МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет радіоелектроніки
Факультет Комп'ютерних наук
(повна назва)
Кафедра Програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів оптимізації тестів та візуалізації інформації при автоматизованому тестуванні

(тема)

Виконав:
Студент 2 курсу, групи ІПЗм-19-1
Фундукян А.А.
(прізвище, ініціали)

Спеціальність 121– Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми Освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Керівник проф. Четвериков Г.Г.
(посада, прізвище)

Допускається до захисту

Зав. кафедри _____ З.В. Дудар
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

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

Освітня програма Інженерія програмного забезпечення
(повна назва)

ЗАТВЕРДЖУЮ:
Зав.кафедри _____
(підпис)
« 26 » березня 2021 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента Фундукяна Артура Андрійовича
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів оптимізації тестів та візуалізації інформації при автоматизованому тестуванні
затверджена наказом університету від 26.03.2021 № 386Ст
2. Термін подання роботи до екзаменаційної комісії 15 05 2021р.
3. Вихідні дані до роботи електроні ресурси за обраною тематикою, опублікована стаття, вимоги до додатку що підлягає розробці, набір тестових випадків
4. Перелік питань, що потрібно опрацювати в роботі аналіз предметної галузі, методи оптимізації тестів, визначення пріоритетності виконання тестових випадків, алгоритми вимірювання відстані, візуалізацію інформації, архітектуру додатку, алгоритм використання, аналіз отриманих результатів.
5. Перелік графічного матеріалу із зазначенням креслеників, схем, слайдів, ілюстрацій: титульний слайд, мета роботи, постановка задачі, аналіз предметної галузі, методи оптимізації у автоматизованому тестуванні, визначення пріоритетності, Візуалізація інформації, аналіз предметної галузі, структура додатку, середній процент знайдених помилок, покриття помилок, час виконання, схожість тестових випадків, історія виконання тестових випадків, висновки .

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Інструктаж з безпеки життєдіяльності	25.01.2021 р.	Виконано
2	Ознайомлення з організаційною структурою бази практики	25.01 – 29.01	Виконано
3	Отримання індивідуального завдання	25.01 – 29.01	Виконано
4	Аналіз предметної галузі	01.02 – 12.02	Виконано
5	Постановка задачі	15.02 – 19.02	Виконано
6	Визначення передумов	22.02 – 26.02	Виконано
7	Дослідження практик безперервної інтеграції	01.03 – 12.03	Виконано
8	Дослідження методів виявлення пріоритетів тестів	15.03 – 04.04	Виконано
9	Визначення гіпотези та формування до додатку	05.04 – 14.04	Виконано
10	Розробка додатку	14.04 – 28.04	Виконано
11	Проведення дослідження	28.04 – 10.05	Виконано
12	Рецензування	10.05 – 14.05	Виконано
13	Підготовка докладу	14.05 – 17.05	Виконано
14	Допуск до захисту	17.05	Виконано

Дата видачі завдання 26 березня 2021р.

Студент _____
(підпис)

Керівник роботи _____ проф. Четвериков Г.Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 79 с., 12 рис., 5 табл., 21 джер.

АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, ЄВРИСТИКИ ОПТИМІЗАЦІЇ, ОПТИМІЗАЦІЯ ТЕСТІВ, ПОКРИТТЯ ПОМИЛОК, ПОШУК ПОМИЛОК, СЕРЕДНІЙ ПРОЦЕНТ ЗНАЙДЕНИХ ПОМИЛОК, ТЕСТОВІ ВИПАДКИ, PYTHON, ROBOT FRAMEWOK.

Об’єктом дослідження є проблема тривалого очікування зворотного зв’язку під час виконання автоматизованого тестування у системах постійної інтеграції.

Метою роботи є визначення та розглядання методів оптимізації автоматичного тестування через процеси визначення пріоритетів для виконання тестових випадків.

Методи розробки базуються на таких технологіях, як Python, Robot Framework, та бібліотеках numpy, textdistance, scipy та pyplot

В результаті роботи було досліджено методи оптимізації виконання тестових випадків, а саме визначення пріоритетності виконання на основі різноманітності, проведено аналіз та моделювання предметної області, розроблено програмну реалізацію для дослідження.

AUTOMATED TESTING, AVERAGE PERCENTAGE of FAULT DETECTED, FAULT COVERAGE, FAULT SEARCH, OPTIMIZATION EURISTICS, PYTHON, ROBOT FRAMEWORK, TEST CASES, TESTS OPRIMIZATION.

The object of this study is the problem of the long feedback cycles from CI systems during the automated testing.

This study aims to figure out and select some methods of test optimization for automated testing by use of test case prioritization.

Development methods are based on such technologies as Python, Robot Framework, and some specific technical libraries such as NumPy, textdistance, scipy, and pyplot.

As a result, some methods for test optimization were investigated, specifically the method of test case prioritization based on diversity. Analysis and modelling of the subject area were performed as well as some software for studying was developed.

Я, Фундукян Артур Андрійович, студент групи ІПЗм-19-1, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів оптимізації тестів та візуалізації інформації при автоматизованому тестуванні», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

ПЕРЕЛІК СКОРОЧЕНЬ

APFD	Average percentage of fault detected, середній процент знайдених помилок
CD	Continuous Delivery, безперервна доставка.
CI	Continuous Integration, безперервна інтеграція.
Code	Код тестового випадку.
Jac	Індекс Жакарра.
Lvs	Відстань Левенштейна.
MDS	Multidimensional Scaling, Метод багатовимірного шкалювання
Name	Назва тестового випадку.
Ncd	Нормалізована відстань ущільнення.
Rnd	Random, випадковим чином.
T-sne	T-distributed stochastic neighbor embedding, Стохастичне вкладення сусідів с T-розподілом

ЗМІСТ

ВСТУП 9

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ 11

1.1 Методології розробки та тестування програмних додатків 11

1.2 Тестування програмного забезпечення 14

1.3 Оптимізація тестів 20

1.4 Візуалізація інформації 22

1.5 Постановка задачі 24

2 МЕТОДИ ОПТИМІЗАЦІЇ ТЕСТІВ 26

2.1 Оптимізація тестів 26

2.2 Методи евристичного аналізу 30

2.3 Порівняння методів оптимізації 32

3 ОПТИМІЗАЦІЯ ТЕСТІВ ТА ВІЗУАЛІЗАЦІЯ ІНФОРМАЦІЇ 35

3.1 Пріоритетність тестів на основі різноманітності 35

3.2 Стратегії кодування інформації 36

3.3 Алгоритми вимірювання відстані 38

3.4 Візуалізація отриманої інформації 39

3.5 Зменшення розмірності 40

4 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ 43

4.1 Засоби реалізації 43

4.2 Архітектура додатку 44

4.3 Алгоритм використання 46

4.4 Аналіз отриманих результатів 47

ВИСНОВКИ 59
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 61
ДОДАТОК А 63
ДОДАТОК Б 64
ДОДАТОК В 65
ДОДАТОК Г 66
ДОДАТОК Д 75
ДОДАТОК Е 78

ВСТУП

Безперервна інтеграція широко використовується у сфері розробки програмного забезпечення для об’єднання та автоматизоване тестування змін у загальному коді є важливою частиною цього процесу. Проблемою автоматизованого тестування у великих системах є тривалість очікування зворотного зв’язку, що уповільнює процес розробки [1].

Оскільки велика кількість компаній стикаються з цією проблемою і шукають ефективні методи оптимізації тестів, саме через це пошуки зробити ці процеси більш швидкими є дуже актуальним питанням.

Метою роботи є проведення аналізу галузі тестування, формування гіпотези стосовно здатної вирішити знайдені проблеми, розробка тестового додатку для допомоги перевірки гіпотези та проведення процесу дослідження разом із фінальним представленим результатом дослідження.

Об’єктом дослідження виступають сховища тестів, у системах постійної інтеграції, велика наповненість яких призводить до збільшення часу потрібного для повноцінного проходження усіх етапів автоматизованого тестування. На основі попередніх досліджень можна зробити висновок, що у якості вирішення отриманої проблеми є вдалим використання методів оптимізації тестів, що і є предметом дослідження у магістерській роботі.

Методи дослідження можна поділити на два типи – це теоретичні та практичні. З боку теоретичних методів виступають перевірка проблемної галузі, минулих проведених досліджень, та формування теоретичного процесу перевірки покращення. З боку практичних методів виступає процес розробки відповідного додатку, та його подальше використання у процесі перевірки гіпотези, разом із формуванням результатів.

У результаті було використано метод визначення послідовності виконання тестових випадків на основі різноманітності, з цілю досягнення більш швидкого знаходження помилок, що у свою чергу скорочує час очікування командами

розробки. Також були перевірені різні методи для визначення різноманітності тестових випадків, та отримано досить позитивні результати стосовно використання визначення пріоритетів у тестовому сховищі. Наступними етапами дослідження можуть стати спроби поєднання використаних методів, разом із аналізом останніх внесених змін з метою досягти ще більш позитивного результату.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Методології розробки та тестування програмних додатків

Протягом останніх десятиріч розробка програмних додатків пройшла процес від формування як напряму в цілому, до однієї із найбільш розвинутих та масштабних галузей у світі. На сьогодні майже кожна сфера людської життєдіяльності не обходиться без використання програмного забезпечення у тому чи іншому вигляді. Великий попит та всеосяжність сфери використання призвели до необхідності збільшення ефективності та всілякого вдосконалення процесів поєднаних із розробкою, підтримкою та подальшою експлуатацією програмних додатків.

Процеси та методології розробки програмних додатків також зазнали великих змін за цей період та продовжують свій розвиток кожен день. У сьогоденні методології розробки які стали найбільш популярними, перспективними та широко використовуваними можна вважати Agile та DevOps, в яких одним із найважливіших процесів виступає процес тестування.

Agile – це гнучка методологія розробки програмного забезпечення, основна ідея якої сконцентрована навколо ідеї ітеративної розробки, де вимоги та вирішення складностей розвиваються через співпрацю між крос-функціональними командами [2]. Кінцева цінність Agile-розробки полягає в тому, що вона дозволяє командам швидше показувати результат, з більшою якістю та передбачуваністю, а також більшою схильністю реагувати на зміни.

Основними етапами процесу розробки з використанням методології Agile є процес який складається із процесу збору вимог, виконанню процесу проектування додатку, розробки додатку, його тестування, та розгортання. Цей процес зазначено на рис. 1.1.

Перевагами використання Agile є:

– висока якість продукту;
– задоволеність замовника;
– кращий контроль;
– покращене планування;
– зменшення ризиків.

Як можна зрозуміти з рис. 1.1, процес тестування є невід’ємною частиною у кожному циклі. До особливостей тестування у цій методології можна зазначити те, що процес тестування відбувається увесь час безперервно, що призводить до швидкого знаходження та виправлення недоліків. Завдяки постійному тестуванню забезпечується безперервний зворотній зв'язок із командою розробки. Тестування проводиться не тільки спеціалізованими тестувальниками, а й усіма іншими членами причетними до розробки.

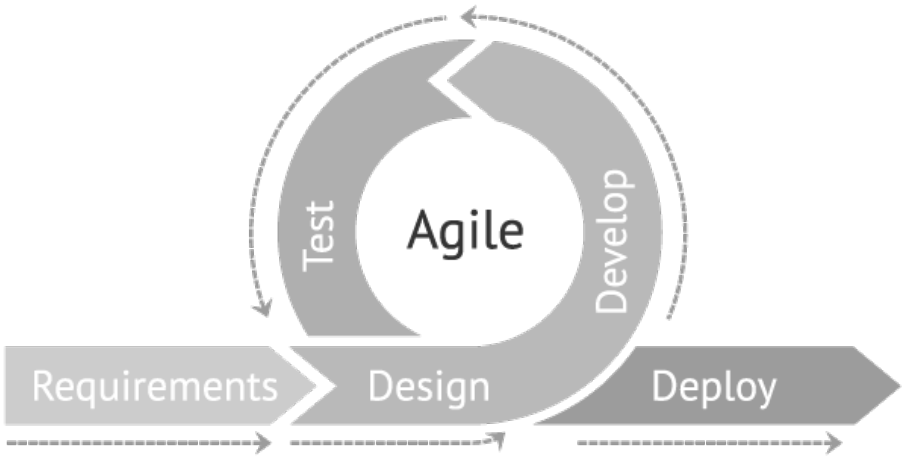

Рисунок 1.1 – Схема процесу Agile

DevOps – практика яка полягає у тому, що розробники та фахівці інформаційно-технологічного обслуговування працюють разом на протязі усього циклу розробки додатку, починаючи із проектування, продовжуючи у процесі розробки і закінчуючи процесом підтримки [3]. Основною метою процесів DevOps є автоматизація та інтеграція процесів між різними командами розробки таким чином, що розробка, побудова, тестування та випуск програмного забезпечення

швидше та більш надійно. Безперервний характер DevOps показує як фази життєвого циклу пов’язані один з одним. Не дивлячись на те, що цикл протікає послідовно, петля символізує необхідність постійної співпраці та ітераційного вдосконалення протягом усього життєвого циклу. Цикл взаємодії відображено на рис. 1.2.

