримання високої якості програм та забезпечення їх надійності, необхідно розробляти та впроваджувати нові методи генерації тестів, які враховують специфіку програм і призводять до покращення ефективності тестування.

Мета даної магістерської роботи полягає в удосконаленні методу генерації тестів для програмного забезпечення, спрямованого на покращення ефективності тестування і виявлення вразливостей програм.

Об'єктом дослідження є процес генерації тестів для програмного забезпечення.

Предметом дослідження є методи та алгоритми генерації тестів з для програмного забезпечення.

Відповідно до мети можна виділити наступні задачі дослідження:

– провести аналіз предметної області;

– дослідити концепції кодогенерації;

– визначити та дослідити алгоритми які на основі вхідних даних та обраної області тестування допоможуть реалізувати сам метод генерації тестів;

– виконати проектування відповідного програмного засобу;

– провести тестування та практичну апробацію отриманих результатів;

– дослідити ефективність алгоритму;

– проаналізувати результати дослідження та сформувати рекомендації щодо доцільності їх впровадження.

Наступним етапом роботи є проведення аналізу алгоритмів, які допоможуть безпосередньо реалізувати метод генерації тестів, та обрати найбільш доцільний для реалізації поставленого завдання, визначення моделей та методів вирішення даного завдання, визначення метрик оцінювання результатів генерації алгоритмів.

2 КОНЦЕПЦІЇ, МОДЕЛІ ТА МЕТОДИ ВИРІШЕННЯ ЗАДАЧІ ГЕНЕРАЦІЇ ТЕСТІВ З ПОШКОМ ДІЙ

2.1 Основний принцип побудови моделі методу генерації тестів.

На основі проведеного аналізу в попередньому було описано основні вимоги та завдання до розробки методу генерації тестів. Основний принцип методу генерації тестів з пошуком дій полягає в створенні специфічного методу для програмного забезпечення, які спрямовані на генерацію тестів за вказаними параметрів дій користувача. Цей підхід є важливим етапом у вдосконаленні процесу тестування, оскільки дозволяє зосередитися на взаємодії програми з користувачем, а не лише на внутрішній структурі коду.

Метод генерації тестів з пошуком дій було реалізовано з метою генерації тестів для автоматизованого тестування для модульних тестів, де важливо визначити та перевірити специфічні дії користувача та їх взаємодію з програмним забезпеченням.

Модульні тести – це частина юніт-тестування, що становить основну ланку в процесі тестування програмного забезпечення. Структура модульних тестів може бути організована за певними принципами для забезпечення ефективності та якості тестування. Основні компоненти структури модульних тестів виглядають приблизно так (Рисунок 2.1):

– набір даних для тестування. В даному компоненті визначаються вхідні дані, необхідні для виконання конкретного модульного тесту. Ця фаза має на меті групування тестових випадків за різними конфігураціями вхідних даних, забезпечуючи повне покриття можливих варіантів використання модуля;

– виконання сценарію тестування. В даному компоненті проводиться виклик конкретного модульного тесту, який включає передачу набору вхідних даних та отримання результатів від тестованого модуля. Цей етап ретельно

документує кроки виконання тесту та спостереження за взаємодією тестованого модуля з іншими компонентами системи;

– перевірка та порівняння очікуваного та фактичного результату. Перевірка та порівняння очікуваного та фактичного результату – фаза тестового процесу, що включає аналіз отриманих результатів тесту та їх порівняння з очікуваними. Застосовуються різні методи порівняння, включаючи оператори та порівняння значень. Ця фаза дозволяє визначити, чи пройшов тест, і виявити можливі дефекти в роботі модуля.

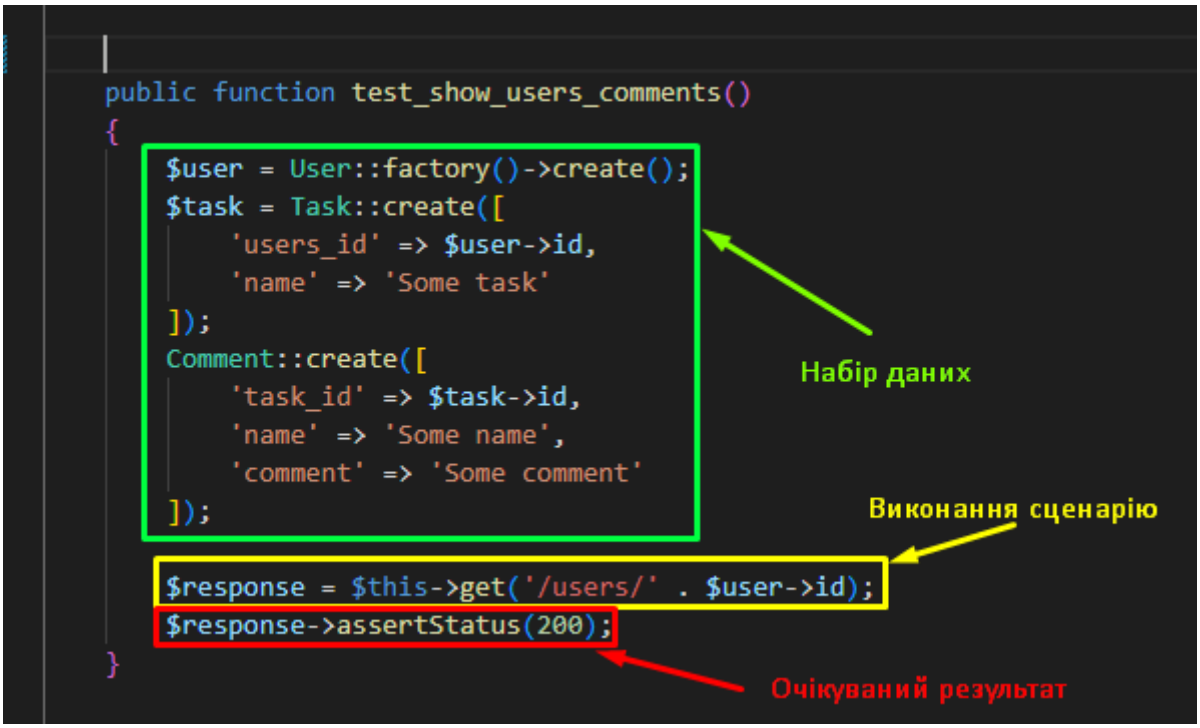


Рисунок 2.1 – Структура модульного тесту

Наведена структура є важливим компонентом у формуванні початкових даних для алгоритму кодогенерації. Її обґрунтована система включає три ключові етапи, що ретельно розроблені для забезпечення повноцінного використання алгоритму в контексті генерації тестів. Така системна структура становить невід'ємну частину методології кодогенерації, готуючи фундамент для точного та ефективного процесу генерації тестів.

Наступний момент є формування основ принципу алгоритму генерації тестів. На рисунку 2.2 продемонстрована схема роботи алгоритму генерації тестів з пошуком дій. Схему структури методу генерації тестів наведено на рисунку А.3 (додаток А).

Даний алгоритм згідно схеми поділений на 5 основних модулів які забезпечують життєдіяльність алгоритму. Ці модулі є наступні:

– вхідні дані;
– модуль формування початкових даних;
– алгоритм кодогенерації;
– перевірка придатності згенерованих тестів;
– вивід результатів.

Рисунок 2.2 – Структура методу генерації тестів

Вхідні дані. В алгоритмі генерації тестів вхідні дані представляють собою параметри, значення та умови, які визначають конкретний тестовий випадок. Це може включати вхідні дані для функцій, методів чи компонентів програмного забезпечення, а також контекстові умови, які впливають на виконання тесту. Визначені та структуровані вхідні дані дозволяють алгоритму ефективно генерувати тестові сценарії, враховуючи різноманітні аспекти програмного забезпечення та його взаємодії з різними ситуаціями. Саме в даному модулі кінцевий користувач умовно може керувати даним методом, налаштовуючи ціль тестування, пошуки дій і інші параметри, відправляючи дані в наступний важливий модуль

Модуль формування початкових даних. Основна мета даного модулю це визначення умовної «адаптація» між вхідними даними користувача та самим алгоритмом кодогенерації. Тобто основна задача даного модуля приймати вхідні дані від користувача, перекладати їх на мову алгоритму кодогенерації. Схема роботи даного методу продемонстрована на рисунку 2.3.

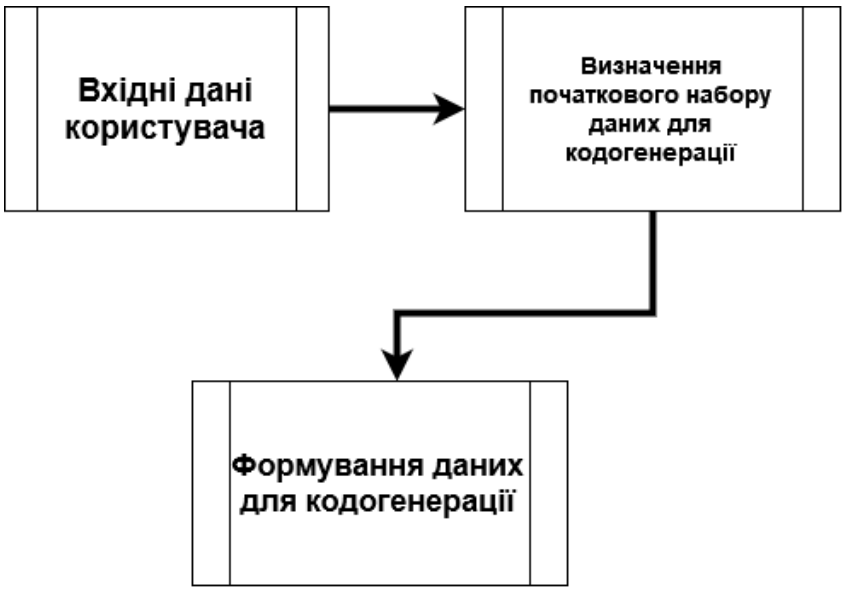


Рисунок 2.3 – Структура МФПД

Даний модуль приймає вхідні дані від користувача. На основі цих даних формує базову вибірку(про яку буде описано в наступному розділі) і відповідно всі сформовані дані передає алгоритму генерації який буде формувати відповідно тести.

Останні модулі методу це оцінка придатності сформованих тестів і вивід на екран користувача. Оцінка придатності сформованих тестів алгоритмом генерації тестів визначається за допомогою критеріїв, таких як покриття коду, ефективність виявлення помилок, повнота тестового покриття та інші параметри. Мета – забезпечити, оцінювання якості згенерованих тестів з метою вибірки найбільш доцільних варіантів для виводу результатів алгоритму кодогенерації.

2.2 Концепції алгоритмів для застосування методу генерації тестів

Задача генерації тестів є ключовою в розробці програмного забезпечення, оскільки вона забезпечує перевірку правильності та стабільності програмних продуктів. Однак, коли розглядається вирішення задачі генерації тестів з пошуком певних дій, що моделює зовнішню поведінку системи, потрібно врахувати ряд унікальних викликів. В цьому контексті розглядаються різні методологічні підходи:

- символічне виконання;
- модельна перевірка;
- генетичне програмування;
- використання машинного навчання.

Символічне виконання – це метод аналізу програм, що використовує символьні значення (symbolic values) замість конкретних вхідних даних. Замість того, щоб використовувати реальні значення, як це робить конкретне виконання коду, символічне виконання оперує змінними, що представляють класи значень.

Символічне виконання дозволяє створювати символьні вхідні дані, які представляють класи даних. Замість конкретного числа використовується

символьне значення, яке представляє усі можливі значення даного типу даних. Символічне виконання аналізує усі можливі шляхи виконання програми для символьних вхідних даних [10]. Це дозволяє визначити, як програма поводиться для різних класів вхідних значень. На основі аналізу створюються умови для тестів, які вказують, які значення вхідних даних призводять до різних шляхів виконання програми. За допомогою отриманих умов генеруються тестові випробування, спрямовані на виклик різних шляхів виконання програми.

Переваги:

– покриття коду. Символічне виконання дозволяє досягти високого покриття коду, оскільки воно розглядає усі можливі шляхи виконання;

– виявлення складних помилок. Здатність аналізувати всі можливі варіанти виконання дозволяє виявляти складні помилки, що можуть бути ускладнені для виявлення за допомогою інших методів.

Недоліки:

– висока обчислювальна складність. Символічне виконання може бути обчислювально-витратним, особливо для великих програм;

– труднощі в роботі зі складними умовами. Обробка складних умов або складно-структурованих програм може бути важкою;

– проблеми з точністю. У деяких випадках символічне виконання може надавати приблизні результати, особливо при роботі зі складними взаємодіями в програмах.

Модельна перевірка – це метод верифікації програмного забезпечення, що базується на аналізі моделей системи. У контексті генерації тестів, цей метод може бути використаний для перевірки коректності програм шляхом перевірки властивостей їхніх моделей [12].

– створення моделей програми. Модельна перевірка передбачає побудову моделей системи, які точно описують її структуру і поведінку;

– формалізація властивостей. Задаються формальні властивості, які має відповідати програма, такі як відсутність дедлайнів, взаємовиключення тощо.

– перевірка властивостей за допомогою моделі. Використовуючи алгоритми модельної перевірки, система перевіряється на відповідність заданим властивостям за допомогою її моделі;

– генерація тестів на основі результатів. Якщо перевірка показала наявність помилок, можна генерувати тестові випробування, які призведуть до виявлення цих помилок.

Переваги:

– висока точність виявлення помилок. Модельна перевірка дозволяє виявляти навіть складні логічні помилки та проблеми паралельної роботи програми;

– гарантована повнота. При правильному використанні, модельна перевірка може забезпечити повноту виявлення помилок в межах заданих властивостей.

Недоліки:

– обчислювальні витрати**.** Деякі задачі модельної перевірки можуть бути обчислювально витратними, особливо для великих програм;

– проблеми масштабованості. Для великих програм і складних систем модельна перевірка може зіткнутися з проблемами масштабованості;

Генетичне програмування – це потужний підхід для генерації коду тестування чорних скриньок та автоматизації тестування [13]. Воно використовує принципи еволюції та відбору для створення оптимальних тестових сценаріїв. Загальний опис процесу використання генетичного програмування (Рисунок 2.4):

– створення початкової популяції. Генерується велика кількість початкових тестових сценаріїв або хромосом, кожна з яких представляє собою можливий тест;

– оцінка пристосованості. Кожен тест оцінюється за допомогою функції пристосованості, яка визначає, наскільки добре він виконує свою роботу;

– схрещування. Тестові сценарії об'єднуються (схрещуються) для створення нових тестів, які можуть успадковувати корисні характеристики батьківських тестів;

– мутація. Деякі тестові сценарії піддаються випадковим змінам, щоб розширити різноманітність тестів;

– відбір. Вибираються кращі тестові сценарії на основі їхньої пристосованості, і вони використовуються для створення наступного покоління тестів;

– повторення. Процес схрещування, мутації та відбору повторюється протягом кількох поколінь.

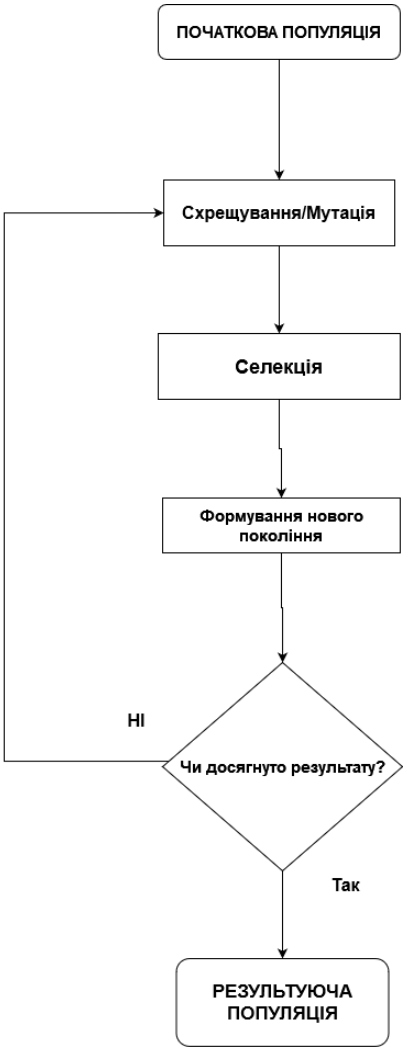


Рисунок 2.4 – Приклад алгоритму генетичного програмування

Плюси генетичного програмування:

– адаптивність. Генетичне програмування може адаптуватися до різних умов і вирішувати складні задачі, які можуть змінюватися з часом;

– можливість знаходження оптимальних рішень: Генетичне програмування дозволяє вирішувати оптимізаційні задачі, знаходячи оптимальні комбінації параметрів;

– робота з нелінійними, некерованими задачами. Воно ефективно вирішує завдання, які важко або неможливо вирішити іншими методами;

– можливість врахування багатьох критеріїв. Генетичне програмування може оптимізувати не тільки один параметр, але і кілька, що важливо в багатьох задачах.

Мінуси генетичного програмування:

– висока обчислювальна складність. Генетичне програмування може бути обчислювально-витратним, особливо для складних завдань та великих популяцій;

– не завжди гарантує знаходження глобального мінімуму/максимуму. Існує ризик застрягання в локальних оптимумах, не забезпечуючи гарантії знаходження глобальної оптимальності;

– потребує великої кількості параметрів налаштування. Ефективне використання генетичного програмування вимагає досвіду та експертизи для налагодження його параметрів;

– можливий ризик перенавчання. Як і багато інших алгоритмів машинного навчання, генетичне програмування може стикатися з проблемою перенавчання, особливо, якщо використовується на малих вибірках даних.

Незважаючи на ці виклики, генетичне програмування залишається потужним інструментом для рішення складних завдань оптимізації та пошуку рішень в просторі можливих конфігурацій.

Метод генерації тестів з використанням алгоритму генетичного програмування зосереджується на створенні тестових випадків для програмного забезпечення з точки зору зовнішньої поведінки (метод чорної скриньки). Основною метою є визначення, як програма взаємодіє з вхідними даними і генерує вихідні результати, без необхідності знань про її внутрішню структуру.

В результаті, генетичне програмування може створити оптимальні тестові сценарії, які відповідають вимогам тестування. Цей підхід особливо корисний для автоматизованого тестування складних програмних систем.

Використання алгоритмів машинного навчання для генерації тестів є новаторським підходом. Моделі можуть навчатися на основі попередніх тестових випробувань та використовувати ці знання для створення нових тестів.