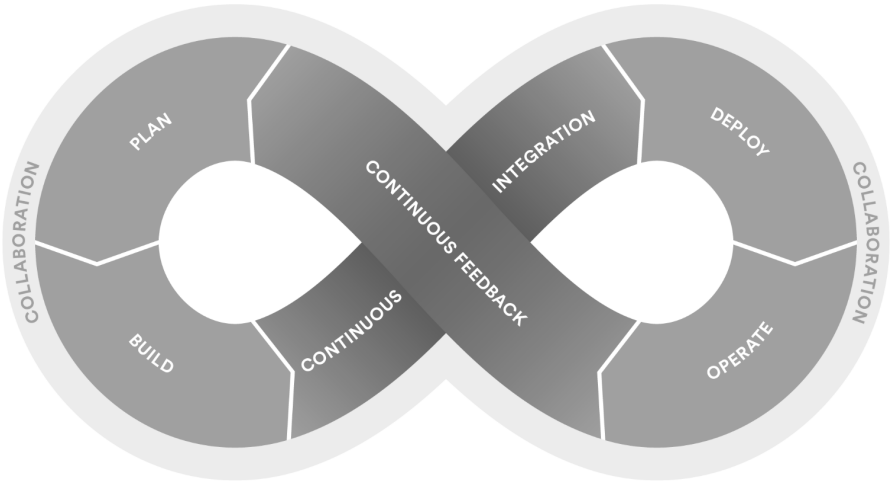

Рисунок 1.2 – Схема процесу DevOps

Життєвий цикл DevOps складається із 6 основних фаз, які відображають основні процеси, а саме планування, побудову, безперервну інтеграцію, доставка, використання, та безперервний зворотній зв’язок.

Але не дивлячись на тип методології розробки програмного забезпечення, усі вони виділяють важливість процесів тестування, оскільки саме цей процес призводить дотримання фінального та завершеного вигляду додатку, з та яким приємно користуватися. Цікавою особливістю більшості сучасних методології розробки, якщо не усіх, є першочергове використання тестування, з дуже великим уклоном у сторону автоматизованого та безперервного тестування.

Безперервне тестування є процесом використання автоматизованого тестування у якості одного із основних типів тестування для постійної перевірки додатку, внесених змін, та отримання швидкого зворотного зв’язку, разом із перевіркою бізнес ризиків. Безперервне тестування спочатку було запропоновано

для того щоб зменшити час очікування зворотного зв’язку через використання тестів що які автоматично перевіряються системою.

1.2 Тестування програмного забезпечення

Як було зазначено раніше, процес тестування є невід’ємною частиною сучасних методології розробки програмних додатків. Цей процес є критично важливим для результатів розробки, перевірки відповідності продукту до системних та бізнес вимог. Саме тому тестування буде більш детально розглянуте у цьому розділі.

1.2.1 Тестування

Тестування програмного забезпечення – це процес, або ряд процесів, призначений для того, щоб переконатися, що комп'ютерний код робить те, для чого він був розроблений, і, навпаки, що він не робить нічого ненавмисного. Програмне забезпечення повинно бути передбачуваним і узгодженим, не представляючи ніяких сюрпризів для користувачів [4].

Під час етапу тестування повинні бути виявлені та виправлені помилки які були допущені під час дизайну та програмування. Помилками програмування можуть бути сценарії використання додатку під час яких відбувається непередбачене припинення його робото здатності або відбувається невідповідність виконання відносно розробленої архітектури. Помилками під час програмування також можуть бути проблеми пов’язані з безпеко або зручністю використання. Помилки програми також полягають у проблемах безпеки або зручності використання. Приладом проблем пов’язаних з безпекою може бути вразливість до

атак з боку зловмисників, яка призводить до доступу до приватних даних. Помилки процесу проектування зазвичай полягають у неузгодженості між тим чого хоче отримати власник проекту, та тим, що реалізується командою розробки. Цей тип помилок виникає на етапі планування та має дуже великий вплив на проект. Не менш важливим є те, що зазвичай помилки проектування дуже складно виправити [5]. Досить часто розробники приділяють занадто багато уваги візуальній складовій, та забувають про відмінності систем користувачів [6].

Тестування є важливою частиною забезпечення якості, та перевірки виконання умов стосовно вимог та бізнес-потреб програмного додатку. Автоматизоване тестування є невід'ємною частиною безперервної інтеграції, де тести виконуються після будь яких зміни в основному коді, щоб гарантувати, що інтеграція різних частин програмного забезпечення працює, а програмне забезпечення залишається якісним, та дійсним. Системи безперервної інтеграції зазвичай також мають як систему контролю версій, так і систему збірки додатку, разом компілюють код і запускають тести. Результатом тестів є зворотний зв'язок для розробників, які використовують його для прийняття рішення про наступний крок у циклі розробки програмного забезпечення, а також знання, працює рішення проблеми чи ні.

1.2.2 Важливість тестування

Тестування є дуже важливим етапом розробки програмних додатків, оскільки відсутність цього процесу може призвести до цілої низки різноманітних проблем. Розглянемо що саме привносить тестування, та від яких можливих проблем перестерігає.

Першим, та мабуть найбільш очевидним результатом правильного тестування є якість додатку що розробляється. Для того щоб продукт мав позитивний вплив на сферу та користувачів, та відповідав всім бізнес-цілям він

повинен працювати як було заплановано. Дотримання вимог, та відповідність заявленій цінності є одними із найважливіших критеріїв оцінки з боку кінцевих користувачів.

Другим пунктом можна зазначити безпеку. Безпека інформації стає все більш важливим фактором який потребує найвищого ступеня уваги, оскільки програмні системи оточують майже кожен процес людського життя та несуть відповідальність за величезну кількість важливих процесів. Особиста інформація користувачів та збереження таємності цих даних мають бути у пріоритеті у компаній які займаються розробкою програмних додатків і процеси тестування якості захисту цієї інформації мають бути відповідними.

Третім пунктом є рівень задоволеності користувачів. Мабуть однією із важливіших цілей є підтримка високого рівня задоволеності користувачів програмними додатками, оскільки вони завжди потребують найвищого рівня користувацького досвіду. Цей показний напряму корелює із лояльністю клієнтів, рейтингом та статусом компанії розробника.

Четвертим, але не менш важливим є економія коштів. Тестування допомагає зберегти кошти у довгостроковій перспективі. Особливо явно збереження коштів відбувається у тих випадках, коли недолік було знайдено на більш ранніх етапах розробки, якомога швидше. На фінальних етапах, або під час експлуатації додатку наявність недоліків та вартість їх виправлення може стати колосальною. Потрібно не забувати про той факт що значимість помилки дуже сильно варіюється у залежності від типу системи. Наприклад у банківських, медичних або автомобільних системах вартість помилки може впливати на великі кошти або навіть стан людей залежних від якості праці системи.

П’ятим, але ніяк не останнім, чинником до якого призводить тестування є прискорення розробки. Тестування та зокрема тести надають розробникам можливість полегшити процес пошуку помилок та сценаріїв які можуть їх виявити, що у свою чергу надає можливість прискорити їх виправлення. Більш того, цей чинник має накопичуваний результат, це означає що в разі наступного відтворення

схожої помилки, інформація про можливу проблему вже є як у тестувальників, так і у розробників, що істотно зменшує кількість часу необхідного на її вирішення.

1.2.3 Типи тестів

Існує велика кількісті різноманітних типів тестів які використовуються під час тестування програмних додатків і кожен з них має свою певну мету. Серед типів тестів хочу виділити модульне тестування, регресійне тестування, та інтеграційне тестування.

Модульне тестування – це тип тестування який у якості цілі має перевірку окремих методів або функцій класів, компонентів або модулів програмного додатку. Зазвичай вони повинні бути розроблені для окремих . Зазвичай вони не ж дуже складними, оскільки покривають окрему частину коду, є досить не складними у процесах автоматизації тестування. Результатом модульного тестування є перевірка окремого функціоналу та повернення результату, що окремі модулі програми працюють як було задумано, що надає впевненості у суцільній правильності виконання. Цей тип тестів широко використовується у тестуванні у процесі безперервної інтеграції.

Регресійне тестування – це тип тестування який за ціль має перевірити працездатність додатку або його окремих частин які вже були розроблені та пройшли процес тестування. Ці повторні перевірки зазвичай стають у нагоді після додавання нового, або змінення старого функціоналу, оскільки будь-які зміни можуть призвести до появи нових дефектів.

Інтеграційне тестування – це тип тестування який має ціль відповісти на запитання чи дійсно різні модулі або сервіси програмного додатку працюють вдало як єдина одиниця. У свою чергу зазвичай інтеграційне тестування відбувається після тестування та перед валідаційним тестуванням. Цей тип тестування

використовує більше ресурсів системи тестування оскільки у продовж його виконання потрібно щоб різні частини додатку працювали водночас.

Зазначені вище типи тестів дуже часто стають основоположними з точки зору використання у CI процесах, та саме автоматизованому тестуванні. Ці типи тестів схильні до процесу автоматизації, зберігаючи високий рівень якості перевірки системи що потребує тестування. Вони також є досить зручними, оскільки можуть повертати результати в простому вигляді – позитивне або негативне проходження.

1.2.4 CI / CD та автоматизоване тестування

Безперервна інтеграція – це практика яка базується на постійній збірці програмного забезпечення, виконанні автоматизованих тестів та перевірок, розгортанні програмного забезпечення та опрацюванні зворотного зв'язку (результатів). Ця практика намагається відповісти на запитання, чи дотримуються стандарти кодування, яке покриття коду тестами, тощо. Слід також зазначити, що основним принципом безперервної інтеграції є побудова та перевірка програмного забезпечення після кожної внесеної зміни.

Однією дуже розповсюдженою проблемою під час розробки програмного забезпечення є припущення [7]. Наприклад, припускаючи, що розробники дотримуються стандартів кодування та дизайну, велика вірогідність що отриманим програмним забезпеченням буде важко керувати. Кожне припущення збільшує ризики проекту. Однак багато припущень можуть стати фактами, якщо використовувати безперервну інтеграцію [7]. Наведений вище приклад припущення врегулюється за допомогою CI з автоматизованою інспекцією програмного коду.

Пол М. Дюваль зі співавторами [7] вважають що CI має велике значення для зменшення ризиків та необхідності використання повторюваних ручних процесів. У результаті використання безперервної інтеграції стає можливим створення

програмного забезпечення, яке можна розгорнути де завгодно та в будь-який час, що в свою чергу надає команді розробників більшої впевненості у програмному забезпеченні.

Існує безліч різноманітних механік безперервної інтеграції, наприклад безперервна інтеграція баз даних, тестування, перевірки коду, розгортання та надання зворотного зв'язку [7]. Також дуже корисно постійно оновлювати інформацію у базі даних. Це оновлення призводить до того, що програмне забезпечення та база даних залишаються у синхронізованому вигляді увесь час. Постійні перевірки дозволяють командам не тільки зменшити складність коду та позбутися дублювання, а також дотримуватися стандартів [7]. При кожній інтеграції код перевіряється сервером збірки, щоб оцінити, наскільки він відповідає заданим правилам.

Безперервна доставка (Continuous Delivery) – це ще одна практика, і її найзручніше описати як безперервну інтеграцію для розгортання програмного забезпечення на виробничих системах. Необхідність безперервної доставки замість ручного процесу мотивують тим, що завдяки цьому стає можливим частий випуск різноманітних версій програмного забезпечення, навіть із невеликими змінами.

Автоматизоване тестування – це практика автоматичного запуску тестів для кожної зміни в системі контроля версій. Це важливий аспект безперервної інтеграції, оскільки дозволяє розробникам оцінювати програмне забезпечення [8]. Однак, оскільки у більшості випадків слід дочекатися завершення тестів, перш ніж починати інше завдання, час зворотного зв'язку стає важливим. Якщо автоматизовані тести займають багато часу, розробка уповільнюється, розробники чекають, поки зміни які вони зробили пройдуть перевірку у системі контроля версій. Зі збільшенням розмірів проекту, можливість обирати пріоритети стає більш важливою.

Автоматизоване тестування широко використовується у різних сферах іт. індустрії і у випадках, коли сховища тестів невеликого розміру, час усього виконання також є досить прийнятним. Однак це не стосується компаній з більш об'ємними сховищами тестів, оскільки через тривалість процесів виконання – час

очікування результатів зворотного зв’язку може стати занадто довгим, щоб залишатися корисним для розробників. Причиною цього є те, що час, необхідний для обробки всього тестового сховища, збільшується у пропорції із збільшенням розміру сховища. У деяких ситуаціях, у якості рішення проблеми, компанії можуть збільшити ресурси що витрачаються на тестування, щоб компенсувати довші цикли зворотного зв’язку. Однак це лише короткострокове, та не дуже ефективне рішення, тому довші цикли зворотного зв'язку можуть рано чи пізно повернутися. Більш того, у деяких випадках збільшення ресурсів що використовуються під час тестування не призводить до покращення, оскільки існують задачі, які потребують довгострокового очікування, та не залежать від обчислюваної потужності.

Також потрібно ввести позначення сховища тестів. Сховище тестів – це місце, де зберігаються всі тести, пов’язані з проектом. Сховища тестів часто розташовані в системах контролю версій. Ведення сховища тестів означає створення, видалення та зміну тестових випадків і відіграє важливу роль у тестових циклах. Під тестовими циклами мають на увазі кроки, вжиті для визначення тестів, налаштування та обслуговування тестових середовищ, виконання тестів та аналізу результатів виконання, включаючи підтримку самого сховища тестів. Цей процес повторюється до тих пір, поки не буде проведено випробування на завершенні проекту. Зв’язаний термін – це цикли зворотного зв’язку, що стосується часу, що минув з моменту запуску тесту, до отримання результатів виконання розробником.

1.3 Оптимізація тестів

Оптимізація тестів може стати важливою частиною процесів безперервної інтеграції оскільки як було зазначено раніше, кількість часу потрібного на проведення автоматизованого тестування може уповільнювати процес розробки. У глобальному розумінні – оптимізація тестів є процесом впровадження змін у

процес тестування таким чином, щоб зробити його менш затратним з точки зору часу та коштів без втрат с точки зору якості результатів.

Насамперед самі по собі тести повинні бути розроблені оптимально. Під оптимально розробленими тестами можна вважати ретельно продумане сховище тестів, тести з якого відповідають деяким характеристикам.

Робити тести невеликими та атомарними. Невеликі тести перевіряються швидше ніж їх масштабні версії. Під атомарністю мається на увазі перевірка окремих, маленьких, незалежних частин. Якщо великий тест можна поділити на декілька маленьких частин без втрати його властивостей – це потрібно зробити. Оскільки зазвичай після розділення маленькі тести стають залежними один від одного – у разі не проходження першого, перевірка усіх інших стає не потрібною, що у свою чергу зменшить кількість перевірених тестів та тривалість усього процесу.

Покривати важливий функціонал. Насамперед покриття тестами повинно бути зроблено для найбільш важливих або складних у перевірці частин поетку. Це призводить до збільшення вірогідності знайдення та швидкого виправлення проблем у найбільш важливих частинах проекту.

Не дублювати тести. Дублювання тестів стає у заваді у швидкості перевірки усього тестового сховища, але не надає ніякої суттєвої вигоди оскільки не привносить нової інформації.

Намагатися не використовувати очікування. Очікування, наприклад затримка на n-секунд, у тестах не є позитивним моментом. Це затримує як виконання самого тесту, так і виконання усього тестування в цілому, тому у випадках коли цей процес можна якось замінити на інший – це потрібно робити.

Але навіть за наявністю максимально правильно розроблених тестів, лише через їх кількість тривалість виконання може стати занадто великою. У таких випадках стає необхідним використання альтернативних методів оптимізації процесу тестування.

Було проведено багато досліджень різних методів оптимізації тестів, але їх можна поділити на три основних типи:

– вибір окремих наборів тестових випадків;

– встановлення пріоритетів тестових випадків;

– скорочення набору тестів.

Ці методи будуть більш детально розглянуті у наступному розділі цієї роботи.

1.4 Візуалізація інформації

Візуалізація інформації – це окрема практика яка полягає у подачі якихось абстрактних даних, будь то чисельні, або текстові, у зручному для сприйняття форматі. Візуалізація інформації дозволяє більш детально відстежувати процес інтеграції проектних рішень в рамках розробки сучасного програмного забезпечення [9]. Зазвичай для візуалізації користуються графіками, гістограмами, тощо. Цікавою особливістю автоматизованого тестування є те, що окрім самого процесу тестування накопичується додаткова інформація яка зазвичай є поза зоною зору користувачів.

Прикладом такої інформації може стати історія виконання тестів, розподілення тестів за ознаками тощо. Ця інформація може бути інтерпретована та використана, що дасть можливість робити більш розгорнутий аналіз для подальшого прийняття рішень щодо тестування. Більше того, ця інформація може бути використана для виявлення закономірностей у виконанні тестів, що найчастіше призводить до цікавих відкриттів стосовно системи під час тестування, або навіть самих процесів розробки.

Нажаль, зазвичай не уся інформація одразу готова до використовування, насамперед через її кількість і багатомірність. Існує вирішення зазначеної вище проблеми, яке полягає у використанні зменшення обсягу інформації для

візуалізації різноманітності різних тестових випадків із сховища тестів таким чином, щоб інтерпретувати їх для тестувальників. Результати зменшення розмірності можуть бути використані для візуалізації інформації про сховище тестів, що стає цінним для розробників та тестувальників у їхніх зусиллях підтримувати та покращувати його якість. Потенціал використання інформації про різноманітність представленим тут способом полягає у тому, що загальна вартість або складність оптимізації різноманітного тестування може стати більш керованою завдяки додатковій інформації, що надається тестувальникам, розробникам та іншим зацікавленим користувачам. Це також призводить до збільшення результатів від інвестицій у автоматизоване тестування.

Прикладом графічного представлення результатів може стати теплова карта, зображена на рис. 1.3. Теплові карти є представленням інформації у графічному вигляді з використанням різноманітних кольорів для позначення певної характеристики, або відмінності між характеристиками.

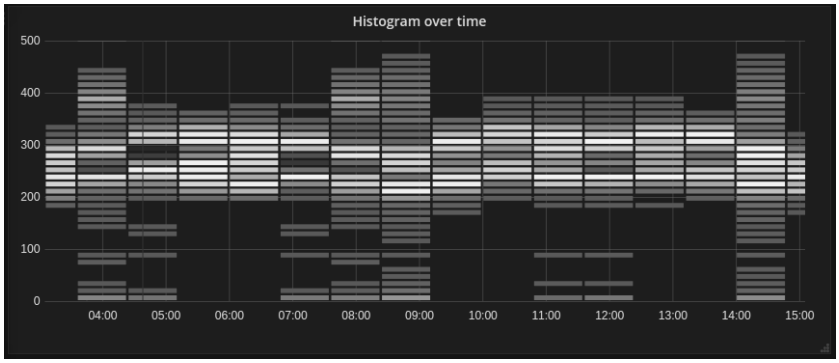

Рисунок 1.3 – Теплова карта

Теплові карти мають досить широкий спектр сфер використання, але у цьому випадку вона може використовуватися для відображення тестів та результатів їх проходження протягом певного часу. Результати графічної Закономірності які відображаються на такому графіку можуть вказати на найбільш схильну до помилок частину проекту, або іншу цікаву статистичну інформацію.

Іншим прикладом графіку із статистичної інформації може стати діаграма розсіювання, приклад зображено на рис. 1.4. Цей тип діаграми також

використовується для відображення певних даних у графічному вигляді, але користується для цього набором точок, кожна з яких має своє власне значення. Таким чином можна переглядати залежності між даними.

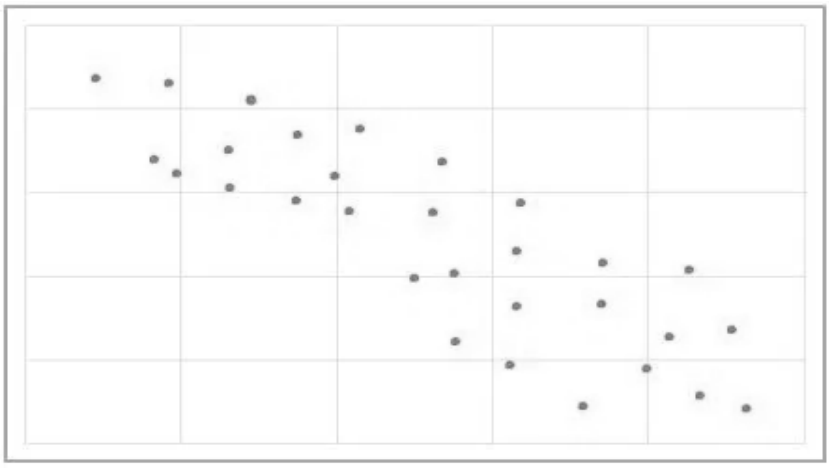

Рисунок 1.4 – Діаграма розсіювання

Цей тип графіку може відображати рівень подібності або відмінності тестових видиків за якоюсь окремою ознако, тощо.

1.5 Постановка задачі

На основі проаналізованої проблемної галузі та систем безперервної інтеграції було складено ряд задач які потрібно розглянути під час виконання дослідження. Основоположним питанням яке розглядається є процеси тестування у системах безперервної інтеграції. Основним виділеним недоліком є грандіозне збільшення часу очікування завершення автоматичного тестування у разі наявності великих сховищ тестів. Це у свою чергу призводить до уповільнення процесів розробки та тестування, оскільки для того перевірка змін наявними тестами є першочерговим закладом у автоматизованому тестуванні.

Для вирішення зазначеної проблеми у процесах безперервної інтеграції, та зокрема автоматизованого тестування потрібно розглянути варіанти оптимізації процесів тестування.

Спираючись на зазначені вище проблеми у процесах безперервної інтеграції, основними задачами є:

– визначити які типи інформації можна використовувати для оптимізації тестів;

– розглянути та обрати методи оптимізації процесів автоматизованого тестування;

– виявити інформацію стосовно сховища тестів та результатів виконання тестів, яка може стати в нагоді тестувальникам;

– запропонувати метод візуалізації додаткової інформації.

Таким чином у першому розділі атестаційної роботи було проведено аналіз предметної галузі з якого було здобуто інформацію стосовно сучасних процесів розробки та тестування програмних додатків, виділено проблему у процесі автоматизованого тестування, та складено основні задачі для подальшого дослідження.

2 МЕТОДИ ОПТИМІЗАЦІЇ ТЕСТІВ

2.1 Оптимізація тестів

Як було зазначено раніше, у випадках коли сховища тестів мають великий об’єм інформації час потрібний на виконання усіх тестів зростає і може стати занадто тривалим. Існує декілька різних підходів для вирішення цієї проблеми, основними з яких є вибір окремих наборів тестових випадків, встановлення пріоритетів тестових випадків, скорочення набору тестів.

Методи оптимізації можуть бути використані, та у свою чергу використовувати при тестуванні як методами чорної, так і білої скриньки. Якщо тести проводяться за допомогою методу чорної скриньки, оптимізація тестів не залежить від вихідного коду програмного забезпечення, прикладом такого методу може бути статичний аналіз. У свою чергу, використання методів білої скриньки означає, що алгоритми оптимізації тестів використовують інформацію, доступну під час виконання програмного забезпечення що тестується. Інформація що використовується у методі чорної скриньки – вимоги до програмного забезпечення, назви тестів, код тестів, результати виконання тестів, та виявлення несправностей. Подібними прикладами інформації для методу білої скриньки є охоплення вихідного коду та охоплення функцій або методів.

2.1.1 Вибір тестових випадків

Вибір тестових випадків – це один із підходів до оптимізації тестів, який полягає у виборі підмножини тестових випадків відповідно до певного критерію [10].

Прикладом критерію може стати використання інформації стосовно останніх змін, що проходять тестування. Таким чином цей підхід є ключовим при розробці

тестових стратегій які полягають у зменшенні кількості, виключенні зайвих тестів або тестової інформації, але у той самий час із спробою зробити це без небезпеки до виявлення помилок.

У той час як метод скорочення наборів тестів є остаточним і не залежить від змін або поточного стану системи що тестується, вибір тестових випадків навпаки приділяє більш уваги до стану та змін у системі. Більш того, оскільки цей підхід базується на пошуку найбільш відповідних тестів відносно змін, який відбувається кожного разу, це не призводить до зменшення сховища тестів та їх варіативності, але все одно надає змогу зменшити кількість тестів які перевіряються кожного разу. Процес аналізу змін та пошуку відповідності зазвичай потребує статичного аналізу коду методів білої скриньки.

Цей підхід є ефективним з точки зору вибору тестів які найбільш вдало підходять за певними критеріями відносно останніх змін у програмному додатку, але його мінусом є те що через вибір тестів на основі останніх змін які було зроблено нові тести які було додано до сховища тестів можуть бути не використані у перевірках.

2.1.2 Скорочення наборів тестів

Техніка скорочення наборів тестів переслідує ціль скоротити набори тестів через вилучення зайвих, не важливих, або схожих за змістом тестів. Ці перетворення, а саме вилучення, відбуваються один раз, тобто вилучення тесту із набору є остаточним.

Цей тип оптимізації бере за увагу той факт, що не усі тести у наборах тестів дійсно потрібні для того щоб впевнитися що система працює як було заплановано, тому є можливість вилучити тести які дублюють перевірки, або якимось іншим чином є не найбільш затребуваними. Іншими словами, якщо для перевірки системи існує деякий достатній об’єм тестування, який може бути досягнутий шляхом

перевірки будь яким окремим, чи мінімальною кількість тестів із набору тестів або тестового сховища, то кількість тестів повинна бути скорочена до цієї необхідної кількості. Отже підхід можна представити у якості скорочення набору тестів до найменшої кількості достатньої для тестування певного функціонала.

Слід зазначити, що скорочення до найменшої кількості базується на припущенні, що кожна вимога до додатку може бути перевірена єдиним тестовим сценарієм. Зазвичай на практиці – це не може бути правдою. Наприклад, якщо вимога є функціональною, вона зазвичай потребує більш ніж одного тестового сценарію для того щоб бути впевненим у тому, що усе працює як було заплановано. Для досягнення цілі із наявності одного тестового сценарію для вимоги, може стати потрібним внесення відповідних корегувань тестових сценаріїв. Процес корегування може бути як у вигляді збільшення рівня абстракції, для того щоб єдиний тестовий сценарій міг покривати єдину вимогу, або вимога може бути розділена на менші під-вимоги, які у свою чергу можна перевірити єдиним тестовим випадком.