Машинне навчання може бути використане для генерації тестів у різних областях, таких як програмне забезпечення, мережі, аналіз даних та інші. Основні методи включають [11]:

– генерація тестів на основі вхідних/вихідних даних. Моделі можуть бути навчені розпізнавати патерни у вхідних та вихідних даних, щоб автоматично створювати тестові набори;

– автоматичне визначення граничних значень. Машинне навчання може допомогти визначити граничні значення для параметрів, що полегшує створення тестових випадків на основі крайових умов;

– класифікація тестових випадків. Моделі можуть класифікувати тестові випадки за їхньою важливістю або ймовірністю виявлення помилок;

– аналіз покриття. Машинне навчання може використовуватися для аналізу покриття коду тестами та визначення областей, які потребують додаткового тестування.

Переваги машинного навчання в генерації тестів:

– швидкість та ефективність. Моделі можуть автоматично створювати тестові випадки швидше, ніж ручне тестування;

– складність аналізу даних: Машинне навчання може розпізнавати та аналізувати складні патерни в даних, що робить його ефективним для генерації тестів у складних системах;

– автоматизація процесу. Моделі можуть автоматично адаптуватися до змін у коді та виконувати генерацію тестів без значного втручання.

Недоліки машинного навчання в генерації тестів:

– потреба в об'ємних даних: Деякі методи машинного навчання вимагають великої кількості даних для навчання ефективних моделей;

– навчання на помилках. Моделі можуть навчатися на існуючих помилках у коді, що може призвести до втручання дефектів у тестах;

– складність інтерпретації. Деякі моделі машинного навчання можуть бути складними для інтерпретації, що робить їх менш прозорими для тестувальників;

– не всі типи тесті. Машинне навчання може бути менш ефективним для деяких видів тестів, таких як тести на продуктивність або тести на безпеку.

Машинне навчання в генерації тестів може бути потужним інструментом, особливо у поєднанні з іншими методами тестування. Важливо усвідомлювати його переваги та недоліки та використовувати його там, де воно є належним.

Застосування генетичного алгоритму для генерації тестів може бути більш доцільним порівняно з іншими алгоритмами з наступних причин:

– гнучкість. Генетичні алгоритми можуть ефективно працювати у складних, нелінійних просторах пошуку, що робить їх гнучкими та придатними для різноманітних завдань;

– глобальний пошук**.** Генетичні алгоритми добре справляються з глобальним пошуком оптимальних рішень, що може бути важливим у випадках, коли інші методи можуть застрягати в локальних мінімумах або максимумах;

– адаптивність. Генетичні алгоритми є адаптивними до змінних умов і можуть пристосовуватися до змін у середовищі, що робить їх стійкими до непередбачуваних факторів;

– паралельна обробка. Генетичні алгоритми можуть ефективно використовувати паралельні обчислення, що дозволяє їм прискорювати процес генерації тестів на багатьох рівнях;

– множинні рішення: Генетичні алгоритми здатні обробляти множину рішень одночасно, що може призводити до різноманітності тестових сценаріїв та поліпшення покриття.

Хоча генетичний алгоритм має свої переваги і недоліки але застосування саме даного методу для розробки методу генерації тестів є цілком доцільним, згідно вище описаних завдань і вимог то розробки даного методу та врахування вхідних даних і структури модульних тестів.

2.3 Метрики оцінки результату роботи генерації тестів.

Згідно минулих розділів стосовно де було описано оцінка придатності згенерованого тесту є одним із дуже важливих етапів методу генерації тестів. Метою оцінювання, як і раніше в попередніх розділах було вказано, є визначення коректності згенерованих тестів шляхом застосування метрик. Згідно проведеного аналізу є дві основні метрики

- покриття генерації;
- співставлення відносно початкового набору.

Метрика «Співставлення відносно початкового набору» в контексті тестування вимірює, наскільки згенерований тест відрізняється від структури початкового набору. Ця метрика важлива для визначення ефективності тестових сценаріїв і виявлення різниць між очікуваним та фактичним станом згенерованих даних. Формула для обчислення метрики виглядає наступним чином (2):

$$\text{Співставлення відносно початкового набору} = \left[\frac{\text{Коефіцієнт початкового набору}}{\text{Коефіцієнт згенерованого набору}}\right] \cdot 100\% \qquad (2)$$

Це потрібно для оцінки надійності тестових сценаріїв та виявлення на скільки велика є розбіжність між початковим набором, який приблизно має дорівнювати згенерованим даним. Високий показник співставлення свідчить про те, що згенеровані дані, сприяють успішному результату виконання алгоритму генерації тестів.

Покриття генерації тестування вказує на те, яка частина програмного коду була охоплена (протестована) тестами. Це вимірюється у відсотках і показує, скільки рядків коду або який відсоток функціональності програми було викликано або покрито в процесі виконання тестів. Чим вище покриття коду, тим більше можливостей тестування та виявлення помилок в коді, що засновано на формулі 3.

$$\text{Покриття генерації } (\%) = \left[\frac{\text{Кількість рядків коду з поч. набору}}{\text{Загальні кількість рядків коду згенерованих тестів}}\right] \cdot 100\% \quad (3)$$

Вказана формула визначає відсоток покриття генерації варіантів, які були використані в алгоритмі, у порівнянні з загальною кількістю рядків коду згенерованим алгоритмом кодогенерації. Чим вищий відсоток, тим більше здійснено покриття генерації.

2.4 Висновки

У даному розділі кваліфікаційної роботи здійснено аналіз моделей та методів кодогенерації для проєктування відповідно самого методу генерації тестів. На основі досліджень було визначено 4 основних алгоритми, за допомогою яких можна здійснити реалізацію даного методу:

- символічне виконання;
- модельна перевірка;
- генетичне програмування;
- використання машинного навчання.

Визначені їхні недоліки та переваги та інші аспекти і на основі даного аналізу для реалізації алгоритму генерації тестів було обрано генетичний алгоритм як найбільш збалансований за перевагами та недоліками з-поміж інших алгоритмів. Окрім цього, було спроектовано модель методу генерації тестів з пошуком дій, яка складається з 5 модулів, що взаємодіють між собою:

– вхідні дані;

– модуль формування початкових даних;

– алгоритм кодогенерації;

– перевірка придатності згенерованих тестів;

– вивід результатів.

Наступним етапом роботи є аналіз детальних вимог до програмного засобу, його проєктування та декомпозиція, підбір та обґрунтування засобів реалізації.

3 ТЕХНОЛОГІЯ РЕАЛІЗАЦІЇ МЕТОДУ ГЕНЕРАЦІЇ ТЕСТІВ З ПОШУКОМ ПЕВНИХ ДІЙ

3.1 Аналіз вимог до програмної системи.

На основі аналізу предметної області, розробленого методу та алгоритмів, необхідно сформувати та описати вимоги до створюваного даного ПЗ, метою розробки якого є створення методу генерацій тестів з пошуком дій. Оновними завдання даного алгоритму:

- формування вхідних даних які задає кінцевий користувач;
- забезпечення генерації тестів на основі вхідних даних;
- визначення придатність згенерованих тестів;
- вивід всіх результатів в інтерфейс користувача.

Класи користувачів та їх характеристики.

Дана ПЗ матиме лише один клас користувачів. Користувачі можуть вводити дані в програмну систему. Далі сама система на основі ж цих даних запускає алгоритм генерації тестів який автоматично формує результат генерації тестів і відповідно видає результати екран який відповідно бачить кінцевий користувач.

Середовище функціонування.

Середовищем функціонуванням є набір двох ПС, які об’єднані на основі клієнт-серверної архітектури. Клієнт-серверна архітектура - це структура, що визначає взаємодію між клієнтськими та серверними компонентами в розподіленому середовищі. Основні риси цієї архітектури:

а) клієнтська частина:

1) користувацький інтерфейс: Клієнт відповідає за надання користувачеві графічного інтерфейсу або інших засобів взаємодії;

2) обробка даних: Обробка частини логіки на клієнтському боці для полегшення взаємодії та зменшення навантаження на сервері.

б) серверна частина:

1) бізнес-логіка: Містить бізнес-логіку та основний функціонал додатка;

2) зберігання даних: Відповідає за збереження та управління даними;

3) обробка запитів: Приймає та обробляє запити від клієнтів, надсилає їм відповіді.

в) комунікація:

1) протоколи: Використовується стандартні протоколи (наприклад, HTTP, HTTPS) для обміну даними між клієнтом та сервером. В даному випадку не важливий вибір протоколу адже засотувати можна будь-який з них;

2) асинхронність: Клієнт та сервер можуть працювати асинхронно, здійснюючи обмін даними без блокування обох сторін (рисунок 3.1).

Рисунок 3.1 Схема архітектури «клієнт-сервер»

3) характеристики методу генерації

В минулих розділах (Розділ 2) було спроєктовано модель самого методу генерації тестів який складається з 5 основних модулів:

- вхідні дані;
- модуль формування початкових даних;
- алгоритм кодогенерації;
- перевірка придатності згенерованих тестів;
- вивід результатів.

Характеристики даних модулів є наступні:

а) вхідні дані. Користувач за допомогою зручного інтерфейсу вводить необхідні початкові дані для початку роботи з методом. В даному випадку має

бути прописана відповідна валідація до форми та встановлені граничні значення з метою уникнення введення помилкових значень які призведуть до поломки ПС;

б) модуль формування початкових даних. Після введення вхідним даних користувачем, дані надходять в даний модуль який має за мету адаптувати їх для алгоритму кодогенрації та визначити точні початкові значення які відправляться до наступного модулю;

в) алгоритм кодогенерації. Даний модуль має забезпечити процес кодогенерації на основі згенерованих даних користувачем та МФПД;

г) придатність згенерованих значень. Основна задача даного модулю це визначення якості та придатності готових результатів алгоритму та здійснити вибірку найбільш доцільних результатів та передати їх на клієнтську частину ПС;

д) вивід результатів. Даний модуль відповідно має забезпечувати вивід результатів в інтерфейс користувача.

3.2 Проєктування програмної системи

Проєктування є одним із важливих етапів розробки ПС. Завдяки вдалому та грамотному проєктуванні залежить 50% всієї успішного роботи.

Розроблювана програмна система, згідно описаних вимог попереднього розділу, має бути повноцінним програмним засобом, котрий складається з двох окремих частин:

– серверна частина;

– клієнтська частина;

В свою чергу, серверну частину ПС можна поділити на три додаткові модулі.

– серверна частина для опрацювання запитів;

– серверна частина для опрацювання алгоритму;

– модуль роботи з базою даних.

Модуль інтерфейсу користувача відповідає за отримання та відображення даних, отриманих від нижчих рівнів архітектури. Також, інтерфейс користувача виконує роль посередника між користувачем та всіма іншими модулями, оскільки саме цей модуль може передавати значення, які обирав, вводив чи змінював користувач у різні поля, перемикачі та інші елементи інтерфейсу. На рисунку 3.2 зображено загальний вигляд взаємодії компонентів програмної системи. Схему взаємодії компонентів ПС наведено на рисунку А.1 (додаток А).

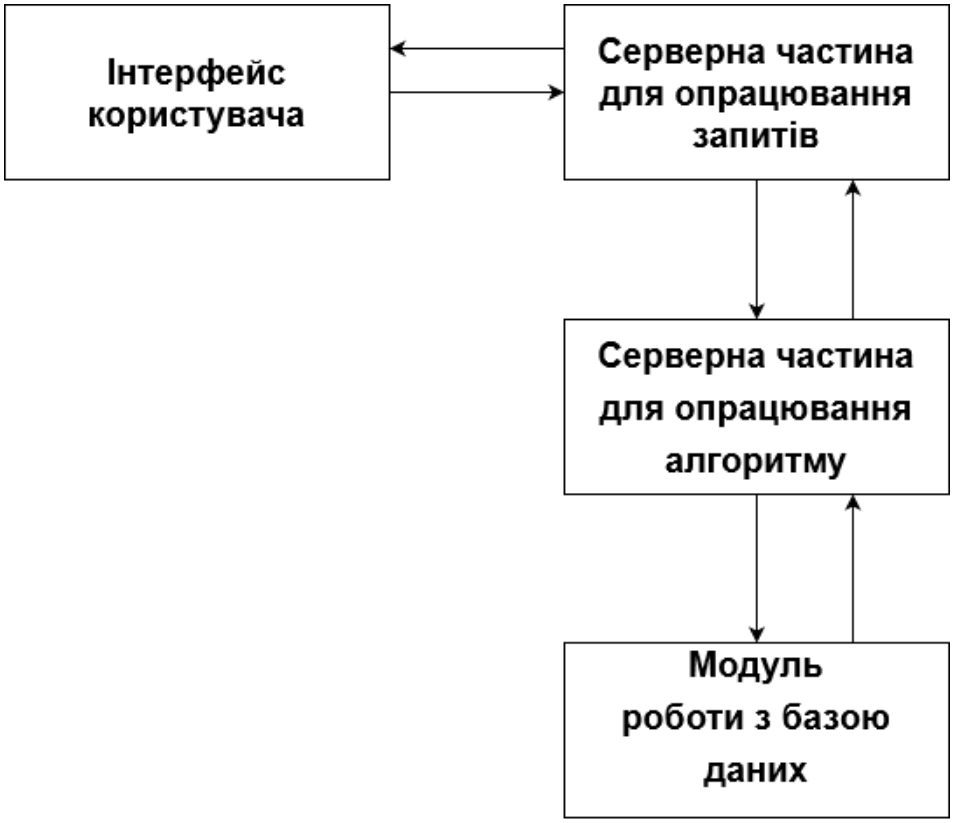


Рисунок 3.2 – Схема взаємодії компонентів ПС

Розглянемо модулі та функції клієнтського додатку. Модуль інтерфейсу користувача має наступні під модулі:

- модуль відображення даних;
- модуль введення даних.

Модуль відображення даних відповідає за отримання та відображення даних з нижніх рівнів архітектури користувачу, а також за взаємодію з користувачем, шляхом отримання введених даних користувачем чи виклику різного функціоналу системи шляхом взаємодії з різноманітними елементами інтерфейсу.

Модуль введення даних застосовується відповідно для введення та передачі даних відповідно на нижні рівні архітектури системи з метою здійснювати керування ПС

Модуль обміну даними з БД відповідає за перетворення та оперування даними, які записуватимуться чи діставатимуться безпосередньо з бази даних, зокрема за коректне створення, зміну даних, а також їх видалення

Модуль взаємодії з інтерфейсом користувачем відповідає за транспортування та налаштування передачі даних між модулем інтерфейсу користувача та системою через сервер, а також за очікування та отримання даних від сервера.

Модуль формування вхідних даних для алгоритму це модуль в серверної архітектури відповідає за формування початкових для проведення генерації тестів, на основі вхідних даних користувача.

Модуль кодогенерації відповідає за обробку кодогенерацію модульних тестів на основі сформованих даних

Модуль придатності відповідає за оцінку та вибірку згенерованих придатніших тестів які згодом виводяться в інтерфейс користувача. Схему взаємодії елементів ПС наведено на рисунку А.2 (додаток А).

Взаємодію компонентів та модулів клієнтського додатку зображено на рисунку 3.3.

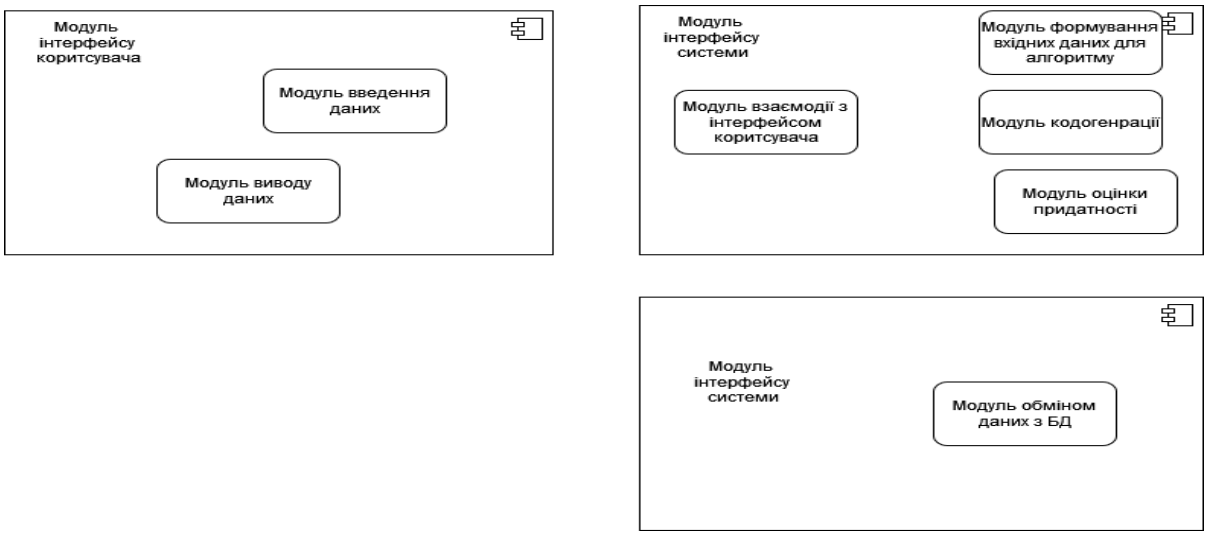


Рисунок 3.3 – Схема взаємодії елементів ПС

Що до архітектури ПС яка зображена на рисунку 3.4 є три основні компоненти:

– інтерфейс користувача;

– REST API;

– сервер.

Інтерфейс користувача подає «post» запит до серверу через «REST API» з даними для обробки і сервер в свою чергу приймає дані, обробляє і відправляє відповідь користувачу.

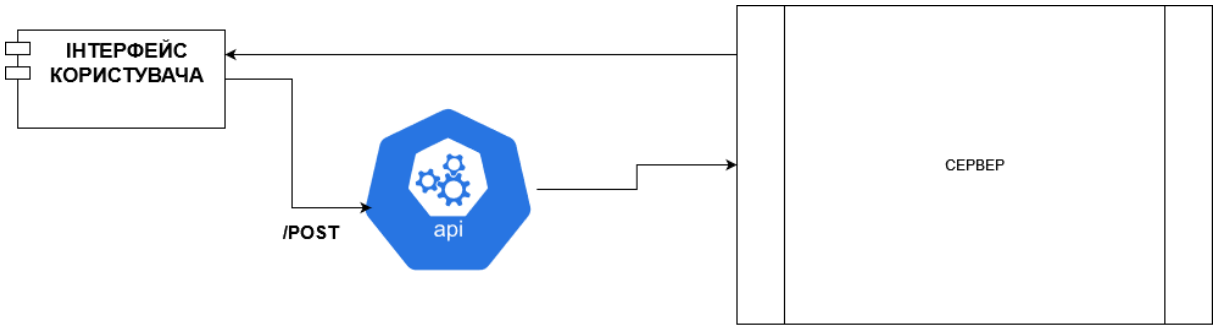


Рисунок 3.4 – Схема архітектури системи

3.3 Опис цілей тестування та вибір алгоритмів.