Однією з відомих евристик, запропонованих для скорочення на основі коду, є Жадібний алгоритм. Цей алгоритм також може бути застосований до тестових наборів, отриманих за допомогою модельних методів. Він вибирає тестовий випадок, який задовольняє максимальній кількості незадоволених вимог, і довільний вибір робиться, якщо є ситуація, пов’язана з нічиєю. Цей процес застосовується неодноразово до всіх тестових випадків у наборі тестів і дає зменшений набір тестів. Його зупиняють після того, як задовольняються всі вимоги до тесту.

Цей підхід є ефективним з точки зору зменшення розмірів тестових випадків, та пришвидшення процесу тестування. У якості його негативної характеристики є те, що у разі неякісного вилучення тестів існує вірогідність втратити ефективні тести. Також у більшості випадків цей метод використовують без уваги до змін які було зроблено до системи що тестується, або сховища тестів, що у свою чергу призводить до можливості пропустити значиме нововведення.

2.1.3 Встановлення пріоритетів тестових випадків

Встановлення пріоритетів тестових випадків – це ще одна техніка оптимізації тестів, яка полягає у виявлені та призначені певної послідовності виконання тестів. Вона надає змогу встановлювати послідовність для виконання тестів, наприклад виконання найбільш важливих або складних відносно якогось параметру тестів першочергово, та отримувати певний результат. Ціль такого визначення пріоритетів полягає у виявлені несправностей раніше та надання зворотного зв'язку з командами розробки та тестування. Статистичні результати цієї техніки можуть також стати у нагоді під час пошуку найбільш оптимальної послідовності тестів.

Метою методів визначення пріоритетів тестових випадків є сортування тестових випадків у порядку, який максимізує або мінімізує цільову функцію, наприклад швидкість виявлення несправності. Суттєвою перевагою цієї техніки є те, що якщо виконання тесту переривається (наприклад, через обмеження часу очікування або обмежену доступність обладнання), розробники та тестувальники можуть бути впевнені, що найважливіші тестові випадки були виконані першими.

Серед відомих технік визначення пріоритетності можна зазначити визначення пріоритетності на основі покриття коду, на основі ризиків, на основі вимог, та інші.

Метрика покриття – це показник який часто використовується у якості критерію визначення пріоритетів. Суть полягає у тому, що рання максимізація структурного покриття коду збільшить імовірність більш раннього виявлення несправності. Таким чином хоча метою визначення пріоритетів тестових випадків залишається досягнення більш високого рівня виявлення несправності, насправді він спрямований на пришвидшення перевірки максимального охоплення коду.

Використання аналізу ризиків також використовується для визначення пріоритетів. Спочатку визначаються потенційно проблемні області, які у разі наявності помилок можуть призвести до максимально негативних результатів.

Після цього тести проходять визначення пріоритетів маючи на увазі потенційно проблемні області. На основі результатів отриманих у процесі аналізу ризиків – відбувається побудова послідовності тестування.

Використання вимог до додатку у процесі визначення пріоритетів також є дуже цікавим варіантом. Ця методика заснована на припущені, що деякі вимоги мають більш критичний характер, а саме тому мають бути перевірені першочергово. У визначенні важливості вимог можуть бути використані такі фактори, визначення важливості власником продукту, визначення командою розробки на основі складності функціоналу, покриття функціоналу який змінюється частіше за все, покриття функціоналу у якому вже колись було допущено помилки.

Встановлення пріоритетів є ефективним з тієї точки зору, що у випадках якщо не використовується обмежена кількість тестів для тестування – усі тестові сценарії, нові чи старі будуть використані під час перевірок, що автоматично робить вірогідність знайдення помилки максимальною, у рамках використання існуючого набору тестів. Алгоритм визначення пріоритетів у свою чергу збільшує вірогідність швидкого зворотного зв’язку. Мінусом є те, що у такому випадку використовуються усі тестові випадки із сховища тестів і час виконання усього тестового запуску не зменшується відносно класичного методу.

2.2 Методи евристичного аналізу

2.2.1 Метод адаптивного випадкового тестування

Метод адаптивного випадкового тестування – є оптимізованим варіантом випадкового тестування. Основна теорія випадкового тестування ґрунтується на тому, що більш широке але рівномірне використання тестових випадків із різноманітних частин додатку призводить до збільшення ефективності пошуку помилок. Основне припущення цього методу полягає у тому, що якщо при

перевірці тестових випадків із групи «А» помилки не було виявлено, то більш вірогідно що перевірки іншого регіону додатку, який віддалений від того що тестується, може їх виявити.

Ця теорія була перевірена у декількох експериментах, наприклад [11]. На основі проведених досліджень, більш ефективним цей метод виявився у випадку перевірки значень які використовувалися під час тестування окремих методів. Основними результатами перевірок є те, що зазвичай негативні результати проходження тесту мають діапазони параметрів що використовуються, та найбільш вірогідно, що у разі негативного результату з використанням параметру n, використання наближених до нього значень також призведе до негативного проходження тесту і навпаки, у разі позитивного проходження тесту зі використанням параметру n, існує істотна вірогідність що наближений до нього діапазон також не призведе до знаходження помилки. Цей метод також інколи використовується під час вибору тестів для подальшого використання. Іншими словами, зазвичай тести, які перевіряють один функціонал будуть мати однаковий результат виконання. На основі цього припущення, потрібно розраховувати різницю між тестами, для того щоб надалі обрати лише різноманітні тести [12]. Негативною рисою цього методу є те, що у реальних системах існує загроза пропустити тест через таке припущення.

2.2.2 Методи засновані на пошуку

Вибір тестового випадку можна розглядати як проблему оптимізації, в якій пошукові методи досліджують простір потенційних рішень з метою знайти найкращі комбінації результатів за розумну обчислювальну вартість. Одним типом із цих методів є генетичні алгоритми.

Генетичний алгоритм – це тип алгоритму, який використовується для з'ясування прогресивних наближень і визначається чотирма ключовими

елементами: популяцією, виділенням, перетином та мутацією. Популяція – це сукупність елементів, що генеруються в кожній взаємодії; селекція – це критерій, який використовується для створення нової сукупності в кожній взаємодії; схрещування – це дія комбінування елементів поточної сукупності з метою генерування нової популяції та мутації випадково мінливих елементів сукупності задля створення нових. Ключовим моментом генетичного алгоритму є визначення функції пристосованості. Вибір випадків тестування, як правило, особини (або хромосоми) виражаються набором даних або послідовністю даних розміром n, в яких функція пристосованості повинна бути використана щодо критеріїв тесту, щоб оцінити, наскільки набір даних наближує відповідність [13].

2.2.3 Методи виділення кластерів

Методи виявлення кластерів використовують критерії для групування тестових випадків відповідно до їхніх характеристик, мінімізуючи внутрішньо-групову дисперсію та максимізуючи між-групову дисперсію [14]. Використання виділення кластерів є вигідним, оскільки класифікація та отримання зведених груп даних може збільшити впевненість щодо спостережуваних множин. Незалежно від використовуваного алгоритму, більшість методів виділення кластерів застосовують заходи апроксимації або дистанції для визначення.

2.3 Порівняння методів оптимізації

Проведений аналіз методів оптимізації тестів призводить до важливого запитання, яке можна викласти як потрібність зрозуміти, як саме порівнюються розглянуті методи один до одного, що можна вважати сильними та слабкими

сторонами кожного з них. Табл. 1 відображає загальне порівняння методів оптимізації тестів.

Таблиця 1 – Порівняння типів автоматизації

Компонент	Тип автоматизації		
	Вибір тестових випадків	Скорочення наборів тестів	Встановлення пріоритетів
Стратегія	Вибір тестів на основі останніх внесених змін.	Скорочення набору тестів до найменшої кількості достатньої для тестування певного функціонала	Виявлення та призначеня певної послідовності виконання тестів.
Сильні сторони	Залежність від внесених змін.	– зменшує кількість тестів у наборі тестів.	– визначення послідовності виконання тестів; – використання усіх тестів, навіть нещодавно доданих.
Слабкі сторони	– нові тести можуть бути не використані; – важливий тест може бути виключений із списку.	– важливий тест може бути виключений із списку.	– може займати стільки ж часу, як і не упорядковані тести.

Скорочення набору тестів – зменшує кількість тестів у наборі тестів, цей процес відбувається поступово разом із збільшенням кількість тестів та тестових

наборів. Цей процес досить схожий із вибором тестових випадків, але у свою чергу вибір тестових випадків повинен проходити кожного разу заново, обираючи тести які будуть перевірену у цей конкретний раз на основі оцінки змін які були зроблені у системі що тестується. Мінусом обох методів може стати те, що тест який є важливим у цілому або саме цього разу може бути виключеним із тестової вибірки, що призведе до втрати інформації стосовно помилки. У разі визначення пріоритетності тестів кожен тест, разом із зміненими або щойно доданими до тестового сховища буде перевірений, та використаний у процесі визначення пріоритетності. Це є дуже важливим фактором, оскільки існує вірогідність що нові тестові випадки знайдуть відхилення у роботі системи що тестується, у відмінності від методів визначення пріоритетності виконання. Визначення пріоритетів виконання тестів є ще одним варіантом оптимізації процесу тестування і базується на призначені пріоритетності виконання для тестових випадків. Плюсом визначення пріоритетності є можливість прискорити пошук помилки у додатку завдяки сортуванню, що є остаточною задачею, мінусом є те, що у випадку якщо помилки відсутні, увесь цикл перевірок буде займати стільки ж часу як і у разі тестування без виявлення пріоритетів.

Методи вибору тестових випадків, та скорочення наборів тестів можуть суттєво зменшити час перевірки відносно використання усього сховища тестів, але на відміну від визначення пріоритетності виконання можуть виключати важливі тестові випадки, що призводить до з’явлення вірогідності пропустити помилку.

У результаті порівняння методів можна зробити висновок, що найбільш безпечним у використанні є метод визначення пріоритетності виконання тестових випадків, оскільки він не виключає тести, а лише призначає їм послідовність виконання з використанням певної ознаки, що може призвести до пришвидшення знаходження помилок. Саме цей метод буде більш докладно розглянуто у наступному розділі.

3 ОПТИМІЗАЦІЯ ТЕСТІВ ТА ВІЗУАЛІЗАЦІЯ ІНФОРМАЦІЇ

3.1 Пріоритетність тестів на основі різноманітності

Існує припущення, що різноманітні набори тестів краще покривають варіативні проблеми, ніж набори тестів, що містять дуже схожі тестові випадки. Різноманітність може виражатися різними способами, наприклад, різноманітність охоплення вимог та різноманітність досвіду серед розробників. У рамках визначення пріоритетності, різноманітність можна розглянути як різницю між тестами за якимось даними або тестовими ознаками. Ця різниця може бути відображена як відстань між тестами. Відбір тестових випадків, що базується на різноманітності використовує оцінку усіх тестів з метою відібрати та визначити пріоритетність наборів тестів.

Метою алгоритму визначення пріоритетності є виявлення найрізноманітніших тестових випадків у сховищі тестів та їх сортування відповідно. Щоб алгоритм визначення пріоритетності міг це зробити, потрібно підготувати дані на основі яких можна виявити пріоритетність. Для цього, інформація яку у подальшому планують використовувати проходить етап кодування, та розміщується у векторі, для подальшого визначення відстані на основі тих чи інших характеристик. Отриману у результаті матриця відстані використовується для визначення пріоритетів.

Алгоритм визначення пріоритетів починається з того, що визначається найбільше значення у матриці, що відповідає парі найбільш різноманітних тестових випадків. Слід зазначити що матриця є симетричною, тому немає різниці з якого із трикутників починати пошук. Існує вірогідність що декілька пар тестів будуть мати однакову значення відмінності, у такому випадку послідовність буде обрано випадковим чином.

Коли у матриці знайдено найбільш різноманітне значення (найрізноманітніша пара тестових випадків) потрібно наступним кроком обрати для неї відповідну пару. Для кожного з них ми підсумовуємо всі відповідні

значення різноманітності, тобто весь стовпець або рядок у матриці. Тест із найбільшим значенням суми буде визначений як найбільш різноманітний з двох відносно інших тестових кейсів у матриці. Ця сума є підсумком усіх парних відстаней у матриці. Однак слід зазначити, що алгоритм включає лише значення над основною діагоналлю матриці відстані при обчисленні суми. Використання значені під основною діагоналлю включало б повторювані значення в суму, це призвело б до додаткових і непотрібних обчислень в алгоритмі, час виконання потрібно мінімізувати, так як ці додаткові розрахунки можуть кардинальним чином сказатися під час використання у великих сховищах тестів.

Коли алгоритм визначив найрізноманітніший тестовий випадок із пари, тестовий випадок додається до набору тестів із визначеними пріоритетами, а його стовпець та рядок видаляються із матриці відстані. Винятком із цього може стати ситуація коли два тести мають однакові суми. У цьому випадку алгоритм випадково вибере один із двох тестових випадків, який буде додано до набору тестів.

Цей алгоритм буде відтворено до тих пір, поки найбільшим значенням у матриці не стане нуль. Після цього тестові випадки, якщо такі є, які залишилися будуть розташовані у довільному порядку. Їх порядок не має значення оскільки випадок коли значення у матриці відстані дорівнює нулю означає що тестові випадки є однаковими за обраною характеристикою, тому і послідовність не має значення. Таким чином буде отримано набір тестових випадків упорядкованих за певною ознакою.

3.2 Стратегії кодування інформації

Сутність стратегій кодування полягає в тому, що вони готують інформацію для подальшого використання під час вимірювання відстаней, розміщують її у векторі, та забезпечують використання такого формату, який підходить для

використання. Прикладом стратегії кодування може бути використання специфічного іменування тестових випадків таким чином, що їх можна розрізнити за специфікою, модулем для перевірки якого вони використовуються, тощо. У разі використання різних стратегій кодування, можна використовувати відповідність рядків для того щоб визначити подібність на основі назв тестів, модулів для яких вони розроблені, конкретних задач які вони виконують.

Існує два типи тестової інформації, яку можна закодувати, а саме статична та динамічна інформація. Статичний тип містить інформацію, яка не вимагає запуску тестової системи. Наприклад, назви тестових випадків та їх код – це приклади статичної інформації, яка доступна без запуску програмного забезпечення. З іншого боку, дані тесту динамічного типу вимагають виконання програмного забезпечення, щоб інформація була доступною. Прикладом такої інформації є історія виконання тестів.

На додаток до того, що існують різні типи інформації, тести також мають різні рівні, наприклад, тести на рівнях модулів та інтеграцій. Кожен рівень тесту відповідає конкретній меті тесту під час тестування різних аспектів системи, наприклад інтеграційного тестування. Метою інтеграційного тестування є перевірити, чи працюють різні програмні модулі належним чином, коли вони поєднуються у одну систему.

Оскільки у різних організаціях, проектах, або навіть командах стандарти для написання тестів можуть відрізнятися, стратегії кодування будуть також відрізнятися. Відмінності можуть бути як у стандартах назв для тестів, структури наборів тестів, так і на більш абстрактному рівні, на етапі визначення характеристик різноманітності.

Закодовану інформацію можна використовувати у алгоритмах визначення пріоритетності, завдяки методам знаходження відстані (метрикам), які обчислюють відстані між тестами.

3.3 Алгоритми вимірювання відстані

Як було зазначено вище, знаходження відстані (метрики), обчислюють відстані між тестами. Для обчислення відстаней використовують множини або послідовності. Основною відмінністю є те, що множини не враховують порядок. Як правило, міри дистанції є парними, тобто вони обчислюють лише відстань між парою даних, наприклад, двома векторами. Основою цих розрахунків є спільність між двома об'єктами вхідних даних. Якщо вони не мають нічого спільного, отримана відстань дорівнює 1, і навпаки, якщо вони однакові, їх відстань дорівнює 0.

Усі значення відстаней для кожної пари значень формують матриці, які у свою чергу називають матрицями відстані. Отже матриця відстані – це матриця розмірності $n \times n$, де n – це кількість тестових випадків у наборі тестів. Потрібно зауважити що матриці відстані симетричні, тобто відстань між кожною парою значень з'являється двічі, один раз у верхньому та нижньому трикутниках матриці відповідно. У нашому випадку, матриці відстані відіграють важливу роль у процесі виявлення пріоритетів для тестів.

Розглянемо два методи пошуку відстані, а саме індекс Жакарра, та відстань Левенштейна. З цих двох мір відстані індекс Жакарра базується на множинах, тоді як відстань Левенштейна є мірою відстані на основі послідовності.

Індекс Жакарра – це бінарна міра подібності, яка була запропонована Полем Жакарром, та визначається як міра спільної частини, поділена на міру об'єднання множин [15]. Ця міра може бути задана формулою (1), яка зазначена нижче.

$$\frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{1}$$

де A та B – множини.

У свою чергу відстань Жакарра вимірює відмінність множин, та є доповненням коефіцієнта Жаккара до 1 і отримується відніманням коефіцієнта Жаккара від 1, або шляхом ділення різниці мір об'єднання і перетину двох множин на міру об'єднання.

Відстань Левенштейна – це міра відмінності двох послідовних рядків. Вона обчислює найменшу кількість операцій вставки, видалення і заміни, необхідних для перетворення одної послідовності в іншу [16]. Цю міру відмінності можна використати, наприклад, під час порівняння назв тестів і вона буде відображати скільки букв у назвах тестів потрібно замінити для того щоб із назви номер один отримати назву номер два.

3.4 Візуалізація отриманої інформації

Візуалізація інформації стосовно процесів тестування є не обов’язковим елементом систем постійної інтеграції, але може стати дуже приємним додатком до самої системи визначення пріоритетів, оскільки базуючись на основі інформації стосовно кодування інформації, відстані між тестами, та самих результатів тестування що може надати ту інформацію, доступ до якої зазвичай обмежений через її відсутність. Прикладами такої інформації для візуалізації може бути історія виконання тестів, відображення матриць подібності або відмінності.

Графічно відобразити історію виконання тестів можна за допомогою теплової карти. У такому випадку осями буде задана інформація про тестові запуски та унікальні тести які були проведені під час конкретного тестового запуску. Завдяки використанню різних кольорів на графіку можна легко відрізнити результати виконання тесту. Для зручного використання негативний результат можна позначити червоним кольором, позитивний – зеленим, тест який не було перевірено – білим. На основі отриманої теплової карти з деякої кількості виконання тестів, можна буде зрозуміти які результати було отримано протягом

часу. Можливо стане наглядно зрозумілим шаблон, що відобразить у якому модулі додатку найчастіше виникають зміни які призводять до негативного проходження тестів.

Стосовно графічного відображення схожості тестів за ознакою яка була оброблена під час кодування інформації, наприклад назвою, можна візуально відобразити тести за допомогою діаграми розсіювання. У такому випадку потрібно скористатися матрицею відстані яка була отримана шляхом перетворень що були зазначені у минулому розділі і з використанням цієї інформації побудувати сам графік. Представлена у такому вигляді інформація може стати в нагоді, оскільки вона дуже явно відображує групи або кластери на графіку, що означає що тести які потрапили у скупчення опинилися схожими за ознакою яка була обрана для формування матриці відстані.

3.5 Зменшення розмірності

Нажаль, візуалізація матриць відстані не є простою задачею у випадках коли розмірність матриці дуже велика, а оскільки розмірність лише збільшується зі збільшенням кількості тестів це стає проблемою. У такому випадку є сенс скористатися методами скорочення розмірності матриці. Техніки зменшення розмірності використовують вектор великої розмірності та роблять перетворення для зменшення розмірності. У випадку який розглядається тут, потрібно зменшення до розмірності двох оскільки це можна відобразити на графіку.

Стосовно методів зменшення розмірності – існує два основних типи: лінійні, та не лінійні. Основна різниця полягає в тому, що лінійні методи зосереджуються на збереженні точок даних що відрізняються під час зменшення розмірності на великій відстані, тоді як нелінійні методи більш зосереджуються на збереженні подібних точок на близькій відстані. До лінійних методів включають

багатовимірне шкалювання, та метод головних компонентів. Прикладом нелелійних методів може бути т-розподілене вкладення стохастичної близькості.

3.5.1 Метод багатовимірного шкалювання

Багатовимірне шкалювання – метод аналізу і візуалізації даних за допомогою розташування точок, відповідних досліджуваним об'єктів, в просторі меншої розмірності ніж простір ознак об'єктів. Точки розміщуються так, щоб попарні відстані між ними в новому просторі якомога менше відрізнялися від емпірично вимірюваних відстаней в просторі ознак досліджуваних об'єктів [17].

Існує два типи методів багатовимірного шкалювання, а саме метричні та неметричні. Основною відмінністю є те, які обчислення відбуваються, та які метрики використовуються під час обчислень. Якщо елементи матриці відстаней отримані по інтервальним шкалами, метод багатовимірного шкалювання називається метричним. Коли шкали є порядковими, метод багатовимірного шкалювання називається неметричним. Міра відмінностей відстаней в вихідному і новому просторі називається функцією стресу.

3.5.2 Стохастичне вкладення сусідів с Т-розподілом

Стохастичне вкладення сусідів с Т-розподілом – це не лінійний метод зменшення розмірності який базується на машинному навчанні запропонований ван дер Маатеном і Джефрі Гінтоном [18]. Цей метод працює шляхом вкладення багатовимірних даних до двох- або трьох-вимірного простору для подальшого використання, наприклад шляхом відображення на графіках. Алгоритм працює

таким чином, що точки які схожі за ознаками розташовуються близько одні до одних, у той час як відмінні за ознаками точки відображаються віддаленими.

Цей метод доволі часто використовується для візуалізації даних за кластерами, але треба зазначити, що зображення яке буде отримано насамперед буде залежати від тих характеристик які було обрано для виділення параметрів. Існує імовірність, що не якісно обрані параметри призведуть до появи кластерів у тих місцях, де їх не повинно бути.

3.6 Приклад визначення пріоритетів

Алгоритм розстановки пріоритетів може базуватися на жадібному алгоритмі відбору. Основна ідея жадібних алгоритмів відбору полягає в тому, що для кожного тестового випадку, буде обраний тестовий випадок який має найбільшу відстань у матриці відстані. Таким чином, відносно просто розширити жадібний алгоритм відбору, щоб застосувати його також до пріоритетів [19]. Це можна зробити шляхом змушення відібрати таку ж кількість тестових випадків, як і в оригінальному наборі тестів. Після того, як жадібний алгоритм визначення пріоритетів знаходить найвищу відстань у матриці відстані, він обчислює загальну суму відстаней між парами тестів до інших тестів. Іншими словами, обирає один із двох тестів і підсумовує відстані між ним та всіма іншими тестами в матриці. Пізніше тестові випадки із найбільшою сумою будуть додані до набору із визначеними пріоритетами, та вилучені з матриці відстані. Алгоритм повторює цей процес до тих пір, поки матриця відстані не стане порожнію. Таким чином буде отримано набір тестів відсортованих у порядку пріоритетності, який у подальшому використовується для проведення автоматизованого тестування.

4 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

4.1 Засоби реалізації

Однією із частин проведення дослідження, можна навіть сказати попереднім етапом або однією із передумов – є проектування та розробка системи що наддасть змогу перевірити гіпотезу стосовно використання пріоритетів тестів. Система повинна надавати змогу обробляти інформацію та тестові випадка, та визначати пріоритет виконання тестових випадків.

Обраними засобами реалізації є мова програмування Python – інтерпретована об’єктно–орієнтована мова програмування високого рівня зі строгою динамічною типізацією [20]. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Ця мова програмування також добре інтегрується разом із основними система постійної інтеграції, такими як Jenkins та Bamboo і надає змогу без особливих перешкод використовувати розроблені додатки.

Під час розробки були використані широковідомі бібліотеки, наприклад, numpy, textdistance, та scipy які надали змогу використовувати перевірені реалізації методів вимірювання відстані та інших математичних дій і суттєво полегшили процес розробки тестової системи, зберігши впевненість у використовуваних методах.

У якості середовища тестування було використано Robot Framework. Цей фреймворк зазвичай використовується під час автоматизованого тестування і має цікаву структуру тестових випадків [21]. Слід зазначити що він використовує підхід на основі використання ключових слів. Усі тестові випадки у тестовому додатку були розроблені із використанням цієї бібліотеки.

Іншою використаною бібліотекою стала pyplot. Ця бібліотека використовується для відображення графічних даних. У рамках даної системи такими даними стали графік подібності, та теплову карту перевірки тестів.