На основі досліджень минулих розділів даної кваліфікаційної роботи будо досліджено безліч факторів і параметрів які необхідно визначити для початку

проєктування даного методу. В минулих розділах було визначено два основних фактори такі як:

– вид тестування який має генеруватись;

– алгоритм який здійснюватиме безпосередньо кодогенерацію тестів.

Видом тестування для генерації було обрано саме модульне тестування не просто так адже згідно теми кваліфікаційної роботи мова йде про кодогенерацію.

Щодо алгоритму кодогенерації було обрано алгоритм генетичного програмування.

Останнім фактором для забезпечення проєктування даного методу є вибір цілей тестування. Вибір цілей тестування є критичним етапом у процесі розробки програмного забезпечення і визначає, які аспекти системи будуть піддані перевірці та тестуванню. Основні причини чому важливо обирати цілі тестування:

а) забезпечення якості продукту:

1) покриття функціоналу. Визначення конкретних цілей тестування дозволяє забезпечити покриття всіх аспектів функціоналу системи;

2) виявлення дефектів. Тестування націлене на виявлення помилок і недоліків, допомагаючи створити стійкий та надійний продукт.

3) ефективне використання ресурсів, оптимізація витрат. Визначення конкретних цілей допомагає уникнути непотрібного витрачання часу та ресурсів на непов'язані аспекти розробки;

4) адаптація до потреб користувача. Зорієнтованість на користувача. Обрані цілі тестування повинні відповідати вимогам та очікуванням користувачів, забезпечуючи задоволення їх потреб;

5) відповідність вимогам, перевірка відповідності. Цілі тестування повинні бути визначені з урахуванням вимог до системи, щоб перевірити, чи задовольняє програмне забезпечення ці вимоги;

6) мінімізація ризиків, виявлення ризиків. Визначення конкретних цілей тестування дозволяє акцентувати увагу на важливих для вас аспектах, що може допомогти виявити та вирішити потенційні ризики.

Загалом, правильно обрані цілі тестування визначають фокус та стратегію тестування, що сприяє створенню продукту високої якості, відповідного вимогам і очікуванням користувачів також самий основний момент – в залежності від вибору цілей тестування залежить й реалізація алгоритмів кодогенерації адже кожна вид стратегії тестування впливає на програмну реалізацію алгоритму. Тому для початку було обрано дві основні цілі тестування:

– тестування веб-форм;

– тестування запитів.

3.4 Формування вхідних даних

В контексті генерації тестів вхідні дані - це інформація, яка надходить у систему або алгоритм для обробки чи використання. Вхідні дані є основою для проведення тестів, оскільки тестові випадки створюються для перевірки, як система реагує на різні комбінації цих даних. Саме завдяки вхідним даним користувач має змогу здійснювати керування даного методу.

Для реалізації методу генерації тестів можуть бути використані різноманітні вхідні дані. В загальному основні з них включають:

а) модель програми або системи. Опис внутрішньої логіки, структури та функціональності програми, що тестується;

б) специфікації та вимоги. Визначення очікуваної поведінки програми, що може використовуватися для створення тестів, які перевіряють відповідність вимогам;

в) тестові кейси та сценарії. Наявні тестові сценарії та кейси, які можуть бути використані для аналізу та автоматичної генерації нових тестів;

г) дані тестування. Початкові дані для тестових випадків, які можуть включати у себе різноманітні комбінації вхідних параметрів та умов;

д) метрики якості. Визначення критеріїв, за якими можна оцінювати якість тестових наборів;

е) критерії покриття. Інформація про те, які частини коду або функціональності мають бути покриті тестами.

Для початку проєктування методу генерацій тестів вхідні дані було поділено на дві категорії відповідно до об’єктів методу:

– класифікація вхідних даних для кінцевого користувача;

– класифікація вхідних даних для алгоритму кодогенерації.

Класифікація вхідних даних для кінцевого користувача методу генерації тестів може включати різні категорії, які визначають параметри і вхідні умови для алгоритму генерації тестів. Основні категорії вхідних даних для кінцевого включають чотирьох основних параметрів:

– вибір моделі тестування;

– вибір цілі тестування;

– кількість покриття тестування(кількість ітерацій);

– кількість ітерацій.

В таблиці 3.1 продемонстровані основні властивості даних параметрів.

Таблиця 3.1

Назва параметру	Тип даних	Правила валідації
Ціль тестування	Символьна. Вибірка із доступних варіантів	Обов’язкове поле для введення. Лише символи
Модель	Об’єкт класу. Вибірка із доступних варіантів	Обов’язкове поле для введення. Лише символи
Критерій покриття генерації	Числовий.	Обов’язкове поле для введення. Лише числові дані в діапазоні від 0 до 100.
Кількість ітерацій	Числовий.	Обов’язкове поле для введення. Лише числові дані в діапазоні від 10 до 1000.

Цілі тестування це параметр вхідних даних методу за яким визначається область тестування для алгоритму кодогенерації. Ціль тестування визначає, що саме перевіряється, які аспекти програми чи системи важливі для оцінки, і щоб слід отримати в результаті тестового процесу. Даний параметр, згідно загальних вимог яку були описані розділом вище, приймає два значення:

– тестування веб-форм;

– тестування запитів.

Ціль тестування є важливим компонентом тестового процесу, оскільки вона визначає, чого саме очікується від тестування та яким чином має бути оцінено якість продукту. Даний параметр є обов’язковим для введення користувачем та приймає лише символьні дані.

Модель. Параметр який вказує дані моделі для якої має здійснюватися кодогенерація згідно параметрам моделі. Вибірка даних йде згідно спроектованого ПЗ де відповідно було прописані дані моделей як об’єкт класу. Даний параметр є обов’язковим для введення користувачем та приймається лише як об’єкт класу.

Критерій покриття генерації. Даний параметр представляє собою метрику визначення якості генерації тестування. Даний параметр має приймає лише числові дані, є обов’язковим для введення користувачем та приймає значення в діапазоні між 0 до 100 де 0 це мінімальний параметр якості згенерованих тестів буде більш не задовільним але процес генерації буде швидким; значення 100 відповідно вказує максимальний параметр якості згенерованих тестів буде високою але швидкість виконання алгоритму буде повільнішою.

Кількість ітерацій. Згідно спроєктованої моделі методу генерації тестів відповідно до якої буде застосовуватись генетичний алгоритм. Кількість ітерацій у генетичному алгоритмі є параметром, який визначається користувачем, і це один з факторів, які впливають на ефективність алгоритму. Кількість ітерацій визначає, скільки разів повторюється основний цикл алгоритму, де популяція еволюціонує шляхом хромосом. Оптимальна кількість ітерацій залежить від

конкретної задачі, розміру популяції, складності фітнес-функції та інших факторів. Зазвичай визначення оптимальної кількості ітерацій є експериментальним процесом. Даний параметр має приймає лише числові дані, є обов’язковим для введення користувачем та приймає значення в діапазоні між 10 до 1000 де 10 мінімальний параметр значення кількості ітерацій алгоритму за яким відповідно швидкість генерацій буде високою але якість та кількість згенерованих тестів буде низькою; 1000 параметр значення кількості ітерацій за яким також можлива генерація тестів які не будуть відповідати вимогам якості та ще й з дуже низькою швидкістю роботи алгоритму.

Вхідні дані для алгоритму кодогенерації це є початковий набір популяції. Початковий набір даних, у генетичному алгоритмі який здійснює кодогенерацію, включає в себе початкову популяцію. Популяція – це множина особин (індивідів або хромосом), які представляють можливі рішення задачі. Кожна особина в популяції визначається своєю хромосомою, яка містить гени або параметри. Про набір популяцій буде більше описано в наступних розділах даної кваліфікаційної роботи.

3.5 Проєктування бази даних

Щоб забезпечити ефективність і надійність роботи методу, використання бази даних виявляється невід'ємним елементом в реалізації алгоритмів генерації тестів. Це дозволяє зберігати, управляти та оптимізовувати тестові дані та результати.

База даних є віртуальною сховищем для тестового фреймворку, який використовує алгоритми генерації тестів. Вона дозволяє динамічно змінювати та підтримувати тестові дані, забезпечуючи гнучкість і високий рівень автоматизації. використання бази даних стає ключовим елементом для підвищення ефективності генерації тестів та впровадження автоматизованого тестування.

База даних може бути використана для методу генерації тестів наступним чином:

а) зберігання тестових даних;

1) база даних може використовуватись для зберігання тестових даних, таких як вхідні параметри та очікувані результати тестів;

2) різні таблиці можуть відображати різні сценарії тестування, а рядки можуть представляти конкретні тестові випадки.

б) генерація тестових даних:

1) можна використовувати SQL-запити для генерації тестових даних, що відповідає різним сценаріям тестування;

2) використання бази даних для динамічної генерації тестових даних може полегшити роботу з різними варіантами вхідних даних.

в) завантаження тестових результатів:

1) результати виконаних тестів можна зберігати в базі даних для подальшого аналізу;

2) це дозволяє стежити за результатами тестів, визначати, які тести пройшли або виявили проблеми.

г) управління конфігурацією. Використання бази даних для зберігання конфігураційних параметрів тестових середовищ може забезпечити зручний доступ та маніпулювання налаштуваннями тестів;

д) відстеження змін. ведення журналу змін тестових сценаріїв у базі даних може бути корисним для відстеження розвитку тестових наборів;

Загалом, використання бази даних для методу генерації тестів може сприяти ефективнішому та систематизованому підходу до управління тестовими данними та тестовими сценаріями.

На основі визначених вимог до ПС було спроєктовану базу даних яка складається з наступних таблиць:

– набір вхідних даних;

– головна таблиця для генерації;

– набір базових тестів;

– історія згенерованих тестів;

– ітерація тесту.

Набір вхідних даних. (Рисунок 3.5) Дана таблиця застосовується для збереження вхідних даних які вносив користувач через інтерфейс. На основі цієї таблиці алгоритм використовує дані відповідно для початку генерації тестів.

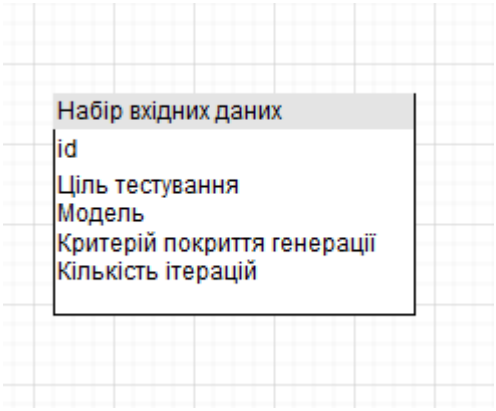


Рисунок 3.5 – Набір вхідних даних

Наступна таблиця (Рисунок 3.6) – це головна таблиця генерація тестів. Основне завдання даної таблиці це зберігання загальної оцінки та використовується як посилання згенерованих тестів по яким можна вивести готові згенеровані тести.

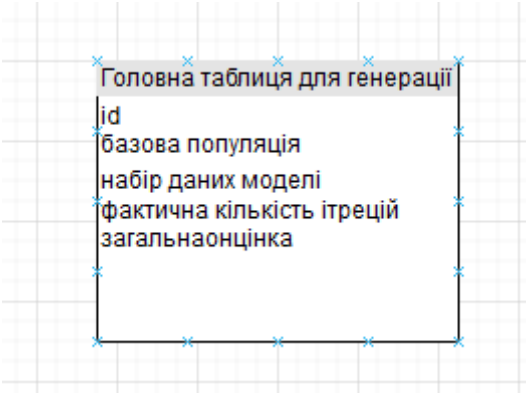


Рисунок 3.6 – Головна таблиця для генерації

Таблиця ітерації тесту представлена на рисунку 3.7. Дана таблиця застосовується для запису згенерованого тесту. По даній таблиці визначається

придатність згенерованих тестів; Придатні тести зберігаються для виводу в інтерфейс користувача а негативні тести відсіюються.

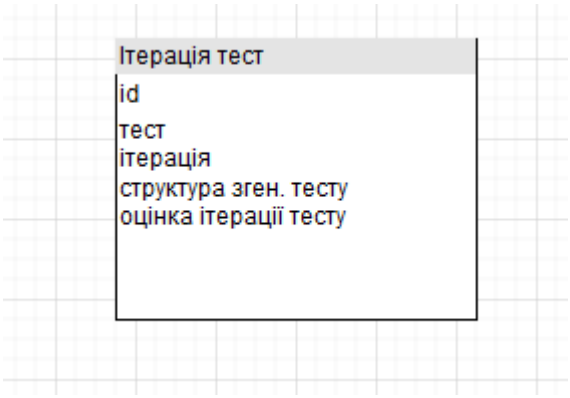


Рисунок 3.7 – Таблиця ітерацію тесту

Наступна таблиця (Рисунок 3.8) – це таблиця базових тестів. Основна задача даної таблиці це зберігання даних для початкових популяцій генетичного алгоритму. Саме від цих даних генетичний алгоритм на основі аналізу генеруює схожі тестові сценарії змінюю структуру тесту.

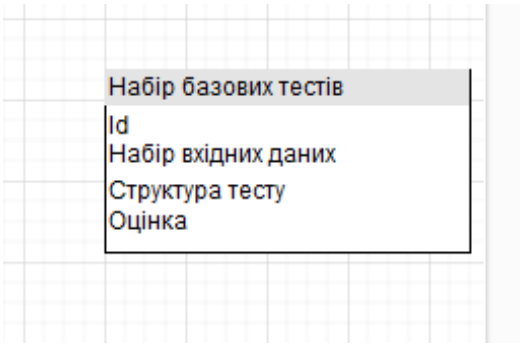


Рисунок 3.8 – Таблиця базових тестів.

Остання таблиця (рисунок 3.9) – це таблиця історій згенерованих тестів. Основна функція даної таблиці це зберігання всіх згенерованих тестів з метою в подальшому проведення аналізу роботи методу. Дана таблиця створена виключно суто з метою дослідження.

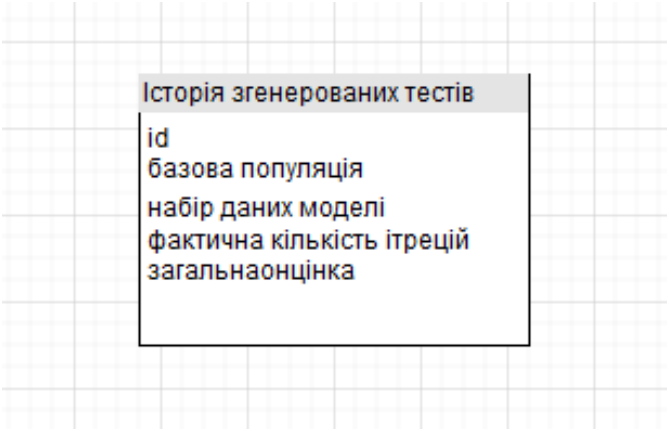


Рисунок 3.9 – Таблиця Історії згенерованих тестів

Основна схема БД продемонстрована на рисунку 3.10. Впровадження бази даних у контексті алгоритмів генерації тестів дозволяє ефективно управляти початковим набором даних, динамічно змінювати тестові сценарії та оптимізувати результати тестування. Це сприяє покращенню автоматизованого тестування та впровадженню високоефективних стратегій тестування.

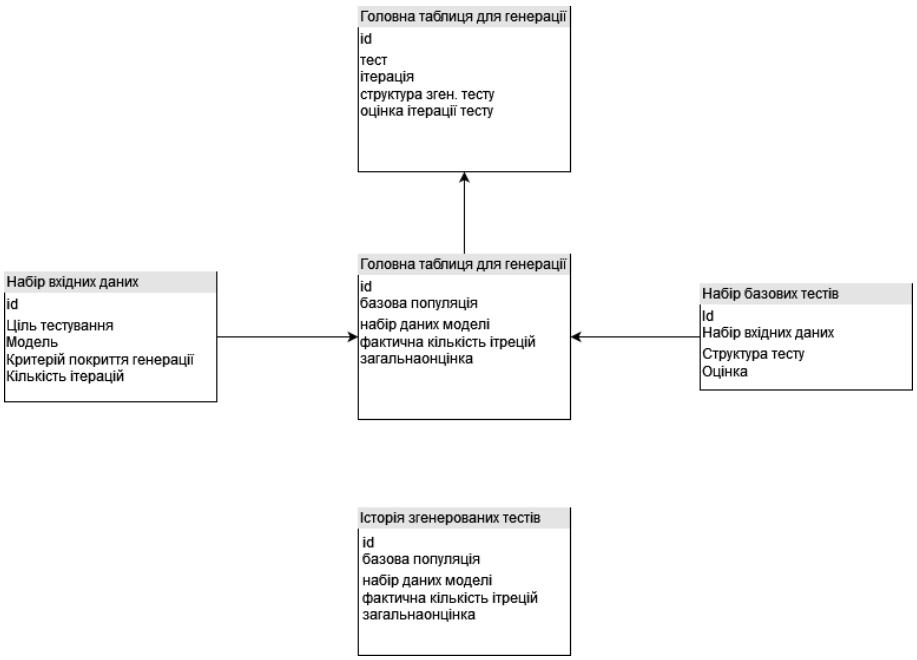


Рисунок 3.10 – Інфологічна схема бази даних

3.6 Застосування генетичного алгоритму

На основі проведеного аналізу в розділі 2 було проведено дослідження алгоритмів які потенційно здатні забезпечити кодогенерацію тестів. Із всіх запропонованих алгоритмів було обрано саме генетичного алгоритм.

Генетичний алгоритм – це ефективний метод оптимізації та пошуку, що інспірується природним відбором та генетичними процесами. Цей алгоритм базується на ідеї еволюції, де покоління популяції проходять через серію мутацій, схрещувань та відбору для досягнення оптимальних рішень.

Процес генетичного алгоритму можна уявити як еволюцію популяції, де кожна «особина» представляє потенційне рішення. Мутації та схрещування забезпечують створення нових рішень, а відбір визначає, які рішення матимуть більші шанси передачі своїх характеристик на наступне покоління.