4.2 Архітектура додатку

Цей розділ моє за мету відобразити структуру додатку та надати її скорчений опис, разом із взаємодією між окремими частинами. Архітектуру системи наведено на рис. 4.1

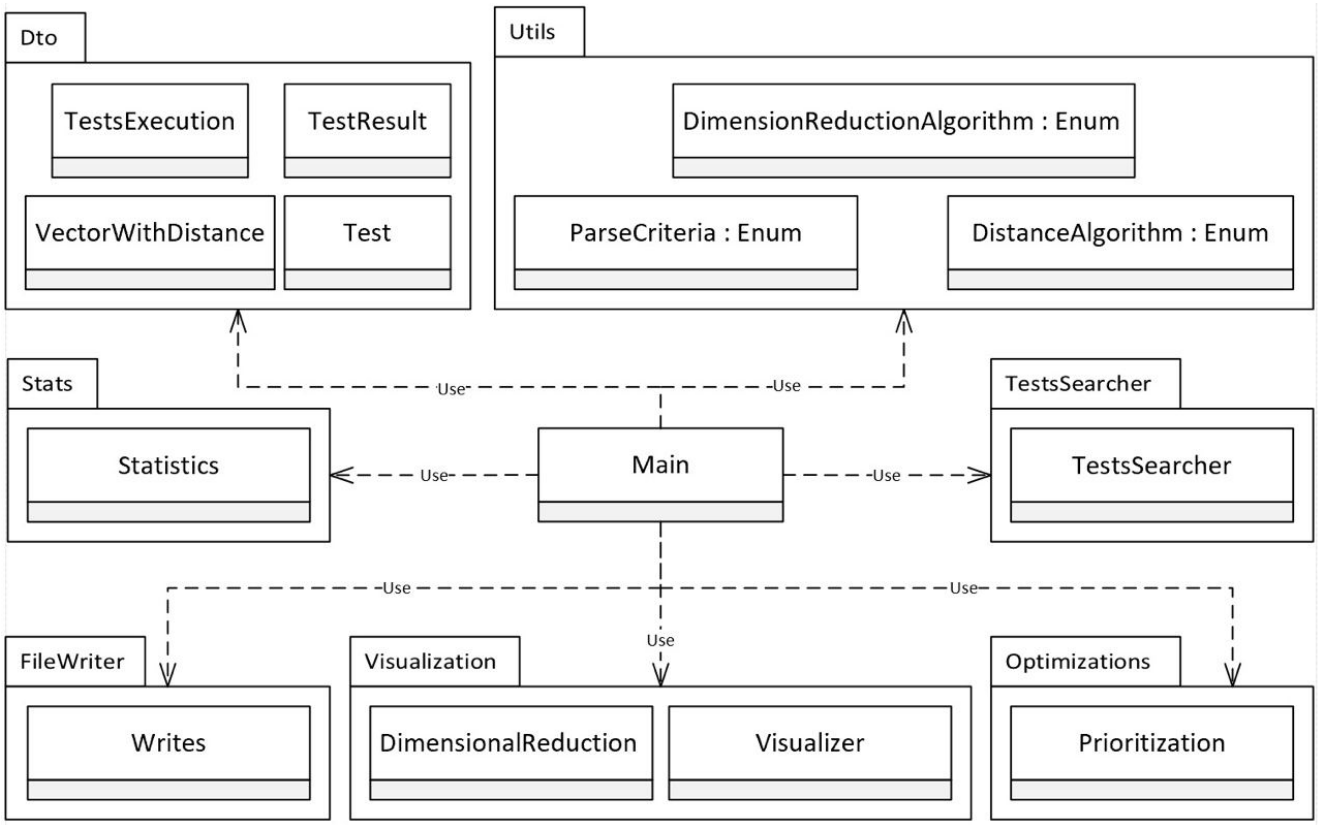

Рисунок 4.1 – Діаграма класів

Як можна побачити рис. 4.1, який відображає архітектуру, була розроблена модульна система яка надає змогу легкого розширення у разі потреби. У результаті було отримано систему, у центрі якої є клас Main, який відповідає за використання різних модулів з метою проведення необхідних вимірювань, збору даних для цього дослідження, проведення процесу тестування.

Модуль Utils відповідає за функціональність яка використовується різними частинами у розробленому додатку. Основними фрагментами цього пакету є перелічення алгоритмів, що використовуються під час визначення відстані, а саме

зменшення розмірності, та вибору критеріїв за якими буде проведено аналізу тестових випадків, прикладами таких даних є назви та код тестових випадків. Серед алгоритмів вимірювання відстані які використовуються у тестовій системі присутні індекс Жакарр, відстань Левенштейна, нормалізована відстань ущільнення та у якості базового варіанту – варіант із використанням випадковості.

Модуль Optimizations який складається із класу Prioritization. Клас Prioritization відповідає за вимірювання відстані між одиницями тестів, формуванні нової матриці яка складається із об’єктів з деякими тестами та відстанню між ними. Цей клас також відповідає за визначення та формування послідовності виконання тестових випадків на основі відстані між ними.

Модуль TestsSearcher використовується за єдиним призначенням – пошуку та повернення усіх тестових випадків у рамках проекту для їх подальшого використання під час визначення пріоритетності.

Модуль Stats складається із класу DataFormer. Цей клас використовується для обробки та пошуку статистичних даних, формування json файлу із історією перевірки тестів, для того щоб мати можливість скористатися цією інформацією під час формування графіків. Він також стає у нагоді для вирахування кількості не пройдених тестів під час їх виконання, що полегшує статистичну частину цієї роботи.

Модуль Visualization – відповідає за графічну візуалізацію статистичної інформації, та складається із двох класів DimensioinalReduction та Visualizer. Перший клас, як можна визначити із назви, використовується для зменшення розмірності даних, під час цього процесу він може використовувати один із двох методів, багатовимірне шалювання або T-розподілене вкладення стохастичної близькості. Другий клас, у свою чергу, відповідає за процес створення графіків на основі отриманої інформації. Є два типи графіків які можна відобразити за допомогою цього класу – це графік подібності, та теплову карту результатів проведеного тестування.

Модуль Dto складається із чотирьох класів які використовуються як об’єкти для передачі інформації. Серед них є уявлення Test – якє відповідає тестовому

випадку та зберігає інформацію стосовно назви тесту, назви його тестового набору та його код. VectorWithDistance – клас який використовується для зберігання у матрицях відстані і має у собі пару тестів та відстань між ними. Клас TestResult – потрібен для того щоб зберігати інформацію стосовно окремого тесту що був перевірений, та зберігає назву тестового випадку, та результат його виконання. У разі позитивного виконання – 1, у разі негативного виконання – (-1), якщо тест не був використано під час перевірки – 0. Останнім класом у цьому модулі є TestsExecution. Цей клас використовується для зберігання статистичної інформації про виконання тестових випадків у системі, та складається із часу виконання, та списку виконаних тестових випадків.

Клас FileWriter використовується за єдиним призначенням – сформувати файл який буде використано для старту виконання тестування. Він зберігає у собі тести які потрібно перевірити у зазначеній, відсортованій послідовності.

4.3 Алгоритм використання

Використання системи починається із основного процесу Main який поєднує у собі процес використання різних модулів з метою досягнення певного результату. Слід зазначити що система розраховується для використання у якості частини безперервної інтеграції, тому цей процес буде виконано, наприклад, коли починається нова зборка, або додано код до загального сховища.

У першу чергу, Main починає процес пошуку тестових випадків серед коду додатку, або сховища тестів, та формує їх у такому вигляді, який у подальшому можна використати.

Після цього, починається процес формування матриці відстані. Цей процес залежить від обраних параметрів, а саме критерію за яким буду проведено розрахунок відстані, таким критерієм може бути назва тестових випадків, або їх код. Другим критерієм є алгоритм який буде використано, варіантами є індекс

Жакарра, відстань Левенштейна, нормалізована відстані ущільнення, або випадковий вибір без залежності від параметрів.

Отримана матриця відстані використовується для визначення пріоритетів виконання тестових випадків, більш детально цей алгоритм було зазначено у розділі 3.1.

Після визначення пріоритетів виконання тестових випадків, цей список переноситься до файлу який буде використано під час процесу тестування.

Оскільки усі передумови виконання тестування було виконано, починається процес тестування. Результати проведеного тестування можуть бути використані для зазначення на графіках.

4.4 Аналіз отриманих результатів

Дуже важливою частиною аналізу дослідження є використання різноманітних метрик. Серед метрик які було обрано для проведення даного дослідження слід зазначити метрики APFD – середній процент знайдених помилок, покриття помилок, час виконання визначення пріоритетів.

APFD – це цікава метрика яка намагається відобразити як швидко знаходиться тестовий випадок який знайде помилку. Процес визначення пріоритетів виконання тестових випадків на основі різноманітності має у якості цілі знайти та впорядкувати тестові випадки таким чином щоб найбільш різноманітні були перевірені у першу чергу, що призводить до припущення, що це збільшує вірогідність знайти різноманітні помилки, через що ця метрика є дуже актуальною. Іншою перевагою швидкого знаходження помилок є те, що якщо використання усіх тестів є неможливим, то найбільш важливі уже були перевірені.

Ця метрика має такий вигляд:

$$APFD = 1 - \frac{\sum_{i=1}^{m} TF_i}{nm} + \frac{1}{2n} \quad (2)$$

де m – кількість помилок; T – набір тестів; TF_i – позиція першого тесту який знайшов помилку; n – кількість тестів.

Другою метрикою є покриття помилок. Для підрахування цієї метрики потрібно поділити кількість помилок які були знайдені під час тестування, на сумарну кількість помилок у системі що тестується. Вона має такий вигляд:

$$Fault\ coverage = \frac{m}{n} \quad (3)$$

де m – кількість знайдених помилок; n – загальна кількість помилок.

У цьому дослідження було декілька раз проведена перевірка із допомогою цієї метрики, але з використанням різних масштабів наборів тестів. Було обрано обмежену кількість найбільш різноманітних тестових випадків для проведення тестування, або іншими словами – перевірено наскільки ефективно, буде показувати себе визначення пріоритетів у разі використання обмеженого набору тестових випадків. Сумарно було обрано 10 наборів із набору тестів з вже визначеною пріоритетністю, з кроком у 10% від загальної кількості.

Останнім але не менш важливим є вимірювання часу потрібного на виконання процесу визначення пріоритетів тестових випадків. Цей показник є досить зрозумілим і наддасть змогу отримати більш детальну картину разом із використанням інших метрик.

Усі показники разом повинні сформувати цілісну картину результатів використання методів оптимізації. На основі отриманої інформації можна зробити висновок щодо доцільності використання методів, та якості їх персонального внеску як у процес розробки програмних додатків, так і до вирішення визначених попередньо проблем.

4.4.1 Середній процент знайдених помилок.

Першою оцінкою визначенню пріоритетів виконання тестових випадків стають результати отримані метрикою середній процент знайдених помилок. Слід зазначити, що тестова система має 101 тестовий випадок так 3 помилки які покриті цими тестовими випадками. Усі дослідження були проведені по 10 раз, для збору статистичної інформації.

Перший графік відображає використання різних методів пошуку відстані між тестами на основі критерію Name. Ознакомится з результатами можно на рис. 4.2.

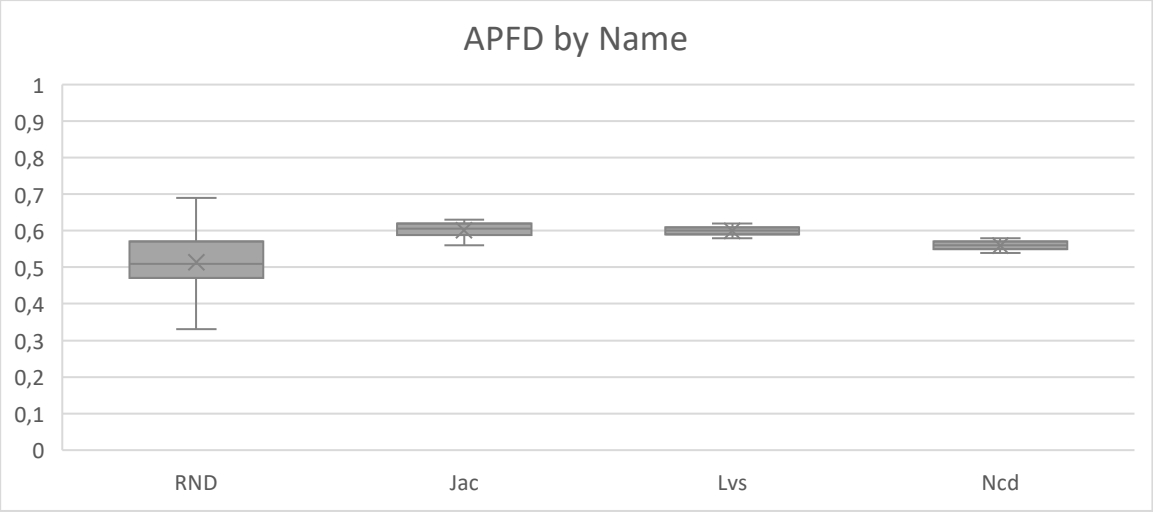

Рисунок 4.2 – AFPD за критерієм назви

Під час перевірки було визначено що Jac, Lvs, Ncd потрапляють у діапазон значень Rnd. Слід зазначити, що використання випадкового сортування призводить до того, що значення його ефективності відносно AFPD стає дуже варіативним. Під час дослідження для Rnd було отримано діапазон між 41% та 69%.

У свою чергу значення медіан для кожного з методів визначення відстані є такими: Rnd – 51%, Jac – 61%, Lvs – 60%, Ncd – 56%. З зазначеного графіку можна зробити закономірний висновок, що відмінні від Rnd методи майже на мають мінливості і через це є більш стабільними, оскільки для того щоб вони біль кардинально відрізнялися, під час використання існуючого алгоритму повинна

відбутись ситуація коли є декілька пар тестових випадків які мають однакову відстані.

Наступний графік відображає використання різних методів пошуку відстані між тестами на основі Code. Результати зображені на рис. 4.3.

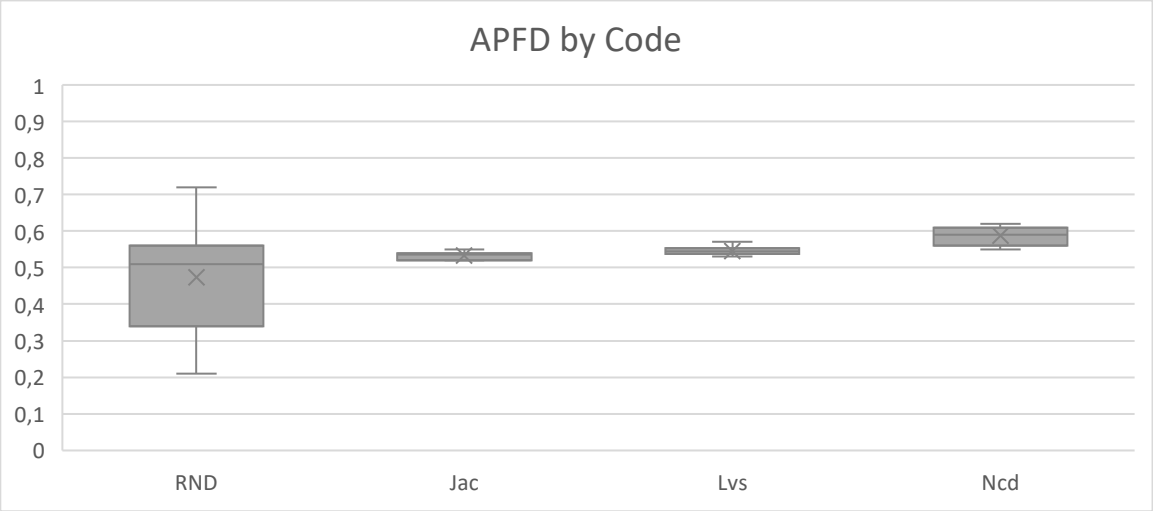

Рисунок 4.3 – AFPD за критерієм коду

Не дивлячись на зміну критерію оцінки який підлягає перевірці, усереднене значення не дуже сильно змінилося відносна минулого графіку. Цікавою відмінністю для обох графіків є те, ще у системі що тестується APFD для Jac, Lvs, Ncd стабільно вище ніж у Rnd варіанті, що є приємним знаком. Слід зазначити що цього разу варіативність Ncd збільшилась як і значення APFD, але це може бути через особливості тестових випадків. Значення медіан для кожного з методів визначення відстані є такими: Rnd – 51%, Jac – 54%, Lvs – 55%, Ncd – 59%.

4.4.2 Покриття помилок

Другою характеристикою що вимірюється є покриття помилок. Цей підрозділ має у собі 2 графіки які відображають скільки відсотків несправносте покрито за кожні 10% обраних тестових випадків. Слід зазначити, що перед

використанням тестовим випадкам була надала пріоритетність за певною ознакою. Кожен графік покриття несправносте містить певні критерії та пов’язані з ними методи підрахунку.

Перший графік покриття помилок зображено на рис. 4.4, він відображає залежність між процентною кількості використаних тестових випадків та процентною кількість знайдених помилок при використанні різних методів пошуку відстані за критерієм назви.

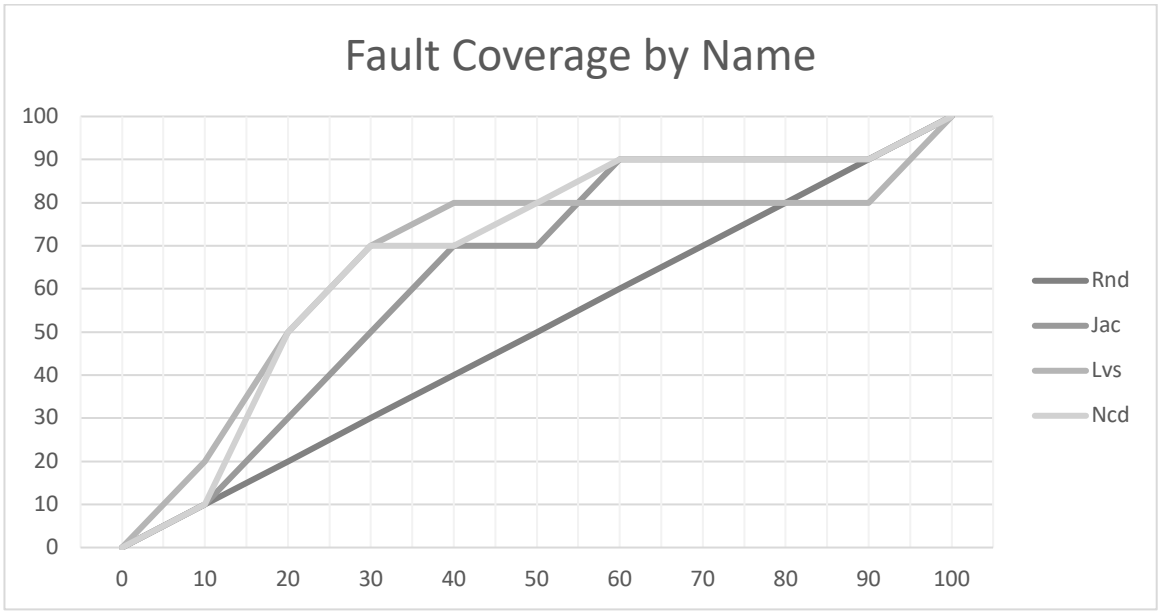

Рисунок 4.4 – Покриття помилок за критерієм назви

Стандартні значення представлені прямою Rnd, яка відображує лінійну залежність між кількістю знайдених помилок та кількістю використаних тестових випадків, оскільки у разі великої кількості перевірок варіативність буде працювати таким чином, що підсумковий результат буде наближатися до середнього значення. Більш цікавим є те, що усереднено використання методів пошуку відстані призводить до збільшення кількості знайдених помилок відносно Rnd з використанням тієї ж кількості тестових випадків, хоча це може бути і особливістю використаного сховища тестів. Не є дивовижним той факт, що без зміни назв та коду тестових випадків варіативність результатів з використанням кожного з методів окрім Rnd не є великою, оскільки єдиним чинником який на неї впливає у

цьому випадку є ситуація коли декілька пар тестів мають однакову відстань і починає працювати вибір випадкової пари із двох.

З методів вимірювання відстані що залишилися у більшості випадків найменьш ефективним став Jac, але слід зазначити, що Lvs просідає за ефективністю після вибору 40% тестових випадків. Найбільшу різницю із Rnd можна побачити у діапазоні між 70% та 80% покриття тестових випадків і різниця становить майже 40% знайдених помилок.

Наступним графіком покриття помилок використовує усі ті ж самі методи що були зображені на рис. 4.4., але з використанням критерію відмінності коду.

Результати виконання та збору статистики стосовно покриття помилок за критерієм Code відображені на рис. 4.5.

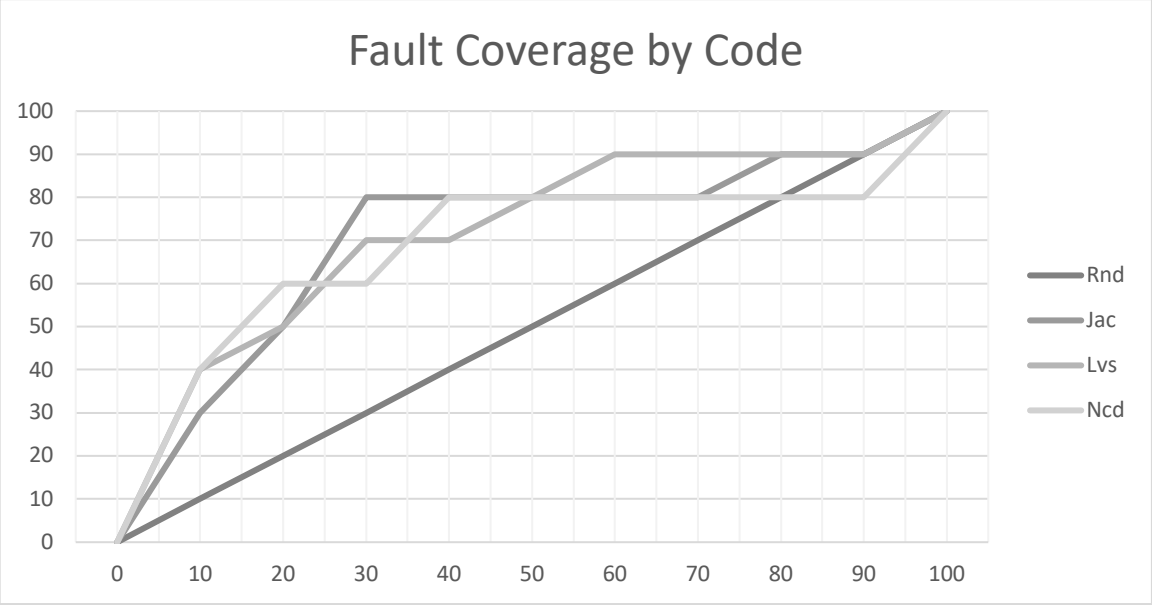

Рисунок 4.5 – Покриття помилок за критерієм коду

Більш того можна побачити, що вони досить швидко досягають високого рівня покриття помилок, а після вибору приблизно 30% тестових випадків із усіх рівень знаходження помилок починає зменшуватися. Найбільш сильно ефективність зменшується у Ncd який надовго заливається на єдиному рівні і навіть починає показувати гірші результати ніж Rnd після вибору 80% тестових випадків.

4.4.3 Час використання

Важливим параметром для оцінки ефективності методів оптимізації тестування є час потрібний для вимірювання часу потрібного на виконання набору тестів. Під час проведення тестування тестове сховище налічувало 110 тестових випадків. Уся інформація пред'явлена у вигляді кількості витрачених секунд на процентну кількість тестів за виявленим пріоритетом.

Табл. 2 відображає кількість часу який було затрачено на визначення пріоритетів та виконання обраного набору тестів за критерієм Name. Вона поєднує у собі усі алгоритми вимірювання відстані, та випадковий варіанти.

Таблиця 2 – Час виконання за критерієм Name

Кількість тестів (%)	Rnd	Jac	Lvs	Ncd
10%	0,225	0,469	2,424	0,424
20%	0,316	0,539	2,516	0,510
30%	0,444	0,669	2,643	0,641
40%	0,524	0,757	2,713	0,725
50%	0,626	0,925	2,825	0,894
60%	0,767	1,081	2,944	1,034
70%	0,884	1,179	3,115	1,145
80%	1,051	1,348	3,285	1,315
90%	1,222	1,485	3,460	1,459
100%	1,369	1,647	3,630	1,683

На табл. 2 представлено результати виконання наборів тестів з різними процентними кількостями тестових випадків та методами розрахунку відстані між ними за ознакою Name. Цікавою відмінність від усіх інших варіантів тут став метод Lvs, оскільки час виконання підрахунків збільшено у декілька разів відносно інших

методів. У результаті можна зазначити що Rnd виявляється трошки швидшим у процесі перевірки через відсутність потреби призводити підрахунку. Слід також урахувати, що згідно попередньо перевіреним метрикам, не дивлячись на трохи швидший процес первинної обробки тестів,

На табл. 3 зображено результати перевірки тих самих тестових випадків що і у минулому набору тестів, але за критерієм Code.

Таблиця 3 – Час виконання за критерієм Code

Кількість тестів (%)	Rnd	Jac	Lvs	Ncd
10%	0,225	0,545	6,435	0,640
20%	0,316	0,605	6,480	0,701
30%	0,444	0,733	6,584	0,822
40%	0,524	0,813	6,640	0,903
50%	0,626	0,928	6,737	1,008
60%	0,767	1,043	6,875	1,154
70%	0,884	1,201	6,984	1,287
80%	1,051	1,382	7,186	1,490
90%	1,222	1,521	7,316	1,611
100%	1,369	1,738	7,505	1,804

Критерії Code робить все більш явною різницю між Lvs та іншими методами, позначаючи час потрібний на підрахунки відстані між кодом тестових випадків. Час виконання контрольної групи Rnd не змінився, оскільки ніякого аналізу не було проведено. Групи Jac та Ncd також трішки збільшили у тривалості виконання через збільшений об’єм інформації яку було потрібно обробити, але різниця між ними та контрольною групою не є значною.

Узагальнено можна зробити висновок, що у порівнянні між Jac та Lvs, Jac отримує беззастережну перемогу, з вигодою у декілька разів під час аналізу коду тестових випадків. NCD у свою чергу показує досить постійний результат, який

збільшується у середині та під кінець виконання . Також не слід випускати з уваги інші важливі метрики.

4.4.4 Графічне відображення

Методи обчислення відстані між тестами та сбір статистики виконання роблять можливим виконання ще однієї цікавої функції – графічне відображення статистичної інформації. Ця інформація може стати у нагоді оскільки вона має можливість відобразити якісь залежності, які зазвичай не потрапляють у поле зору.

Першим зображенням яке було отримано є діаграма розсіювання схожості тестових випадків, зображена на рис. 4.6.