а) глобальний пошук. Генетичні алгоритми придатні для глобального пошуку в просторі можливих рішень. Вони допомагають уникнути провисання в локальних екстремумах, яке характерне для деяких інших методів оптимізації;

б) паралельна обробка. Часто генетичні алгоритми дозволяють легко реалізувати паралельні версії. Це забезпечує прискорення обчислень та покращує продуктивність, особливо для задач, що вимагають великої обчислювальної потужності;

в) гнучкість та універсальність. Генетичні алгоритми можна легко адаптувати до різноманітних типів задач, включаючи оптимізацію параметрів, розкладання задачі, а також генерацію тестів;

г) ефективність при великій кількості параметрів. Для задач з великою кількістю параметрів генетичні алгоритми можуть бути ефективні, оскільки вони взаємодіють з великою кількістю можливих конфігурацій;

д) застосування в задачах оптимізації та навчання: Генетичні алгоритми успішно використовуються для вирішення проблем оптимізації та машинного

навчання, зокрема для підбору параметрів моделей та генерації призначених для навчання наборів даних в тому числі і кодогенерації.

Основними факторами які пливають на проєктування генетичного алгоритму є наступні:

– врахування структури коду модульних тестів;

– згенерувати початкову популяції модульних тестів;

– розробка модулю формування початкових даних.

Врахування структури коду модульних тестів необхідно для того щоб правильно навчити генетичний алгоритм розпізнавати, класифікувати та генерувати тести відповідно до особливостей та логіки вхідних даних.

Генерування початкових популяцій модульних тестів. Генетичний алгоритм, згідно проведеного аналізу у розділі №2, працює шляхом аналізу початкових даних популяцій від яких він «відштовхується» генеруючи нові дані. Таким чином виникає потреба генерування базових «прикладів» модульних тестів а також класифікувати популяцію.

Модулю формування початкових даних необхідний для того щоб формувати власні вхідні дані які буде «розуміти» генетичний алгоритм. Практично цей модуль є перекладачем між користувачем та генетичною «машиною». Окрім цього даний на основі даних які подає користувач, модуль має на меті обирати відповідно популяцію по категорію. Інтеграція даного модулю в даний метод алгоритм дозволяє ефективніше враховувати контекст та завдання тестування, роблячи його більш адаптованим до конкретних вимог користувача.

Після визначення факторів і вимог проєктування - наступним етапом даного розділу буде проєктування окремих модулів генетичного алгоритму кожен з яких відповідає за конкретний аспект генерації тестів. Ключові модулі включатимуть у себе механізми відбору, схрещування та мутації тестових кейсів, що дозволить забезпечити високу якість та маневреність генераційного процесу. Додаткові модулі будуть відповідати за збір та аналіз результатів тестування, а також за забезпечення ефективного використання вхідних даних та ресурсів для

оптимізації продуктивності генетичного алгоритму. Впровадження цих модулів сприятиме реалізації ефективного та надійного методу генерації тестів для автоматизованого тестування програмного забезпечення.

3.6.1 Початковий набір даних, популяція

Початкова популяція в генетичному алгоритмі – це початковий набір індивідів або хромосом, які представляють можливі рішення проблеми, що вирішується. Кожен індивід у популяції представляє собою потенційний варіант розв'язку. Ці індивіди формуються випадковим чином на початку процесу генетичного алгоритму або використовуються заздалегідь визначені вхідні дані (Рисунок 3.11).

Початкова популяція визначає стартовий пул можливих розв'язків, які потім еволюціонують за допомогою відбору, схрещування і мутації, наближаючи генетичний алгоритм до оптимального або прийнятного розв'язку завданої задачі.

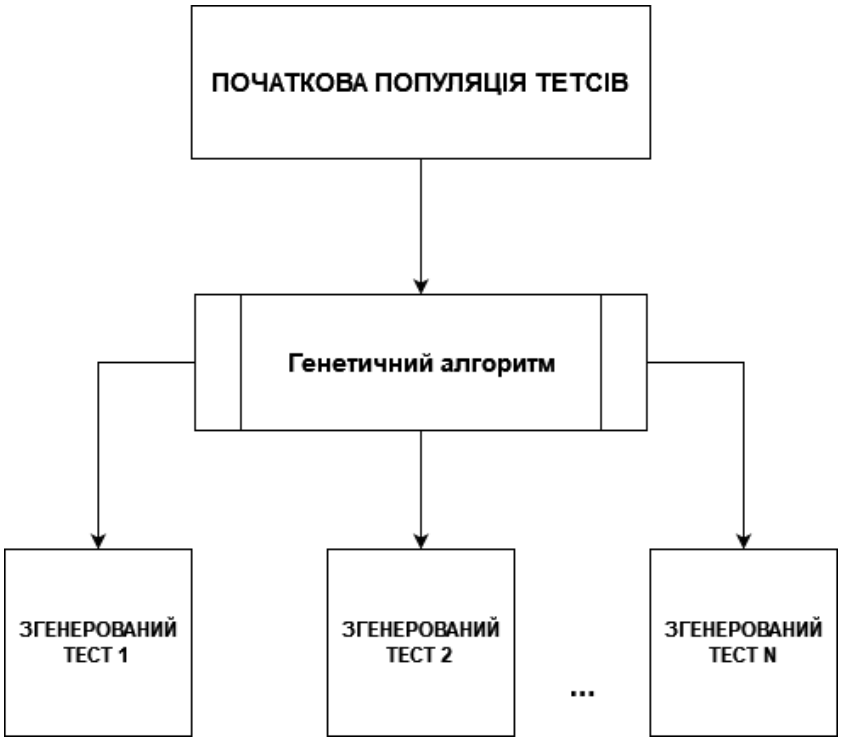


Рисунок 3.11 – Основний принцип популяції для генетичного алгоритму

Для початку формуванню популяцій, необхідно визначити категорії на які будуть поділятись індивіду популяцій. Виділено наступні основні категорії:

– класифікація за моделями;

– класифікація за областю тестування.

Безумовно, що кожна категорія має свої властивості. Наприклад класифікація за моделями враховуються які моделі мають застосовуватись, які в них є параметри. Наприклад є модель «User»(Користувач). Користувач може мати такі параметри як «name», «e-mail», «age» тощо. Тобто потрібно чим більше визначити які є моделі та які в них можуть бути параметри і значення параметрів. Чи більше таких параметрів буде враховано в популяцію тим більший шанс забезпечити якість роботи методу генерації.

Класифікація за областю тестування. Тут все значно простіше адже згідно вимог до методу необхідно реалізувати кодогенерацію для двох цілей тестування це вебформи так запити.

Після визначення параметрів класифікації можна приступати до етапу формування популяції. Для того щоб згенерувати популяції необхідно обрати стратегію формування. Стратегії формування популяції в генетичних алгоритмах визначають, як початковий набір індивідів генерується перед початком еволюційного процесу. Ось кілька загальних стратегій:

а) випадкова генерація. Індивіди створюються випадковим чином, без урахування жодних особливостей задачі. Це дозволяє розпочати еволюційний процес без будь-якої апріорної інформації.

б) генерація на основі експертних знань. Індивіди формуються на основі експертних знань про задачу. Наприклад, якщо відомо, що певні характеристики є важливими, можна створити початковий набір, який враховує ці особливості.

в) ініціалізація на основі існуючих рішень. Якщо є попередні рішення задачі, можна використовувати їх як основу для створення початкової популяції. Це особливо корисно, коли є знання про придатність конкретних рішень.

г) генерація на основі структури. Початкові індивіди формуються з урахуванням певної структури задачі або представлення даних. Це може включати в себе використання особливих правил чи шаблонів.

д) генерація на основі попередніх розв'язків. Індивіди можуть бути створені за допомогою комбінації частин попередніх розв'язків. Це включає в себе використання операцій кросоверу або інших методів об'єднання різних розв'язків.

е) покращення популяції. Іноді початкова популяція генерується випадковим чином, а потім застосовуються покращення або локальні оптимізації для покращення їхньої якості перед початком еволюційного процесу.

Вибір конкретної стратегії формування популяції залежить від природи задачі та вимог до еволюційного процесу. Але в даному випадку можна застосувати будь-яку стратегія з вище перерахованих для генерації популяції тестів. На рисунку 3.12 продемонстровано приклад формування початкової популяції з урахування поділу їх на категорії.

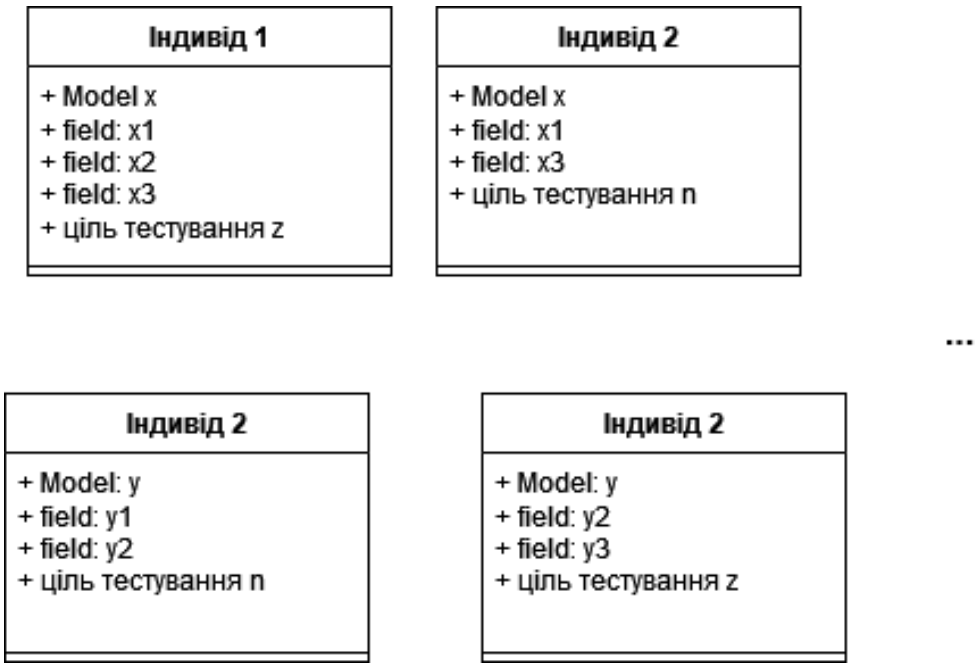


Рисунок 3.12 – Приклад генерації індивідів тестів для початкової популяції

В результаті було сформовано близько 250 індивідів які внесені в базу які представляють початкову популяцію для подальшого еволюційного процесу. Кожен індивід має свої характеристики та генетичну інформацію, яка визначає

його потенційну придатність для вирішення поставленої задачі. Цей набір індивідів є основою для подальших операцій кросоверу, мутації та відбору, спрямованих на покращення якості рішення у кожному поколінні.

3.6.2 Модулю формування початкових даних для генетичного алгоритму.

Після завершення етапу створення початкових популяцій індивідів необхідно створити додатковий модуль для формування вхідних даних для генетичного алгоритму. Даний модуль, як це було описано в попередніх розділах, необхідний для створення умовної «адаптація» між вхідними даними користувача та самим генетичним алгоритмом. Тобто основна задача даного модуля це формування масиву даних вхідні дані від користувача до них додається вибірка індивідів згідно категорій. Таким чином даний модуль працює як умовний перекладати між користувачем та генетичним алгоритмом (рисунок 3.13).

Рисунок 3.13 – Схема здійснення адаптації даних для ГА

Для реалізації вибірки індивідів необхідно обрати потрібну стратегію відбору. Стратегія відбору у генетичному алгоритмі визначає, які індивіди з поточної популяції будуть вибрані для участі в створенні нащадків, які будуть утворювати нову популяцію. Це ключовий компонент еволюційного процесу, оскільки відбір визначає, які гени передаються нащадкам і, таким чином, які рішення можуть бути удосконалені чи викинуті. Деякі загальні стратегії відбору включають:

а) рулетковий відбір. Індивіди обираються на основі їхньої придатності (оцінки цільової функції), причому ймовірність вибору зростає зі збільшенням придатності. Іншими словами, індивіди з більшою придатністю мають більше шансів бути вибраними;

б) турнірний відбір. Випадковим чином вибирається набір індивідів, і серед них обирається найкращий за придатністю. Цей процес повторюється для створення нового покоління;

в) стратегія елітизму. Найкращі індивіди без змін вводяться в нове покоління без внесення змін, що дозволяє зберегти найкращі рішення від покоління до покоління;

г) стратегія відбору за ранжуванням. Індивіди аранжуються за придатністю, і ймовірність їх вибору залежить від їхнього місця в ранжуванні. Це дозволяє виділити найкращих індивідів без значущих величин ймовірностей;

д) стратегія відбору за ранжуванням зі зміненими ймовірностями. Схожа на стратегію відбору за ранжуванням, але ймовірності вибору індивідів можуть бути змінені, щоб надати перевагу менш придатним індивідам;

е) стратегія відбору за параметрами значень. Здійснюється вибірка індивідів на основі збігу певних параметрів.

Вибір конкретної стратегії відбору залежить від характеристик задачі і вимог генетичного алгоритму. Для даного модуля зі всіма вимогами до реалізації методу генерацій тестів найбільш підходящим варіантом є стратегія відбору за параметрами.

Загальний вигляд вхідних даних для генетичного алгоритму після закінчення роботи МФПД виглядатиме згідно таблиці 3.2. Тобто коли даний модуль отримує дані від користувача згідно таблиці 3.1, до цих даних додаються два «json» параметри. У першому параметрі вказано параметри моделі для тестування а в іншому вже згідно стратегії відбору індивідів сформований масив індивідів.

Таблиця 3.2

Назва параметру	Тип даних	Приклад даних
Ціль тестування	Символьна. Вибірка із доступних варіантів	Форма
Модель	Об’єкт класу. Вибірка із доступних варіантів	User
Параметри моделі	Json	«{‘name’ => ‘name’, ‘age=> 2’…}»
Критерій покриття генерації	Числовий.	30
Кількість ітерацій	Числовий.	400
Вибірка з початкових індивідів	Json	«{ ‘індивід 1‘,‘ індивід 2‘,…‘індивід n‘} »

Розробка даного методу є надзвичайно важливим етапом у розробці методу генерацій тестів адже якщо правильно сформувати початкові дані для генетичного алгоритму, який буде здійснювати кодогенерацію, відповідно буде правильно здійснюватися генерація тестів згідно всім вимогам до ПС. Якщо даному етапі будуть помилки відповідно ГА буде видавати хибні дані.

3.6.3 Хромосоми генетичного алгоритму

Хромосоми або індивідуум в генетичному алгоритмі представляють собою набір генів або параметрів, які кодують потенційне рішення задачі. Кожна хромосома є представником індивіда в популяції і визначає його генетичну структуру. Гени, що складають хромосому, можуть визначати різноманітні характеристики, такі як параметри алгоритму, ваги у випадку машинного навчання чи інші важливі аспекти завдання.

Важливість хромосом полягає в тому, що вони служать основним засобом передачі інформації в процесі еволюції. Під час кросоверу чи мутації вони можуть комбінуватися та змінюватися, утворюючи нових індивідів у наступних поколіннях. Цей механізм імітує природний відбір та еволюцію, де кращі адаптації передаються нащадкам.

У контексті конкретних завдань хромосоми можуть мати різні рольові специфікації. У даному випадку задачі генерації тестів для програмного забезпечення, хромосоми можуть мати набори вхідних даних, послідовності дій чи параметри тестових випробувань. На основі цих даних формується відповідний тестовий сценарій. Фактично індивід – це є сформований готовий модульний тест. Їхнє правильне представлення і обробка грає ключову роль у вдалому вирішенні завдання еволюційного генерації оптимальних рішень.

У контексті створення генетичного алгоритму, на основі аналізу в розділі 2, моделі було сформовані наступні параметри для хромосоми або генів хромосом:

- параметр моделі;
- параметр операції;
- параметр очікуваного результату;
- параметр мети тестування.

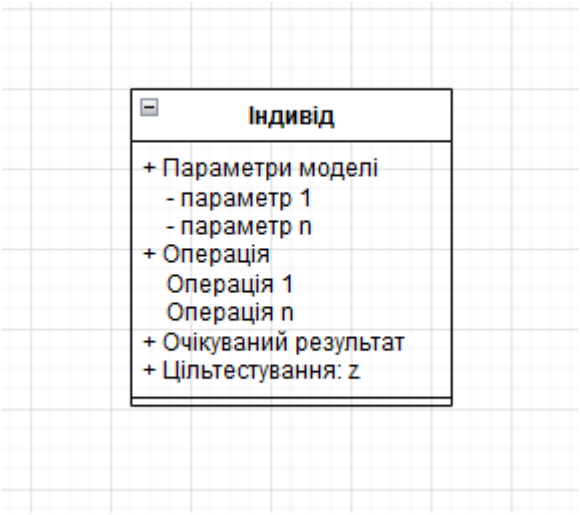


Рисунок 3.14 – Приклад параметрів(генів) хромосом(індивіда)

Параметр моделі – це набір вхідних даних для тестування. Генетичний алгоритм може змінювати дані значення цих параметрів. Наприклад, при тестуванні форми в тестових сценаріях підставляють різні значення параметрів з метою визначення поведінки ПЗ при різних значення даних.

Операція – це набір функцій, які мають тестуватись. Наприклад при тестуванні запитів генетичний алгоритм буде формувати різний набір функцій.

3.6.4 Схрещування та мутація генетичного алгоритму

Схрещування (кросовер) в генетичному алгоритмі – це процес обміну генетичною інформацією між двома батьківськими хромосомами для створення нових хромосом, які стануть нащадками. Цей процес моделює природний механізм рекомбінації ДНК в біології.

Принцип роботи схрещування зазвичай полягає у виборі певного місця (точки схрещування) на батьківських хромосомах і обміні частинами хромосом між цими точками. В результаті формуються нові хромосоми, які успадковують частини генетичної інформації від обох батьків.

У генетичних алгоритмах існує кілька стратегій схрещування, які визначають, як саме відбувається обмін генетичною інформацією між батьківськими хромосомами. Деякі з популярних стратегій включають:

– одноточкове схрещування. Вибирається одна випадкова точка на хромосомі, і гени з одного батька обмінюються з генами іншого батька по одному боці від цієї точки;

– багатоточкове схрещування. Схожий на одноточкове схрещування, але вибирається більше однієї точки обміну, і гени обмінюються між батьками в кількох місцях;

– рівномірне схрещування. Кожен ген обирається для обміну з певною ймовірністю. Це означає, що кожен ген має шанс обмінятися між батьками;

– арифметичне схрещування. Гени хромосоми обмінюються арифметичним чином. Це зазвичай використовується для числових хромосом, де гени представляють числові значення;

– гомологічне схрещування. Гени обмінюються на відстані від точки схрещування, щоб зберегти певний порядок генів в кожній хромосомі.

– вибір конкретного методу схрещування може впливати на швидкість збіжності алгоритму та якість отриманих рішень, і цей вибір часто залежить від природи задачі та структури даних.

Для механізму схрещування було обрано метод багатоточкового схрещування, оскільки цей метод дозволяє вибрати кілька точок обміну генами між батьками тим самим забезпечуючи високу швидкість та продуктивність роботи генетичного алгоритму. В результаті цього утворюється потомство, яке успадковує комбінації генів від обох батьків в кількох місцях.

Багатоточкове схрещування може сприяти збереженню блоків генів, які розташовані поруч один з одним у батьківських хромосомах, забезпечуючи більшу стабільність та сприяючи передачі групи генів від одного покоління до іншого. Це може бути особливо корисним, коли специфічні групи генів взаємодіють або взаємодіють у контексті тестових наборів, забезпечуючи більшу ймовірність виявлення корисних комбінацій для тестування програмного забезпечення.

Якщо коротко описати механізм схрещування в контексті даного методу в даному випадку за допомогою функції схрещування батьківські хромосома передає параметри моделі, операцій та цілей новому поколінню (Рисунок 3.15)

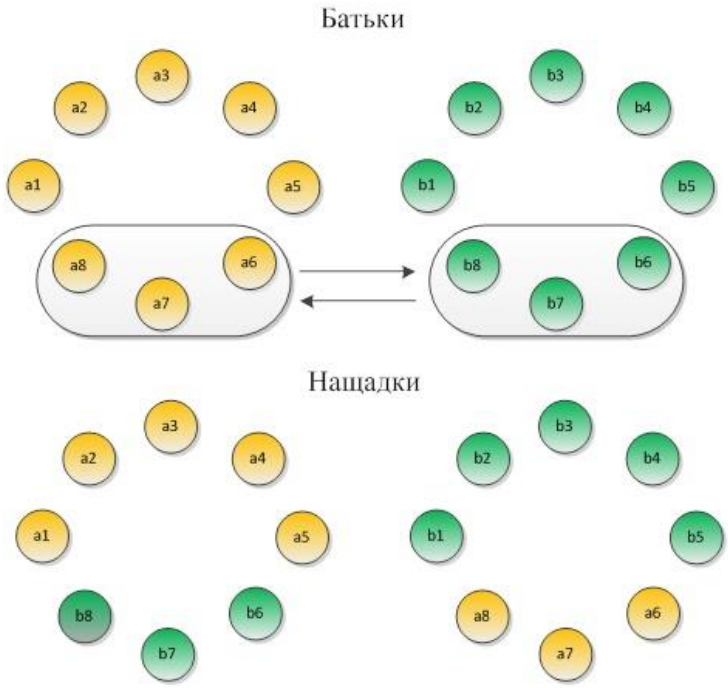


Рисунок 3.15 – Приклад застосування багатоточкове схрещування

Мутація в генетичному алгоритмі – це процес випадкового змінення генетичної інформації в хромосомі, що представляє собою один із тестових наборів. Мутація вводить елемент випадковості в еволюційний процес, допомагаючи виходити за межі поточних рішень та допомагаючи алгоритму досліджувати нові області пошуку.

Процес мутації включає в себе зміну одного або декількох генів (параметрів) в сукупність генів хромосоми. Ця зміна може бути виконана різними способами, такими як:

– зміна значення гену. Випадковим чином обирається нове значення для конкретного гена в хромосомі;

– вставка або видалення гена. Додавання нового гена або видалення існуючого;

– інші операції. Інші техніки мутації, такі як обмін частинами генів, можуть використовуватись залежно від конкретної реалізації генетичного алгоритму.

Мутація сприяє в популяції і може допомогти уникнути зависання в локальних оптимумах, що робить генетичний алгоритм більш гнучким у виявленні оптимальних рішень.

Якщо коротко описати механізм мутація в контексті в контексті методу генерацій тесті в даному випадку це випадкове зміна параметра даних моделі (Рисунок 3.16).

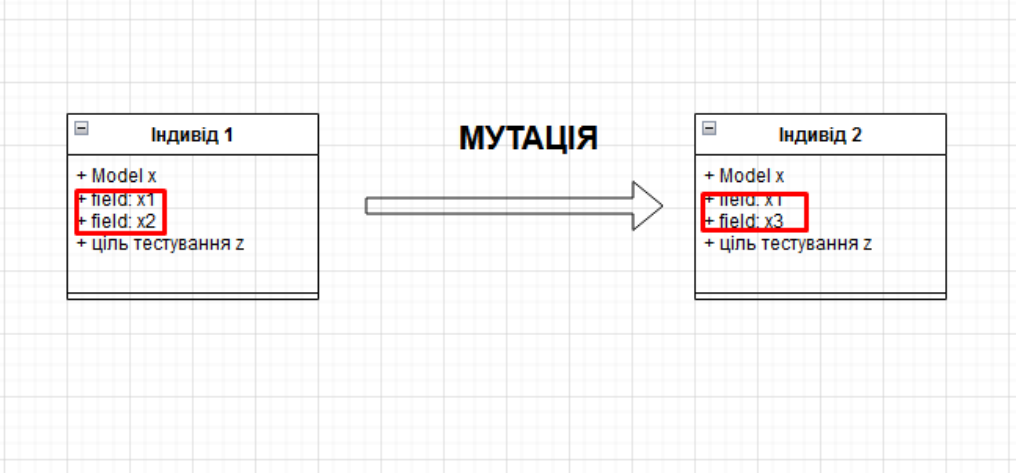


Рисунок 3.16 – Приклад мутації в методі генерації тестів

3.7 Висновок

У даному розділі кваліфікаційної роботи проведено аналіз вимог до програмного засобу та визначено які завдання має виконувати методо генерації тестів:

– формування вхідних даних які задає кінцевий користувач;

– забезпечення генерації тестів на основі вхідних даних;

– визначення придатність згенерованих тестів;

– вивід всіх результатів в інтерфейс користувача.

Важливою частиною розробки даного методу є правильне проєктування та реалізація:

– структурування та класифікація вхідних даних для початкових популяцій;
– проєктування модуля формування початкових даних;
– проєктування генетичного алгоритму та його компонентів.

У розділі також обрана архітектура ПЗ, визначено модулі ПЗ та описано призначення кожного модуля, здійснено їх декомпозицію на компоненти, побудовані діаграми UML (зокрема діаграми компонентів та послідовності).

Наступним етапом було описано цілі тестування для яких даний метод генерації генерує тести з урахування вхідних даних користувача.

Отже, під час написання даного розділу було проведено проєктування ПС на рівні алгоритмів, описано структуру та суть генетичного алгоритму, продемонстрований приклад роботи алгоритму, структури даних та архітектури системи.

Наступним етапом є програмна реалізація спроектованого ПС та його тестування, а також доведення ефективності роботи даного методу.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МЕТОДУ ГЕНЕРАЦІЇ ТЕСТІВ

4.1 Програмна реалізація

Засіб реалізації методу генерації тестів – це програмна частина або інструмент, який використовується для практичної реалізації алгоритмів генерації тестових наборів. Цей засіб може включати в себе програми, бібліотеки, фреймворки або інші компоненти, призначені для автоматизації процесу створення та виконання тестів у розробці програмного забезпечення.

4.1.1 Засоби реалізації

Для реалізації даної програмної систему було застосовані два технічних засобів:

- php фреймворк Laravel;
- мова програмування Python.

Laravel – це безкоштовний фреймворк для розробки веб-застосунків, написаний на мові програмування PHP. Він надає ряд зручних інструментів та сервісів для швидкого і ефективного створення сучасних веб-застосунків. Laravel використовує принципи MVC (Model-View-Controller) для структуризації коду, а його потужний ORM (Object-Relational Mapping) Eloquent дозволяє працювати з базою даних, використовуючи об'єктно-орієнтований підхід. Цей фреймворк надає широкий набір функцій, таких як маршрутизація, керування сеансами, аутентифікація, кешування та інші, що спрощують процес розробки. Laravel також володіє активною спільнотою, великою кількістю розширень та докладною документацією, що робить його популярним вибором для розробників веб-застосунків [14].

Даний програмний засіб був використаний в кваліфікаційній роботі для реалізації наступних завдань:

- побудова інтерфейсу користувача;

– побудова мікро-серверну система для використання REST API;

– керування інтерфейс керування БД;

– перевірка згенерованих тестів інструментами формування модульних тестів від Laravel.

Для того щоб забезпечити мінімальну зручність керування методом генерацій тестів було побудовану елементарний користувацький інтерфейс з полями для вводу даних та виводу результатів методу генерацій тестів.

Для забезпечення передачі даних REST API через між Laravel та Python, завдяки якому реалізовано сам метод генерації(див. розділ 4.1.2), було побудовано мікро-серверну частину ПС. RESTful API – це архітектурний стиль для створення веб-служб, який використовує принципи HTTP-протоколу для обміну даними. Саме через RESTful API вхідні дані від Laravel будуть передаватись в Python а результат згенерованих рішень відповідно будуть йти від Python до Laravel [16].

Дана ПС використовує базу даних MySQL, яка була спроєктована відповідно в розділі 3.5. Laravel має вбудований ORM (Object-Relational Mapping), відомий як Eloquent, дозволяє працювати з базою даних в зручний і об'єктно-орієнтований спосіб.

Laravel надає має потужних інструментів для тестування, які спрощують процес розробки та підтримують високу якість коду – це інструмент PHPUnit Integration. [15].

Python – це високорівнева, інтерпретована, загальнопризначена мова програмування. Вона була розроблена Гвідо ван Россумом і перша версія вийшла у 1991 році. Python призначений для спрощення процесу розробки програм та читання коду, забезпечуючи чистий і лаконічний синтаксис.[17]

Основні особливості Python включають:

а) хитабельність коду. Python відзначається чистим синтаксисом, що дозволяє розробникам висловлювати ідеї меншою кількістю рядків коду порівняно з іншими мовами;

б) інтерпретованість. Python є інтерпретованою мовою, що означає, що можна виконувати код без попередньої компіляції. Це полегшує тестування та відлагодження програм;

в) об'єктно-орієнтована. Python підтримує об'єктно-орієнтоване програмування та дозволяє розробникам використовувати класи та об'єкти в їхній роботі;

г) багатий стандартний бібліотека. Python має велику стандартну бібліотеку, яка включає різноманітні модулі та пакети для вирішення різноманітних задач, таких як робота з мережами, обробка текстів, робота з базами даних і багато іншого;

д) крос-платформеність. Код, написаний на Python, може бути виконаний на різних операційних системах без змін;

е) розширюваність. Python легко інтегрується з іншими мовами програмування, такими як C і C++, що дозволяє розробникам використовувати існуючі бібліотеки і ресурси.

Мова програмування Python виявляється ідеальним інструментом для реалізації методів, що ґрунтуються на принципах машинного навчання. Її зручний та читабельний синтаксис, багата екосистема бібліотек, таких як TensorFlow, PyTorch та scikit-learn, дозволяє легко втілювати складні алгоритми навчання. Тому для реалізації методу генерації тестів зі всіма компонентами методу як модуль МФПД та генетичний алгоритм було застосовано було застомовано бібліотеку DEAP (Distributed Evolutionary Algorithms in Python) [18] .

DEAP є потужним інструментом для створення та експериментування з еволюційними алгоритмами.DEAP включає в себе різноманітні засоби для роботи з генетичними алгоритмами, включаючи:

а) структури Даних Генетичного Алгоритму. DEAP надає готові структури для представлення особин (індивідів), їхніх хромосом і популяцій;

б) оператори схрещування та мутації. Бібліотека містить різноманітні вбудовані оператори схрещування і мутації, які можна легко комбінувати та адаптувати для конкретних задач;

в) модулі для Паралельної Роботи. DEAP підтримує паралельні обчислення, що дозволяє прискорити еволюційний процес за допомогою розподілених обчислень;

г) інструменти для Аналізу та візуалізації. Бібліотека надає засоби для моніторингу та аналізу процесу еволюції, а також інструменти для візуалізації результатів;

д) підтримка різних типів задач. DEAP може використовуватися для вирішення різних задач, від оптимізації параметрів машинного навчання до створення розкладів та інших.

4.1.2 Реалізація програмної системи через Laravel

Метою застосування php фреймворку Laravel в контексті реалізації методу генерації тестів лежить наступне:

- побудова інтерфейсу користувача;
- побудова мікро-серверну система для використання REST API;
- керування інтерфейс керування БД;
- перевірка згенерованих тестів інструментами формування модульних тестів від Laravel.

Самим першим етапом розробки програмної частини на Laravel є побудова інтерфейсу. Інтерфейс забезпечить формування та передачу вхідних даних самому методу Згідно проєктованого плану розробки методу було побудовано форму для введення наступних даних:

- ціль тестування;
- модель тестування;

– критерій покриття генерації;

– кількість ітерацій.

Інтерфейс побудований за допомогою JS бібліотеки Bootstrap. Дана бібліотека допомагає швидко будувати веб-додатку. На рисунку 4.1 зображено інтерфейс ПЗ.

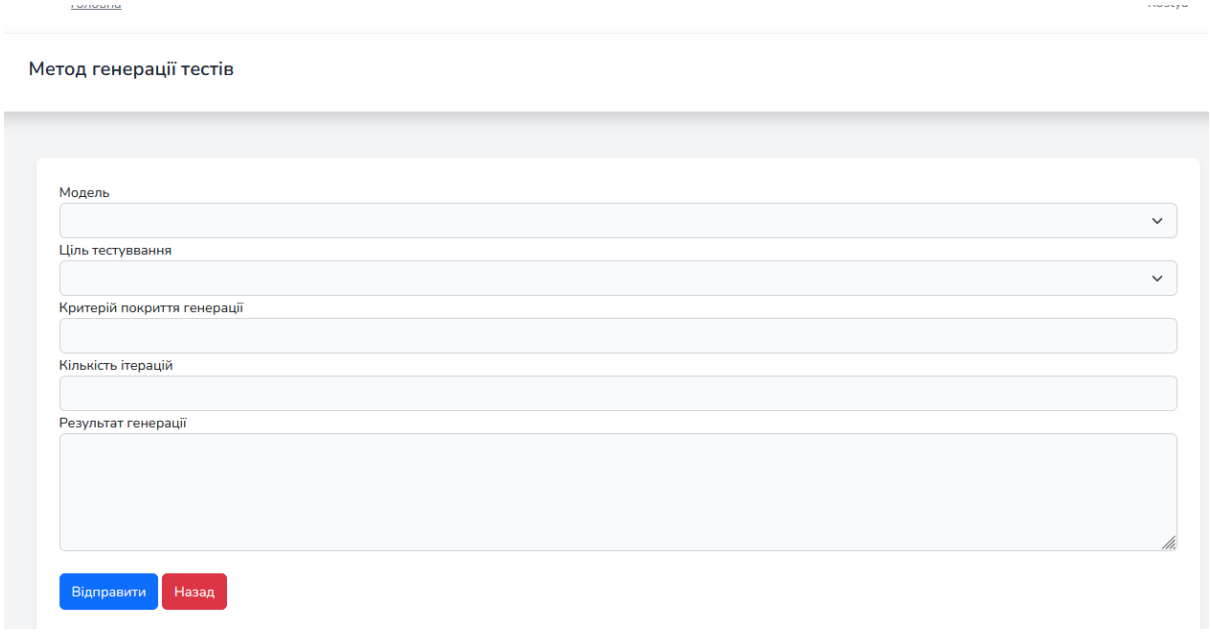


Рисунок 4.1 – Інтерфейс ПЗ

Наступним етапом це розробка функцій для керування БД.

```
return services;
}
class DataBaseModel{

    function __construct()
    {
        DB::$user = 'root';
        DB::$password = '';
        DB::$dbName = 'genereting_test';
    }

    public function insert_data_to_db($value,$table){
        DB::insert($table, $value);
    }

public function get_all_by_param_data_db($table,$param,$value){
```

```
            return DB::query("SELECT * FROM ".$table." WHERE ".
$param." = ". $value);
        }
     public function update_data_to_db($value,$table){
            DB::replace($table, $value);
        }
```

Підключення бази даних відбувається в класі DataBaseModel. Саме в цьому класі знаходиться конвеєр обробки запитів, який займається конфігурацією, підключенням різноманітних проміжних функцій, налаштуванням залежностей та конфігурацією баз даних. Також забезпечує передачу, отримання та оновлення даних бази даних.

Останній етап це розробка мікро-серверну система для використання REST API;

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class ApiController extends Controller
{
    public function getGenetic_data(Request $request)
    {
        $data = [
            'message' => 'GET request data,
            'params' => $request->all(),
        ];

        return response()->json($data);
    }

    public function postGenetic_data (Request $request)
    {
        $data = [
            'message' => 'POST request data',
            'params' => $request->all(),
        ];

        return response()->json($data);
    }
}
```

4.1.3 Реалізація програмної системи через Python

Метою застосування мови програмування Python контексті реалізації методу генерації тестів з пошуком дій та в реалізації генетичного алгоритму. Згідно побудованої моделі методу генерації тестів, яка описана у розділі №2, програмна система самого методу поділяється на такі компоненти:

- вхідні дані;
- модуль формування початкових даних;
- генетичний алгоритм кодогенерації;
- вивід результатів.

Для початку було реалізовано функції які приймають вхідні дані від сервера. Після прийому вхідних даних відповідно вони записуються в стек для тимчасового зберігання.

```
      def GetConnectionString

          try

              var host = Environment.GetEnvironmentVariable(DB_HOST);
              if (string.IsNullOrWhiteSpace(host))
                  throw new Exception(DB_HOST);

              var user = Environment.GetEnvironmentVariable(DB_USER);
              if (string.IsNullOrWhiteSpace(user))
                  throw new Exception(DB_USER);

              var pass = Environment.GetEnvironmentVariable(DB_PASS);
              if (string.IsNullOrWhiteSpace(pass))
                  throw new Exception(DB_PASS);

              var name = Environment.GetEnvironmentVariable(DB_NAME);
              if (string.IsNullOrWhiteSpace(name))
                  throw new Exception(DB_NAME);

              return
$"Host={host};Username={user};Password={pass};Database={param}";

          catch (Exception ex)

              throw new MissingEnvVarException(ex.Message);
```

Наступним етапом є реалізація модуля формування початкових даних. З контексту попередніх розділів, як відомо, даний модуль необхідний формування вхідних даних для генетичного алгоритму. Тобто основна задача даного модуля це формування масиву даних вхідні дані від користувача до них додається вибірка індивідів згідно категорій. Реалізація даного можна поділити на наступні етапи:

– здійснення вибірки популяцій;

– формування json файлу вхідних даних для генетичного алгоритму.

Для початку було розроблено функцію читання даних зі всіх популяцій і запис у стек.

```
data = pd.read_csv('your_dataset.csv')

X = data.drop('target_column', axis=1)
y = data['target_column']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

model = RandomForestClassifier()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

Згодом прописано функцію яка реалізовує здійснення вибірку окремих індивідів зі всі популяції на основі вхідних даних. Ті які співпадають за параметрами – записуються в стек, інші які не підходять по параметрам відповідно ігноруються.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

iris = datasets.load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

```
model = SVC(kernel='linear')
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Точність індивіду: {accuracy}")
```

І останнім етапом є сама реалізація формування json файлу зі списком вибраними індивідами та вхідних для генетичного алгоритму. Даний json файл записується в базу даних.

```
    def train_and_save_model(dataset_path, target_column,
output_file):

        data = pd.read_csv(dataset_path)

        X = data.drop(target_column, axis=1)
        y = data[target_column]


        X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)


        model = RandomForestClassifier()
        model.fit(X_train, y_train)

        y_pred = model.predict(X_test)

        accuracy = accuracy_score(y_test, y_pred)
        print(f"Точність моделі: {accuracy}")

        with open(output_file, 'w') as file:

            for i in range(10):
                file.write(f"Result_data: {y_test.iloc[i]},
Передбачене: {y_pred[i]}\n")

    train_and_save_model('your_dataset.csv', 'target_column',
'output_results.txt')
```

Наступний етап це реалізація алгоритму генетичного програмування. Ініціалізація вхідних даних які сформував модуль

```
     static string SignMessage(string message, string privateKey)
     {
         using (RSACryptoServiceProvider rsa = new
RSACryptoServiceProvider())
         {
             rsa.FromXmlString(privateKey);

             byte[] messageBytes = Encoding.UTF8.GetBytes(message);
             byte[] signatureBytes = rsa.SignData(messageBytes, new
SHA256CryptoServiceProvider());
         with open(read_data, 'w') as data:
             for i in range(10):
                 file.write(f"Очікуване: {y_test.iloc[i]},
Передбачене: {y_pred[i]}\n")


             return Convert.ToBase64String(signatureBytes);
         }
     }
```

Реалізація генетичного алгоритму виглядає наступним чином.

```
     def genetic_algorithm(population_size, generations):
         # Ініціалізація початкової популяції випадковими значеннями
         population = np.random.uniform(-5.0, 5.0,
size=(population_size,))

         for generation in range(generations):
             # Оцінка пристосованості (застосовуємо функцію, яку
хочемо оптимізувати)
             fitness_scores = [fitness_function(x) for x in
population]

             # Вибір найкращих особин за значеннями пристосованості
             selected_indices = np.argsort(fitness_scores)
             selected_population =
population[selected_indices[:population_size//2]]

             # Схрещування (в цьому прикладі - одноточкове
схрещування)
             crossover_point = np.random.randint(1,
population_size//2)
             children =
np.concatenate([selected_population[:population_size//4],

selected_population[population_size//4:]],
                                        axis=0)
             for i in range(population_size//4):
                 children[i + population_size//4] = np.concatenate([
```

```
                    selected_population[i, :crossover_point],
                    selected_population[population_size//4 + i,
crossover_point:]
                ])
            mutation_noise = np.random.normal(0, 0.1,
size=(population_size//2,))
            children += mutation_noise

            population = np.concatenate([selected_population,
children], axis=0)
        best_solution = population[np.argmin(fitness_scores)]
        best_fitness = min(fitness_scores)
        return best_solution, best_fitness
    best_solution, best_fitness =
genetic_algorithm(population_size= population_size, generations=
generations)
```

Алгоритм проведення мутації тестових сценаріїв завдяки якому і генеруються нові індивідууми.

```
import numpy as np
def mutate(individual, mutation_rate=0.1):
    mutated_individual = individual.copy()


    mutation_mask = np.random.rand(len(individual)) <
mutation_rate


    mutated_individual[mutation_mask] += np.random.normal(0,
0.1, size=np.sum(mutation_mask))


    return mutated_individua
```

4.2 Тестування методу. Аналіз результатів

Тестування програмної системи є важливим елементом у життєвому циклі розробки програмного забезпечення. Процес тестування виявляє помилки та недоліки програмної системи, дозволяє зрозуміти, які модулі чи компоненти системи потребують оптимізації.

Для емпіричного дослідження розробленого ПЗ використовуються два види тестування: напівавтоматичне, модульне та навантажувальне.

Напівавтоматичне тестування дозволяє використовувати комбінацію ручного тестування та певних додаткових інструментів автоматизованого тестування. По ручному тестуванню базово визначено працездатність ПС шляхом виконання ряду дій:

– підбір різної комбінації даних для форми заповнення;

– здійснення стрес тестування шляхом виконання хаотичних, різких та не передбачуваних дій зі сторони користувача з метою визначення стійкості системи.

Під час проведення ручного тестування система працювала надійно, без всіляких зависань і не видала жодних повідомлень про помилки.

Щодо автоматизованої тестування необхідно використати допоміжні бібліотеки, які дають змогу працювати з автоматизованими тестами. Jest дає змогу писати модульні тести для додатків, які розроблені мовою JavaScript. Його перевагами є:

– простота;

– велика кількість документації;

– ізольований запуск тестів.

Даний сервіс має змогу генерувати велику кількість тестових наборів даних. Завдяки цьому можна було легко підставляти тестові дані і здійснювати аналіз роботи системи. Генерація наборів була як з повторенням так і без. Під час проведення даного тесту система працювала стабільно та якщо і були спеціально згенеровані хибні дані, відповідно, ПС реагувала згідно правил та виводила помилки.

Модульне тестування має на меті окреме тестування кожної функції та модуля програмного засобу. Навантажувальне тестування має на меті довести, що система залишається стійкою, незважаючи на різку зміну навантаження. Оскільки алгоритм працюватиме вірно лише, якщо всі функції системи працюють правильно, то кожна функція системи протестована за допомогою модульного

тестування. Також було проведене навантажувальне тестування, що дало змогу переконатись у працездатності методу генерацій тестів.

У процесі навантажувального тестування було емулювано навантаження у 500 користувачів, які циклічно виконують декілька послідовних запитів до додатку (ці дії повторювались сім разів). Одна з кінцевих точок додатку працювала повільно та повертала помилку

Також було додано три шкідливих користувачі, які виконували запити до неробочої кінцевої точки, запит до якої призводив до відмови роботи програмної системи.

Результати тестування продемонстровано наочно(Рисунок 4.1-4.2) згенеровані тести запитів та форм даним методом та вказана успішне їхнє виконання.

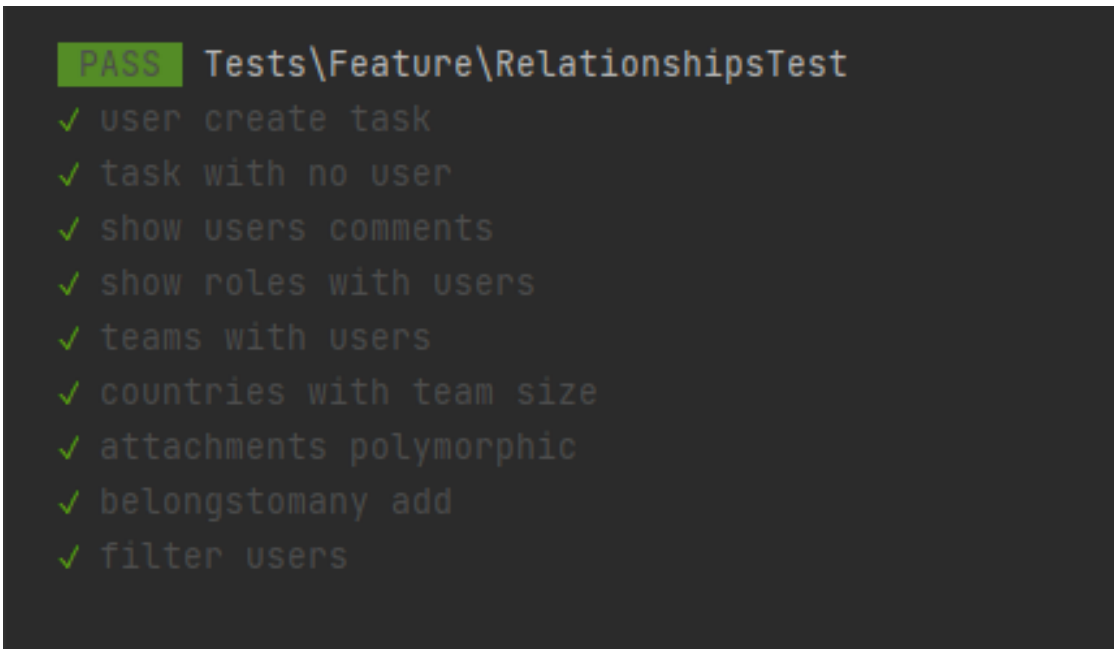


Рисунок 4.1 – Результат виконання згенерованих тестів веб-форм.

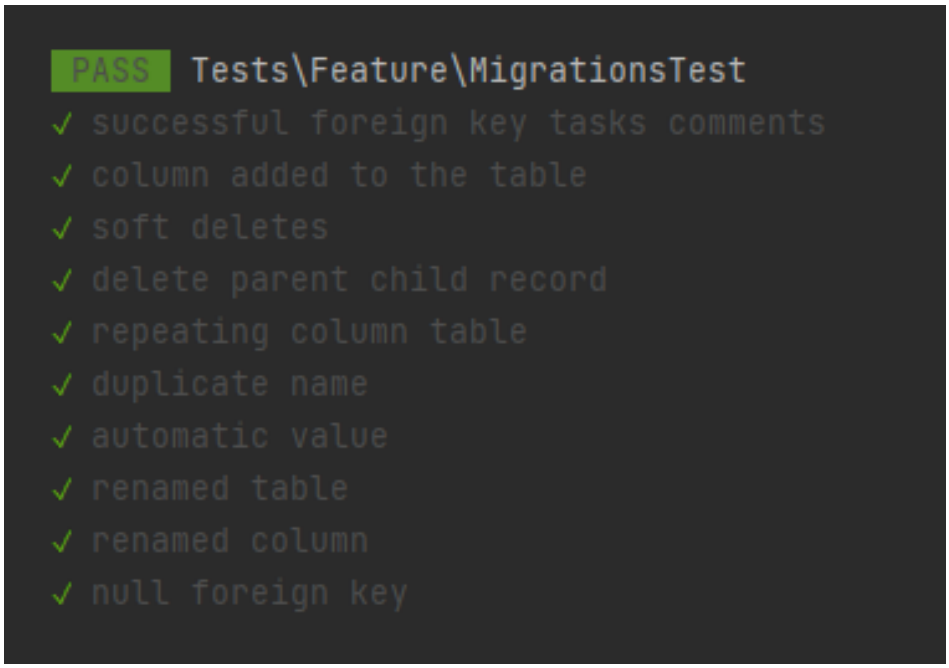


Рисунок 4.2 – Результат виконання згенерованих тестів запитів.

4.3 Оцінка ефективності методу генерацій тестів

Для того, щоб отримати висновок стосовно ефективності розробленої програмної системи, необхідно провести теоретичну та практичну оцінку реалізованого метода.

По перше, що можна зауважити, що під час використання даного методу, який використовує генетичний алгоритм, не було жодного випадку зависань чи якихось жорстких відхилень структур згенерованих тестів. Це вже свідчить про значний успіх реалізації даного методу.

По-друге, практична апробація отриманих результатів показала, що даний метод здатен успішно генерувати тести для різноманітних програмних систем, включаючи ті, які мають складну логіку та великий обсяг коду. Застосування генетичного алгоритму для автоматизованої генерації тестів покращує якість та покриття тестування, а також дозволяє здійснювати швидкий та ефективний пошук помилок у програмах. Це підтверджує ефективність методу у вирішенні завдань тестування програмного забезпечення та підтримує його застосування в реальних умовах розробки.

По третє. Провівши аналіз якості придатності із 266 згенерованих поколінь взявши 6 випадкових поколінь, де 3 покоління з категорії тестування веб-форм та 3 покоління з категорії тестування запитів було побудовано наступні графіки.

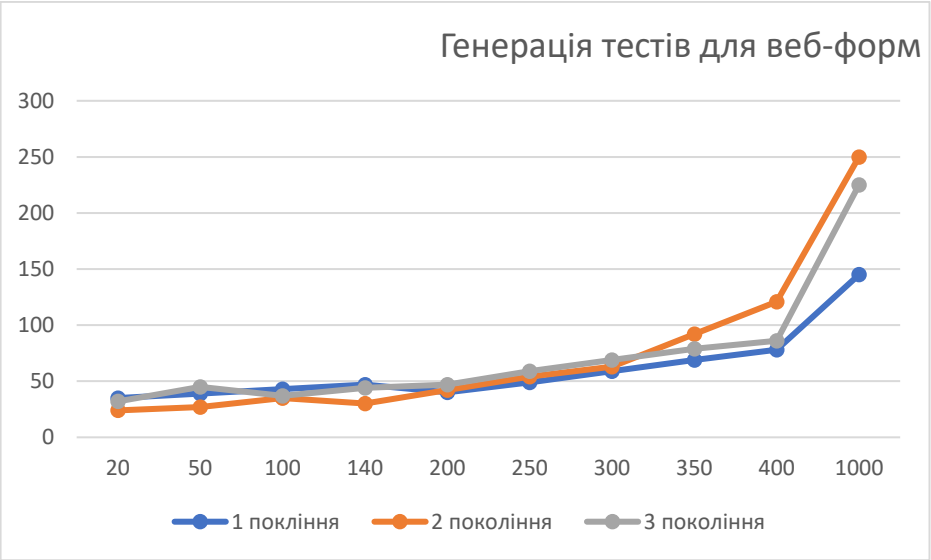


Рисунок 4.3 – Графік оцінки придатності згенерованих тестів для веб-форм

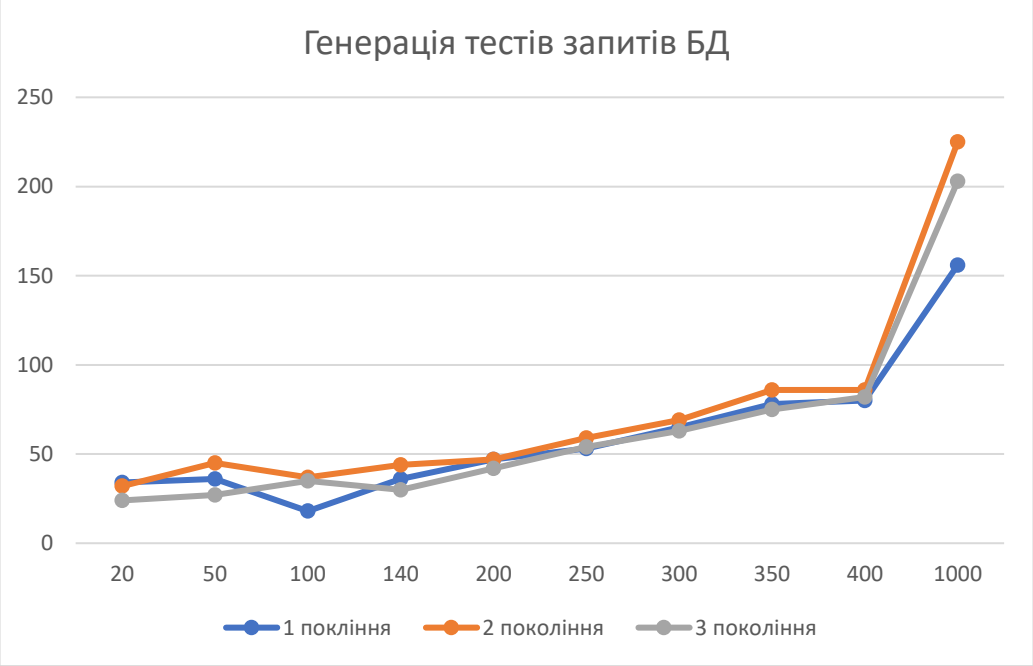


Рисунок 4.3 – Графік оцінки придатності згенерованих тестів запитів.

Виходячи з даного графіку можна виявити певну закономірність. Приблизно 50-100 популяцій у всіх показник якості здійснює «просадку». Проте,

всі як один, після 400 популяцій показують чудовий результат. Це свідчить про стабільність роботи даного методу.

4.4 Висновок

У даному розділі кваліфікаційної роботи визначено основні засоби для реалізації програмної системи, визначені їхні характеристики причини їх вибору та визначені завдання які вони мають виконувати.

Основними засобами реалізації програмної системи обрано:

– php фреймворк Laravel;

– мова програмування Python.

Laravel фреймворк застосовується в контексті ПС наступним чином:

– побудова інтерфейсу користувача;

– побудова мікро-серверну система для використання REST API;

– керування інтерфейс керування БД;

– перевірка згенерованих тестів інструментами формування модульних

Python застосовується в контексті ПС як в реалізації методу генерації тестів з пошуком дій та у реалізації генетичного алгоритму.

В результаті опису програмних модулів було проведено безпосередню програмну реалізацію кожного модуля обох компонентів програмної системи.

На основі розроблених тестових сценаріїв було проведено тестування та аналіз результатів тестування. Оцінивши результати тестування було зроблено висновок про високу надійність роботи даного методу.

Наостанок було проведено оцінку ефективність згенерованих тестів шляхом побудови діаграми, обґрунтовано їхню ефективність та необхідність реалізації саме в такому вигляді.

ВИСНОВКИ

Отже, результатом виконання даної кваліфікаційної роботи є удосконалення методу генерацій тестів з пошуком дій.

В ході написання першого розділу кваліфікаційної роботи було досліджено предметну область, останні напрацювання у сфері кодогенерації, проаналізовано наявні методи та засоби вдосконалення кодогенерації, а також сформовано задачі на кваліфікаційну роботу.

У другому розділі описано концепції алгоритмів із класифікації машинного навчання, побудовано модель методу генерації тестів та визначено методи вирішення поставлених задач. Запропоновані концепції, моделі та методи було описано та обґрунтовано необхідність їх застосування

В ході написання третього розділу кваліфікаційної роботи було проведено дослідження та аналіз вимог до програмного засобу, спроєктовано структуру програмного засобу та структуру даних, визначено вхідні дані для методу та описані завдання для модулю формування початкових даних, здійснено детальний опис роботи генетичного алгоритму. Також було проаналізовано та обрано засоби, необхідні для програмної реалізації розроблених методів та моделей, визначено основну область тестування.

В ході написання четвертого розділу кваліфікаційної роботи було описано структуру та призначення модулів програми, а також взаємозв’язок між ними, проведено безпосередню програмну розробку спроєктованих модулів, зокрема реалізовано генетичний алгоритм, засоби які допомагають якісно проводити кодогенерацію тестів.

На основі отриманих результатів було проведено емпіричне дослідження для обґрунтування працездатності та функціональної придатності методу генерації тестів.

Удосконалений метод генерації тестів з пошуком дій полегшить процес тестування програмного забезпечення для програмістів, що зробить його значно

продуктивнішим та цікавішим, дозволить автоматизувати створення тестів, спрямованих на виявлення помилок та недоліків у програмному забезпеченні.

Удосконалений метод дозволяє підвищити продуктивність, швидкість та надійність процесу тестування. Завдяки застосуванню ряду інноваційних методів як МФПД та генетичного алгоритм забезпечує ефективність при виборі та комбінувати дії для кожного тестового сценарію, що сприяє збільшенню покриття коду та виявленню складних взаємодій між компонентами програми.

Застосування даного методу в реальних проектах може призвести до поліпшення процесу тестування, зменшення кількості помилок та витрат на тестування. Отримані результати свідчать про великий потенціал та перспективи використання генетичних алгоритмів у сфері розробки та тестування програм.

Цей метод не лише надає автоматизовану можливість створення тестів, але й створює основу для подальших досліджень та оптимізації у галузі генерації тестів для розширених та складних програмних систем.

Згідно з результатами досліджень було опубліковано тези доповіді на Всеукраїнській науково-практичній конференції «Актуальні проблеми комп’ютерних наук АПКН-2023» [20]. Копії наукових публікацій подані у додатку В.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Сфера розробки програмного забезпечення є однією з найшвидше зростаючих галузей URL: https://ain.ua/2022/04/30/zvit-digital-2022/ (дата звернення: 21.09.2023).

2. Галузь розробки програмного забезпечення може зростати на 5-10% річно. URL: https://www.epravda.com.ua/news/2019/11/25/654105/ (дата звернення: 07.10.2023).

3. Загальні принципи тестування ПЗ. URL: https://geteasyqa.com/qa/mobile-apps-testing/ (дата звернення: 19.10.2023).

4. Тестування програмного забезпечення може займати від 30% до 50% від загального часу. URL: https://www.ukrinform.ua/rubric-technology/3497671-blizko-78-ukrainciv-sodna-koristuutsa-internetom.html (дата звернення: 19.10.2023).

5. Принципи модульного тестування. URL: https://www.techtarget.com/searchsoftwarequality/definition/unit-testing (дата звернення: 19.10.2023).

6. Оформлення тест-планів. URL: https://qalight.ua/baza-znaniy/test-case-2/(дата звернення: 21.10.2023).

7. Покриття тестування. URL: https://training.qatestlab.com/blog/technical-articles/requirements-coverage/ (дата звернення: 22.10.2023).

8. Алгоритми кодогенарації. URL: https://www.geeksforgeeks.org/simple-code-generator/ (дата звернення: 1.11.2023).

9. Основні алгоритми класу машинного навчання. URL: https://evergreens.com.ua/articles/machine-learning-overview.html (дата звернення: 1.11.2023).

10. Символічне виконання. URL: https://www.miyklas.com.ua/p/informatica/7-klas/algoritmi-ta-programi-python-379620/riadkovi-velichini-380259/re-40a82b7a-e349-4d80-aa3a-6d1692db8713 (дата звернення: 2.11.2023).

11. Використання машинного навчання URL: https://www.zfort.com.ua/blog/cekretni-sili-mashinnogo-navchannya-oglyad-algoritmiv-mashinnogo-navchannya (дата звернення: 12.11.2023).

12. Модельна перевірка. URL: https://www.sim-networks.com/ukr/blog/data-encryption-best-practices (дата звернення: 12.11.2023).

13. Генетичний алгоритм. URL: http://www.znannya.org/?view=ga (дата звернення: 12.11.2023).

14. Офіційний сайт документації Laravel URL : https://laravel.com/docs/10.x/middleware (дата звернення: 24.11.2023).

15. Інструменти тестування Laravel. URL: https://laravel.su/docs/8.x/testing (дата звернення: 24.11.2023).

16. REST API. Laravel URL: https://aws.amazon.com/ru/what-is/restful-api/ (дата звернення: 24.11.2023).

17. Python. URL: https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python (дата звернення: 25.11.2023).

18. Python DEAP. URL: https://deap.readthedocs.io/en/master/ (дата звернення: 02.12.2023).

19. Intro to Evolutionary Computation Using DEAP in Python . URL: https://deap.readthedocs.io/en/master/ (дата звернення: 02.12.2023).

20. Дуда К.М., Кустовський Р.С. Метод генерацій тестів програмного забезпечення з пошуком певних дій// Актуальні проблеми комп’ютерних наук АПКН-2023: Збірник наукових праць за матеріалами XV Всеукраїнської науково-практичної конференції, м. Хмельницький, 17-18 листопада 2023 р. Хмельницький, 2023. С. 84–87.

ДОДАТОК А
(обов’язковий)

СХЕМА ВЗАЄМОДІЇ ПС

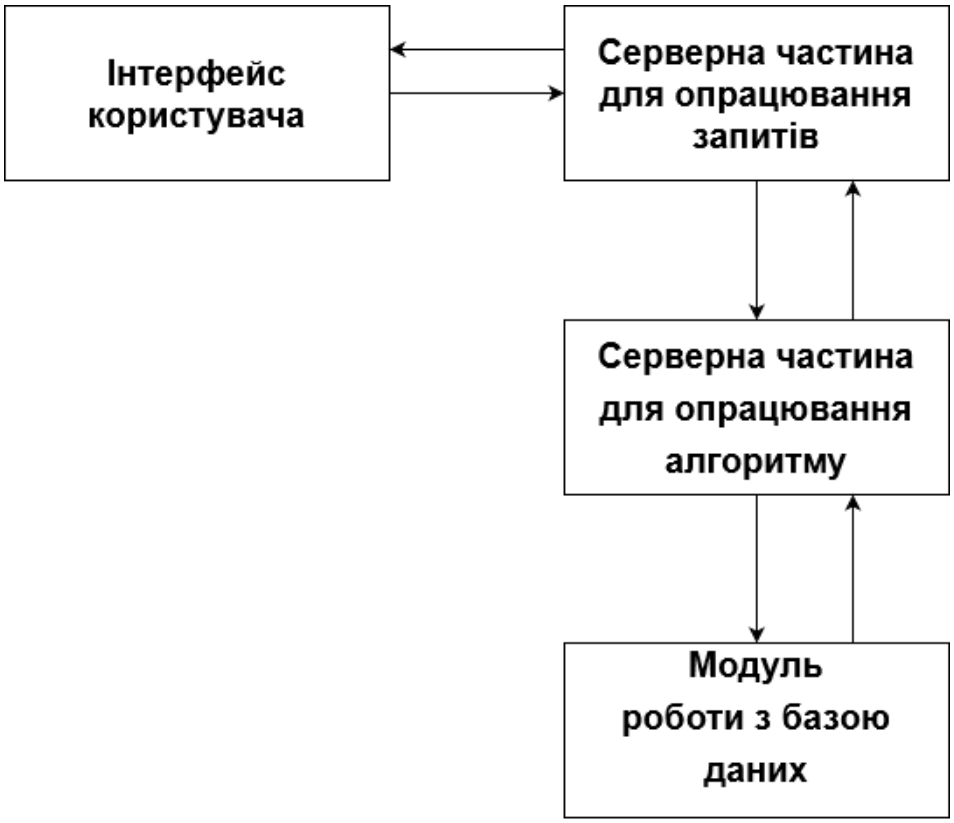


Рисунок А.1 – Схема взаємодії компонентів ПС

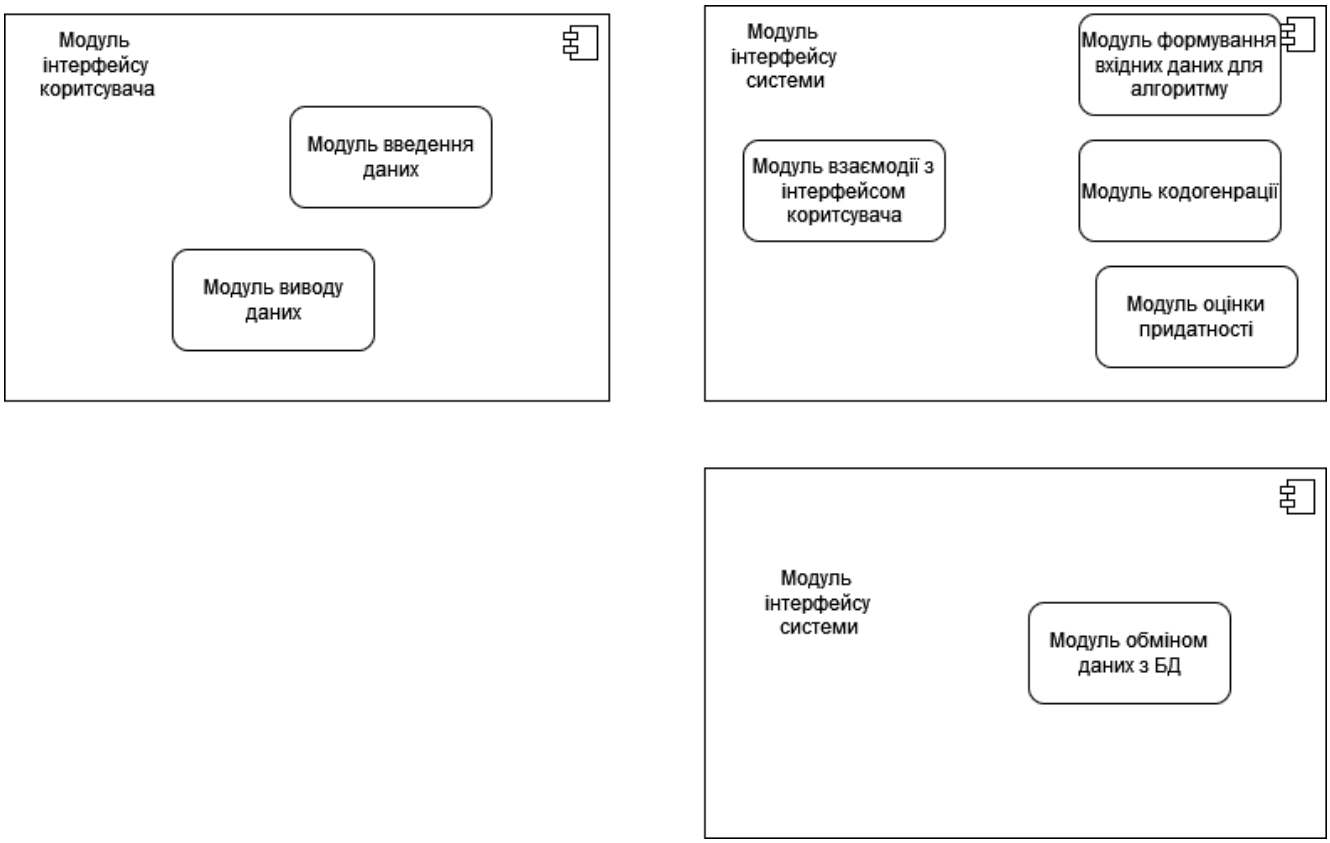


Рисунок А.2 – Схема взаємодії елементів ПС

СХЕМА МЕТОДУ ГЕНЕРПЦІЇ ТЕСТІВ

Рисунок А.3 – Структура методу генерації тестів

ДОДАТОК Б
(обов’язковий)

МОДЕЛІ СТРУКТУРИ ДАНИХ

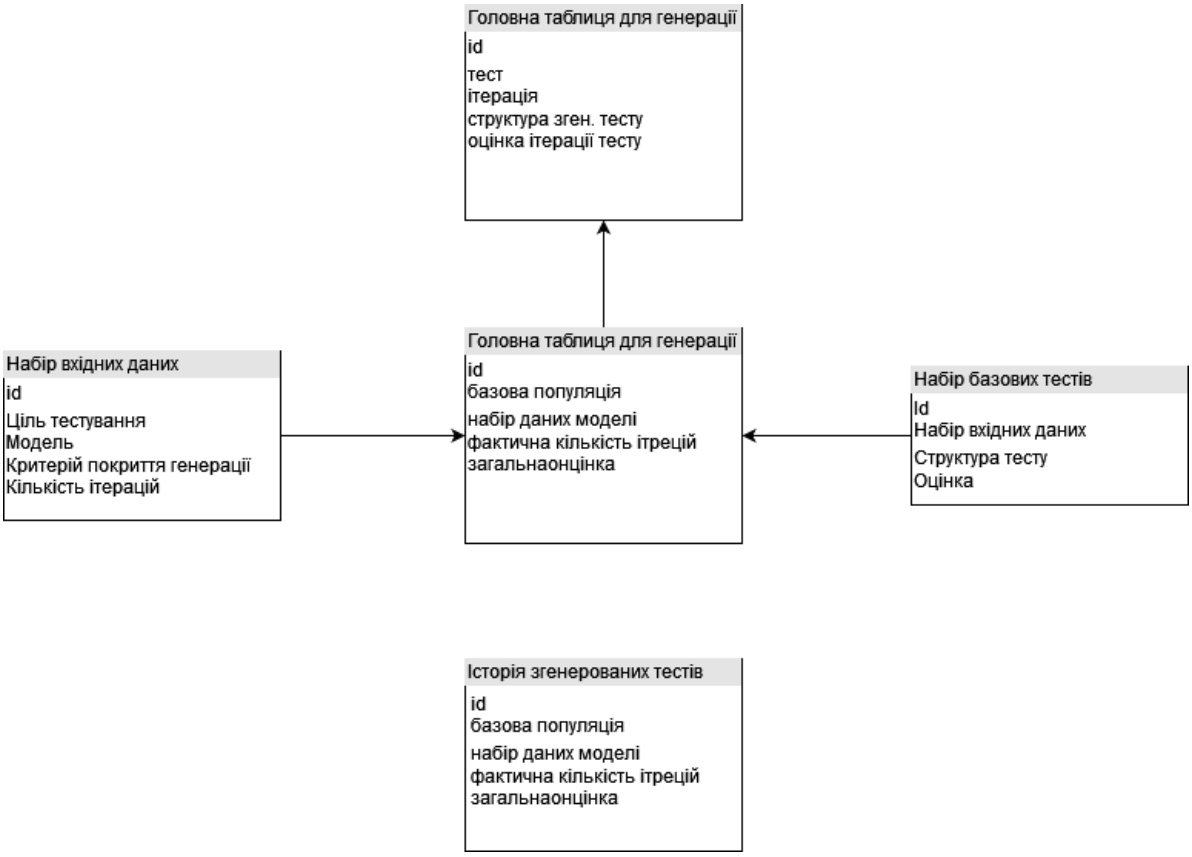


Рисунок Б.1 – Схема БД

ДОДАТОК В
(обов’язковий)

КОПІЯ НАУКОВОЇ ПУБЛІКАЦІЇ

Міністерство освіти і науки України
Хмельницький національний університет

ЗБІРНИК НАУКОВИХ ПРАЦЬ

за матеріалами XV Всеукраїнської науково-практичної конференції
«Актуальні проблеми комп'ютерних наук АПКН-2023»

17-18 листопада 2023

Хмельницький 2023

УДК 004:37:001:62

Збірник наукових праць за матеріалами XV Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2023». Хмельницький. 2023. 345с.

У збірнику наукових праць подані перспективні практичні розробки аспірантів, студентів та здобувачів в області сучасних інформаційних технологій. Розглянуто актуальні проблеми комп'ютерних наук, комп'ютерної інженерії, прикладної математики й інженерії програмного забезпечення, приведено ряд робіт по впровадженню інформаційних технологій у виробництво та управління. Висвітлено перспективні розробки сучасних систем пошуку, обробки й захисту інформації, медійних та комунікаційних системи.

УДК 004:37:001:62

Матеріали конференції відтворені з авторських оригіналів. При макетуванні можливі незначні зміни компоновки контенту авторських оригіналів.

Участь у конференції та складові всіх її етапів (розгляд праць, макетування, публікація збірника наукових праць та видача сертифікатів) є безкоштовними для всіх учасників. Оргкомітет конференції висловлює подяку учасникам конференції та сподівається на подальшу співпрацю.

З питань проведення конференції та подальшого обміну інформацією звертатись на e-mail конференції: *apkt.khnu@gmail.com*

УДК 004.4

Дуда К.М., Кустовський Р.С.

Хмельницький національний університет

МЕТОД ГЕНЕРАЦІЇ ТЕСТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ПОШУКОМ ПЕВНИХ ДІЙ

Розглянуто основні аспекти розробки методу генерації тестів на основі пошук повних дій. Даний метод здійснює генерацію коду для тестування на основі введених даних користувачем які вказують які дії необхідно протестувати, цим самим забезпечує збільшує продуктивність та якість процесу тестування програмного забезпечення.

The main aspects of the development of the test generation method based on the search for complete actions are considered. This method generates code for testing based on the data entered by the user, which indicates which actions need to be tested, thereby increasing the productivity and quality of the software testing process.

Тестування програмного забезпечення відіграє надзвичайно важливу роль у розробці та підтримці сучасних програм і застосунків. В процесі постійного зростання комп'ютеризації і залежності від програм, функціональність та надійність стали критичними факторами для задоволення потреб користувачів та досягнення бізнес-цілей.

На сьогоднішній день є два основних види тестувань[1]:

- ручне тестування;
- автоматизоване тестування.

Ручне тестування – це процес, в якому тестувальники вручну перевіряють програмне забезпечення на наявність помилок, відповідність вимогам та забезпечують якість продукту. Це включає в себе створення тестових сценаріїв, виконання їх, аналіз результатів та документування виявлених проблем. Ручне тестування використовується для перевірки аспектів, які важко автоматизувати, а також для тестування користувацького інтерфейсу та інших аспектів, які потребують людського інтуїтивного розуміння. Ручне тестування може бути ефективним, але воно ресурсозатратне, і тому автоматизація тестування стає важливою для покращення ефективності та точності процесу тестування.

Автоматизоване тестування – це процес використання спеціальних програмних інструментів та сценаріїв для автоматичного виконання тестів програмного забезпечення. Його головна мета – зменшити ручне тестування, полегшити і прискорити процес виявлення помилок та забезпечити більшу надійність програмного продукту.

Загалом принцип тестування полягає в тому, щоб порівнювати фактичний та очікуваний результат певного функціоналу. Перед початком тестування складається тест-план в якому описується функціонал, що тестується при певних умовах, очікуваний та фактичний результат після виконання функції (таблиця 1). Очікуваний результат – це результат який після виконання певної функції видає дані згідно сценарію. Фактичний результат – це той результат або стан, який спостерігається після виконання тесту чи функції. Якщо фактичний і очікуваний результат співпадають – тест пройдений, якщо не співпадають – тест не пройдений[3].

Таблиця 1 – Приклад тест-плану.

№	Очікуваний результат	Фактичний результат	Статус
Тест 1	При натисканні «кнопки1» має завантажуватись текстовий файл.	При натисканні «кнопки1» текстовий файл НЕ завантажується.	Тест не пройдений
Тест 2	При натисканні «кнопки2» має закриватись «блок1».	При натисканні «кнопки2» закривається «блок1».	Тест пройдений

Загалом процес тестування є важким, масивним процесом життєвого циклу програмного забезпечення, але в першу чергу є надзвичайно важливим для його розробки.

Метою роботи є розробка методу генерації тестів з пошуком певних дій який перед усім дозволить спростити та пришвидшити процес тестування програмного забезпечення.

Постановка завдання складається з наступних етапів:

– проаналізувати сферу тестування програмного забезпечення;
– дослідити наявні підходи та концепції генерації тестів;
– виконати проектування програмної системи на основі розробленого алгоритму генерації тестів;
– проаналізувати та обрати засоби реалізації даного методу;
– виконати програмну реалізацію методу генерації тестів з пошуком певних дій;
– провести тестування та практичну апробацію отриманих результатів;
– проаналізувати отримані результати та сформувати рекомендації щодо доцільності і впровадження результатів дослідження.

Найпопулярнішим методом автоматизованого тестування є метод тестування чорної скриньки [2]. Метод чорних скриньок (Black Box Testing) - це метод тестування програмного забезпечення, при якому тестувальник не має знань

про внутрішню структуру чи реалізацію програми. Тестувальник працює з програмою як із "чорною скринькою", де він бачить лише вхідні дані та очікувані результати, але не знає, як саме програма обробляє дані. Тобто основним принцип полягає в тому, що не важливо як програма виконує ту чи іншу дію, а важливо який результат вона має виводити. Саме цей вид тестування підходить для методу генерації тестів.

Генетичне програмування – це потужний підхід для генерації коду тестування чорних скриньок та автоматизації тестування. Воно використовує принципи еволюції та відбору для створення оптимальних тестових сценаріїв. Приклад алгоритму генетичного програмування наведено на рисунку 1.

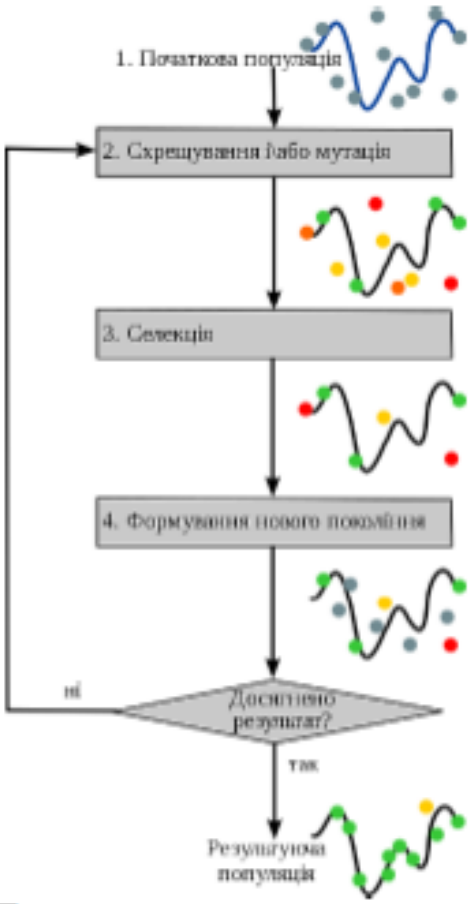


Рисунок 1 – Приклад алгоритму генетичного програмування

Загальний опис процесу використання генетичного програмування [4]:

– **Створення початкової популяції:** генерується велика кількість початкових тестових сценаріїв або хромосом, кожна з яких представляє собою можливий тест.

– **Оцінка пристосованості:** кожен тест оцінюється за допомогою функції пристосованості, яка визначає, наскільки добре він виконує свою роботу.

– **Схрещування:** тестові сценарії об'єднуються (схрещуються) для створення нових тестів, які можуть успадковувати корисні характеристики батьківських тестів.

– **Мутація:** деякі тестові сценарії піддаються випадковим змінам, щоб розширити різноманітність тестів.

– **Відбір:** вибираються кращі тестові сценарії на основі їхньої пристосованості, і вони використовуються для створення наступного покоління тестів.

– **Повторення:** процес схрещування, мутації та відбору повторюється протягом кількох поколінь.

Метод генерації тестів з використанням алгоритму генетичного програмування зосереджується на створенні тестових випадків для програмного забезпечення з точки зору зовнішньої поведінки (метод чорної скриньки). Основною метою є визначення, як програма взаємодіє з вхідними даними і генерує вихідні результати, без необхідності знань про її внутрішню структуру.

В результаті, генетичне програмування може створити оптимальні тестові сценарії, які відповідають вимогам тестування. Цей підхід особливо корисний для автоматизованого тестування складних програмних систем.

Дослідження у галузі генерації тестів для програмного забезпечення з використанням генетичного програмування свідчить про важливість та актуальність цього напрямку в розробці програм. Застосування методів штучного інтелекту для генерації тестів може полегшити процес тестування, покращити якість програм та зменшити трудомісткість тестування. Це дослідження слугує основою для подальших робіт у цій галузі та розвитку більш складних методів генерації тестів.

Перелік посилань

1. Загальні принципи тестування ПЗ URL: https://www.browserstack.com/guide/learn-software-application-testing
2. Оформлення тест-планів URL: https://softwaretestingfundamentals.com/test-plan/.
3. Sadeeq Jan, Annibale Panichella, Andrea Arcuri, and Lionel Briand. Search-based multi-vulnerability testing of xml injections in web applications. Empirical Software Engineering, pages 1–34, 2019.
4. Алгоритм генетичне програмування URL: https://www.geeksforgeeks.org/

ДОДАТОК Г

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Кафедра інженерії програмного забезпечення

Метод генерації тестів для програмного забезпечення на основі пошуку для певних дій

Виконав: *студент гр. ІПЗм-22-1* *Дуда К.М*

Керівник: *канд. техн. наук, доцент* *Яшина О. М.*

Об'єкт, предмет, мета дослідження

Мета. Удосконалення методу генерації тестів для програмного забезпечення, спрямованого на покращення ефективності тестування і виявлення вразливостей програм.

Об'єкт. Процес генерації тестів для програмного забезпечення.

Предметом дослідження є методи та алгоритми генерації тестів для програмного забезпечення.

Актуальність роботи

Актуальність теми полягає в тому, що з ростом складності програмного забезпечення і збільшенням обсягів даних, тестування стає все більш важливим завданням. Для підтримання високої якості програм та забезпечення їх надійності, необхідно розробляти та впроваджувати нові методи генерації тестів, які враховують специфіку програм і призводять до покращення ефективності тестування.

Завдання дослідження

Принцип застосування

Даний метод реалізований з метою здійснення генерації модульних тестів для Laravel З пошуком таких дій такі як

- Тестування веб-форм
- Тестування запитів БД

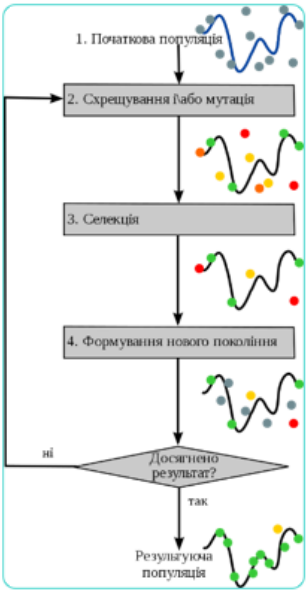
Використані Технології

СТРУКТУРА МЕТОДУ

Генетичний алгоритм

Генетичний алгоритм (ГА) — це еволюційний алгоритм пошуку, що використовується для вирішення задач оптимізації і моделювання шляхом послідовного підбору, комбінування і варіації шуканих параметрів з використанням механізмів, що нагадують біологічну еволюцію.

Генетичний алгоритм



Модуль формування початкових даних

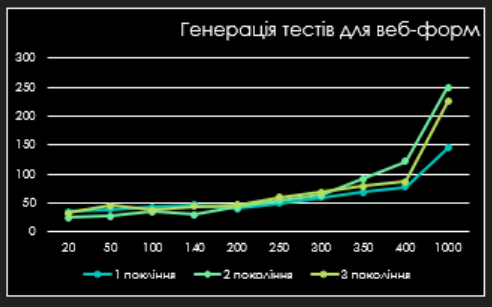
Його основні функції

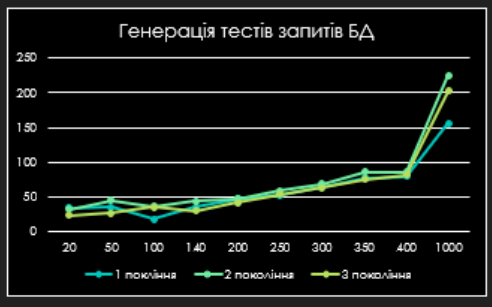
- Формування та передача вхідних параметрів.
- Генерація початкових тестових кейсів для генетичного алгоритму
- Формування оцінки придатності
- Визначення кількості ітерацій

ВХІДНІ ДАНІ ДЛЯ ГЕНЕТИЧНОГО АЛГИРИМТУ

Назва параметру	Тип даних	Приклад даних
Ціль тестування	Символьна. Вибірка із доступних варіантів	Форма
Модель	Об'єкт класу. Вибірка із доступних варіантів	User
Параметри моделі	Json	«{'name' => 'name', 'age=> 2'…}»
Критерій покриття генерації	Числовий.	30
Кількість ітерацій	Числовий.	400
Вибірка з початкових індивідів	Json	«{ 'індивід 1','індивід 2',…'індивід n'} »

РЕЗУЛЬТАТИ ГЕНЕРАЦІЙ





Наукова новизна

- Новизна даної теми полягає в поєднанні методів генерації тестів з алгоритмами генерації для виявлення вразливостей у програмному забезпеченні.
- Розроблені методи та алгоритми можуть сприяти автоматизації процесу тестування і покращити його ефективність.
- Вперше визначено підхід до кодогенерації з використанням генетичного алгоритму та ряд інших чинників

Практичне значення

Практична цінність даного методу дуже велика і охоплює різні сфери:

- Покращення якості ПЗ
- Ефективність тестування.
- Автоматизація неперервної інтеграції
- Зменшення ризиків та витрат

ВИСНОВКИ

- Розроблений метод дозволяє підвищити продуктивність, швидкість та надійність процесу тестування. Завдяки застосуванню ряду інноваційних методів як МФПД та генетичного алгоритм забезпечує ефективність при виборі та комбінувати дії для кожного тестового сценарію, що сприяє збільшенню покриття коду та виявленню складних взаємодій між компонентами програми.

НАУКОВІ ПУБЛІКАЦІЇ

- Дуда К.М., Кустовський Р.С. Метод генерацій тестів програмного забезпечення з пошуком певних дій// Актуальні проблеми комп'ютерних наук АПКН-2023: Збірник наукових праць за матеріалами XV Всеукраїнської науково-практичної конференції, м. Хмельницький, 17-18 листопада 2023 р. Хмельницький, 2023. С. 84–87.

UNICHECK
by Turnitin

Ім'я користувача:
ІПЗ

ID перевірки:
1015993585

Дата перевірки:
11.12.2023 16:23:16 EET

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
11.12.2023 16:23:51 EET

ID користувача:
100012953

Назва документа: **Магістерська Дуда-ПЛАГІАТ**

Кількість сторінок: **85** Кількість слів: **13936** Кількість символів: **115109** Розмір файлу: **902.74 KB** ID файлу: **1015676174**

7.54%
Схожість

Найбільша схожість: **4.31%** з джерелом з Бібліотеки (ID файлу: **1015667908**)

4.28% Джерела з Інтернету	302	Сторінка 87
6.82% Джерела з Бібліотеки	104	Сторінка 88

0.04% Цитат

Цитати	1	Сторінка 89

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	1

Mon Dec 11 15:25:13 EET 2023, Форкун Юрій Вікторович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 2.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 6%**

	Документ		Сумарний збіг по Базі Даних	
ID: 122552 Назва: Метод генерації тестів для програмного забезпечення на основі пошуку для певних дій Додано в БД: 2023-12-11 Автора: Дуда Костянтин Керівники: Яшина Оксана Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	98626	856	4284 (4%)	42 (5%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Костянтина ДУДИ
факультет ІТ, 2 курс, група ІПЗм-22-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений . Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

07.12.2023
дата

підпис

Завідувачу кафедри
інженерії програмного забезпечення
проф. Леоніду БЕДРАТЮКУ
студента групи ІПЗм-22-1
Костянтин ДУДА
Ім'я, ПРІЗВИЩЕ

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «магістр» за спеціальністю 121 «Інженерія програмного забезпечення»:

Метод генерації тестів для програмного забезпечення на основі пошуку для певних дій

(керівник кваліфікаційної роботи – Оксана ЯШИНА)
Ім'я, ПРІЗВИЩЕ

06.12.2023
Дата

Підпис здобувача

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів подібності щодо роботи, продукованими програмно-технічним засобом(ами) перевірки текстів на плагіат.
Назва: «Метод генерації тестів для програмного забезпечення на основі пошуку для певних дій»
Автор: Дуда Костянтин Михайлович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Яшина Оксана Миколаївна, кандидат технічних наук, доцент
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені у розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за два дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи.	
3	Виявлені запозичення не є плагіатом, але частково розміщені у розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того, як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:
Запозичення, виявлені у роботі, є законними і не є плагіатом, оскільки:
1) у тексті кваліфікаційної роботи системою перевірки на плагіат Unicheck виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, бланк завдання), у структурі змісту, у назвах розділів/підрозділів, у назвах публікацій переліку джерел посилання тощо;
2) в якості запозичень системою Unicheck було зафіксовано деякі послідовності вихідного коду і посиланання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;
3) запозичення, виявлені в тексті роботи, є фрагментарними або мають належним чином оформленні посилання;
4) виявлені модифікації тексту не впливають на відсоток схожості.
Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає ___. Обсяг запозичень, визначений системою Unicheck виявлення збігів ідентичності/схожості, складає ___ і адресується до ___ джерел з інтернету і ___ джерела з бібліотеки, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 11.12.2023 р.

Завідувач кафедри ІПЗ _____ Леонід БЕДРАТЮК
Гарант освітньої програми _____ Оксана ЯШИНА
Керівник кваліфікаційної роботи _____ Оксана ЯШИНА

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «магістр»

Здобувач Дуда Костянтин Михайлович

Тема Метод генерації тестів для програмного забезпечення на основі пошуку для певних дій

Спеціальність 121 «Інженерія програмного забезпечення»

Обсяг кваліфікаційної роботи:

Кількість листів креслень _____; кількість сторінок записки 113

1. Короткий зміст роботи та прийнятих рішень У даній кваліфікаційній роботі проведено аналіз предметної області тестування програмного забезпечення, визначена основна проблематика тестування ПЗ. На основі проведеного аналізу було розроблено метод генерації тестів з пошуком дій для php фреймворку Laravel. Розроблений метод допомагає генерувати код для модульного тестування Laravel застосунку шляхом застосування генетичного алгоритму. Даний метод допомагає пришвидшити процес тестування ПЗ роблячи його не ресурсозатратним процесом та зручним.

2. Висновок про відповідність роботи дипломному завданню Кваліфікаційна робота освітнього ступеня «магістр» у повній мірі відповідає поставленому завданню як у теоретичній, так і в практичній її частині.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи У вступі обґрунтовується актуальність теми роботи, мета та завдання дослідження, описується наукова новизна та практична цінність отриманих результатів. У першому розділі охарактеризовано структуру предметної області та існуючих методів і засобів забезпечення тестування ПЗ, визначені методологічні підходи до вирішення задачі, виконана розгорнута постановка задачі. У другому розділі досліджено методи і способи вирішення поставлених задач. Описані теоретичні підходи до застосування алгоритмів для генерації тестів, досліджені на етапі аналізу, було обрано алгоритм генетичного програмування, його переосмислення та застосування. У третьому розділі обґрунтовано проектні рішення, що дають змогу реалізувати технічні вимоги. У четвертому розділі розглянуто питання, що стосуються реалізації програмного засобу на основі прийнятих рішень, а також її технічні та технологічні характеристики. Також проведено дослідження, спрямоване на доведення працездатності розробленого програмного засобу та його функціональної придатності. Обґрунтована ефективність удосконаленого методу генерації тестів з пошуком дій для ПЗ та розроблено рекомендації з його застосування.

4. Позитивні сторони роботи Кваліфікаційна робота демонструє інноваційне рішення щодо генерації коду для тестування ПЗ. Запропоновано метод генерації тестів за допомогою алгоритму генетичного програмування з урахуванням вхідних даних та дій для тестування. Даний метод показав свою високу ефективність під час тестування та випробувань.

5. Негативні сторони роботи У даній кваліфікаційній роботі генерація тестів спрямована на обмежений список тестування функціоналу: тестування запитів та веб-форм. Це пов'язано з браком ресурсів та часу на розробку але в майбітньому студент може

розширити список функціоналу для генерації тестів.

6. Оцінка графічного оформлення та пояснювальної записки роботи Графічне оформлення виконане відповідно до теми кваліфікаційної роботи з дотриманням вимог стандартів. Пояснювальна записка відповідає вимогам стандартів до її оформлення.

7. Відгук про роботу в цілому В цілому кваліфікаційна робота заслуговує позитивної оцінки. Весь матеріал роботи структурований, чіткий та послідовний. Усі розділи роботи є послідовними та логічними, що дозволяє чітко розуміти викладений матеріал у рамках тематики кваліфікаційної роботи. Графічний матеріал дозволяє наочно побачити доцільність та ефективність рішень, які були прийняті для вирішення поставленої задачі.

8.Інші зауваження

9. Оцінка кваліфікаційної роботи Розглянувши позитивні та негативні сторони представленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує оцінки «добре».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)

Марченко Валерій Володимирович,
доц. каф. Автоматизації, комп'ютерно-інтегрованих технологій та робототехніки ХНУ

7.12.2023 р.
Дата

(Підпис)