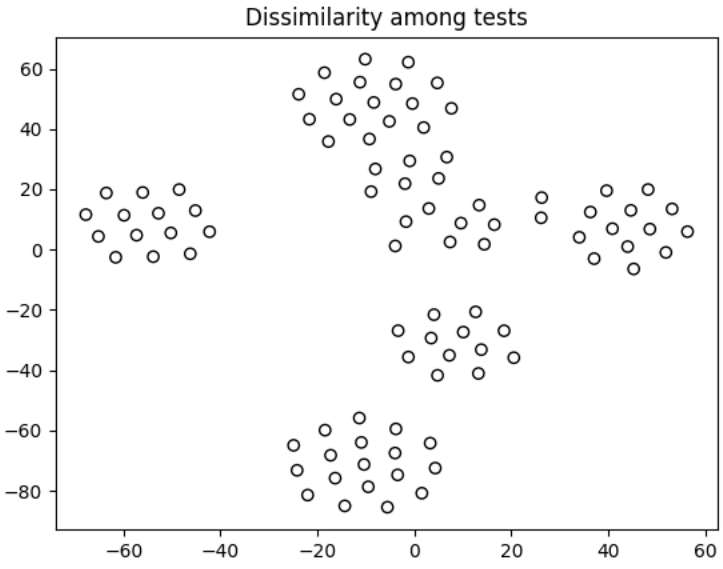

Рисунок 4.6 – Діаграма розсіювання схожості тестових випадків

Ця діаграма була отримана при використання методу Jac разом із методом T-розподіленого вкладення стохастичної близькості для зменшення матриці відстані розмірності до двох. Для обчислення матриці відстані за основу було обрано

критерій оцінки Name. Графік робить достатньо явними різні скупчення, які є відповідними за обраним параметром.

Наступним є діаграма розсіювання, яка використовує різні параметри, вона зображений на рис. 4.7.

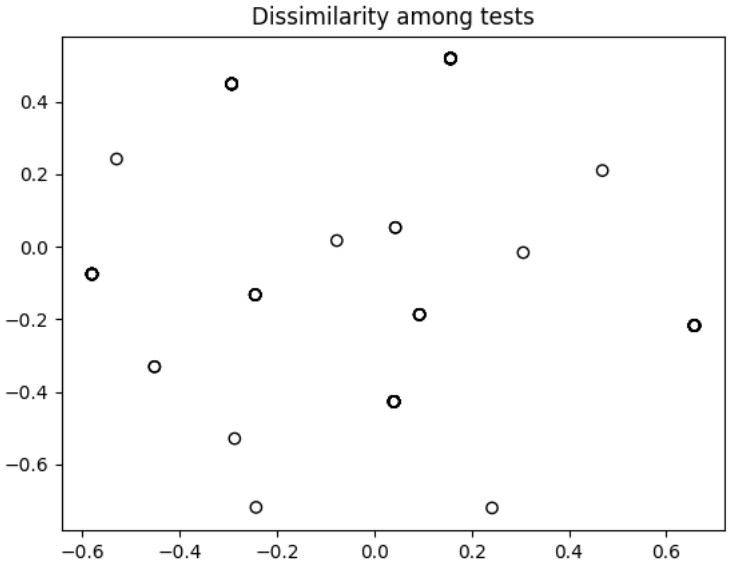

Рисунок 4.7 – Діаграма розсіювання схожості тестових випадків

Ця діаграма також відображає відповідність між тестовими випадками. Для її формування було обрано метод Lvs та Code у якості критерію оцінки відстані, ще одним цікавим моментом є те, що для зменшення розмірності цього разу було використано метод багатовимірного шалювання.

Цікавим дослідженням є порівняння часу потрібного на виконання зменшення розмірності з використанням різних методів зменшення розмірності. Підчас експерименту обидва методи користувалися матрицею відстані на основі критерію Name, яка була сформована з використанням критерією Jac.

Результати проведеного заміру часу виконання методів зменшення розмірності зображено на табл. 4

Як зрозуміло з таблиці, було отримано майже трикратну різницю у часі виконання між двома методами, з перемогою у метода багатовимірного шкалювання.

Таблиця 4 – Час виконання методів зменшення розмірності

	Т-розподілене вкладення стохастичної близькості	Багатовимірне шкалювання
Час	0.177	0.065

Останнім але не менш цікавим графіком який було вирішено сформувати маючи статистичну інформацію є теплова карта виконання тестів. Цей графік зображено на рис. 4.8 і він відображає історію позитивного та негативного проходження окремих тестів.

Це графічне зображення є досить ефективним з точки зору надання інформації, оскільки воно надає змогу зрозуміти як поводили себе тести протягом часу. Графік складається із тестових випадків, часу та результатів їх виконання. Колір відображує результат виконання тестового випадку, де червоний – негативно, зелений – позитивно, жовтий – тест не було використано.

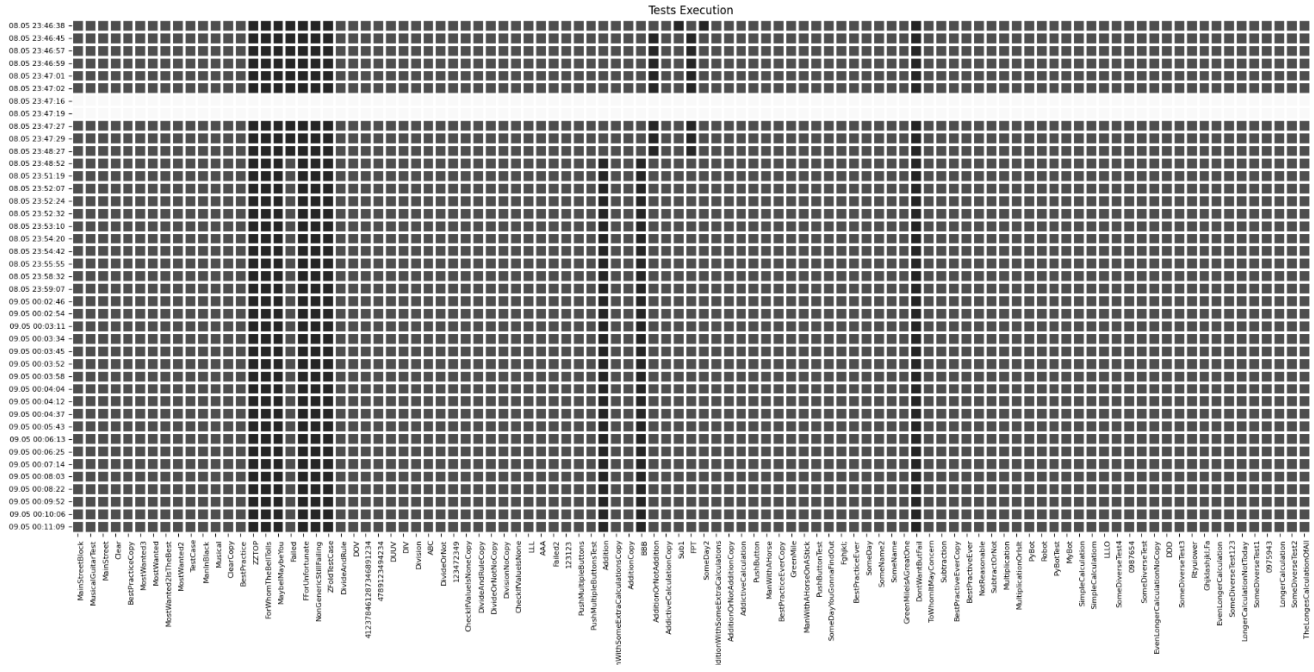

Рисунок 4.8 – Результати виконання тестових випадків за часом

Як зрозуміло з рис. 4.8, протягом певного часу усі тести не були перевірені через якісь труднощі. Перевагами які надає цей графік є можливість більш

наглядно відобразити результати, що може стати у нагоді і допомогли зрозуміти взаємозв’язок між виникнення помилок у декількох тестових випадках.

У результаті використання методів для зображення деякої статистичної інформації на основі даних отриманих під час визначення пріоритету виконання тестів можна зробити висновок, що усі вони надають змогу відобразити корисні дані які можуть надати більше інформації стосовно системи що тестується.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було проведено аналіз галузі тестування, в особливості автоматизованого, виявлено ряд запитань які потребують перевірки в рамках виконання атестаційної роботи, сформульовано основну проблему. Також було проведено теоретичне дослідження методів оптимізації виконання тестових випадків, визначено с планом дослідження для виправлення виявленої проблеми, розроблено додаток який надає змогу перевірити поставлену гіпотезу.

Основна проблема полягає у тривалому часі очікування зворотного зв'язку від системи постійної інтеграції у разі перевірок крупних та наповнених додатків, або сховищ тестів. Оскільки ця проблема стає все більш актуальною для великої кількості кампаній та розробників, існують попередні дослідження та виявлені основні напрямки оптимізації цих процесів. Обраним варіантом для дослідження став метод визначення пріоритетності виконання тестових випадків за певною ознакою, та поставлено гіпотезу щодо використання певної ознаки.

Основною ідеєю дослідження була перевірка гіпотези стосовно використання пріоритетів виконання тестових випадків, беручи до уваги їх різноманітність. Гіпотеза полягає в тому, що схожі тестові випадки мають схожий результат виконання, що призводить до думки, що використання більш різноманітної низки тестових випадків може призвести до більш швидкої відповіді на запитання, чи є у програмному додатку помилки, що у свою чергу може прискорити цикл розробки програмних додатків та надати зворотній зв'язок більш швидко. Додатком до основної ідеї є використати дані які були отримані під час основного процесу визначення пріоритетів для їх графічного відображення, що наддасть змогу мати більш цілісне уявлення стосовно процесів тестування та визначення пріоритетності.

Було обрано декілька методів для обчислення різноманітності тестів та візуалізації інформації, визначено з метриками для оцінки проведеного

вдосконалення, розроблено тестовий додаток який надає змогу перевірити гіпотезу, та проведено відповідне дослідження.

У результаті проведення дослідження було виявлено, що використання визначення пріоритетів виконання має як свої гідності, так і маленький недолік. Практичне тестування гіпотези на невеликому сховищі тестів показало свої позитивні сторони, заробивши більш високий результат за метриками середнього проценту знайдених помилок, та покриття помилок відносно базового методу, які були основними для цього дослідження. Слід також зазначити що у зв'язку із додаванням логіки аналізу додатку на етапі підготовки тестового середовища до виконання, сумарний час виконання зазнає невеликого збільшення, але цей показник є недостатньо значним у порівнянні із перевагами використання методу визначення пріоритетів за іншими метриками.

У якості цікавої особливості серед різних методів визначення відстані між тестами можна зазначити досить серйозну різницю у часі виконання методу Левенштейна відносно інших. На мою думку, з урахуванням усіх метрик, метод Жаккара є найбільш цікавим для використання у порівнянні з іншими, з невеликою перевагою над методом нормалізованої відстані ущільнення.

Відображення статистичної інформації яка була здобута під час виконання визначення пріоритетів та виконання тестових випадків є дуже приємним бонусом, оскільки надає змогу відобразити ті дані, до яких зазвичай нема доступу, наприклад аналіз схожості тестових випадків за критеріями назви чи коду. Відображення результатів виконання тестових випадків у продовж певного часу також є позитивним результатом, який надає змогу знайти певним взаємозв'язок між різними тестами.

Частина інформації із дослідження була опублікована у рамках міжнародної конференції Education and Science of Today: Intersectoral Issues and Development of Science. У якості подальших досліджень можна розглянути використання динамічної інформації для оцінки тестових випадків, наприклад змін у коді, тестове впровадження у більшу та реальнішу систему для подальшого спостереження за результатами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ledru Y., Petrenko A., Boroday S., and Mandran N. Prioritizing test cases with string distances // Automated Software Engineering, 2012. – vol. 19, № 1, с. 65 – 95.

2. Agile Coach // Atlassian. URL: https://www.atlassian.com/agile (дата звернення: 02.02.2021)

3. What is DevOps? // Amazon. URL: https://aws.amazon.com/devops/what-is-devops/ (дата звернення: 03.02.2021)

4. Myers G. The Art of Software Testing. Hoboken: John Willey & Sons, 2005. – 255 с.

5. Despa M. Comparative study on software methodologies, 2014.– vol. 5, №3.

6. Лановий О.Ф. Про один підхід до функціонального тестування web-додатків // Поліграфічні, мультимедійні та web-технології. Т1. Тез. Допов. 2-й Міжнарод. Науч.-техн. Конф. (16-22 травня 2017) / редкол.: В.Ф. Ткаченко, И.Б. Чеботарева и др.– Харків: ХНУРЕ, 2017. –246 с.

7. Duvall P. M., Matyas S., and Glover A. Continuous integration improving software quality and reducing risk. Boston: Addison-Wesley, 2007.

8. Turevska, O., Shubin, I. Improving the automated testing of Web-based services by reflecting the social habits of target audiences. 2015 Information Technologies in Innovation Business Conference, ITIB 2015 – Proceedings, 2015, с. 93 – 96.

9. Лановий О.Ф. Візуалізація в методах тестування програмного забезпечення // Харків, 6-а Міжнародна науково-технічна конференція «ІСТ-2017», ХНУРЕ, 11-16 вересня 2017 р., С.110-111.

10. Bhantnagar K. Regression Test Case Selection Using Machine Learning // Medium URL: https://medium.com/analytics-vidhya/regression-test-case-selection-using-machine-learning-241ded86f559 (дата звернення: 24.03.2021 р.).

11. Arcuri A., Iqbal M., Briand L. Black-box system testing of real-time embedded systems using random and search-based testing. // Testing Software and Systems, 2010. – с 99 – 105.

12. Chen T., Kuo F., Merkel R., Tse T. Adaptive random testing: The art of test case diversity. // Journal of Systems and Software, 2010. – с. 60 – 66.

13. Harman M. The current state and future of search based software engineering // Future of Software Engineering, 2007, с. 342–357

14. Clustering Methods // Science Direct. URL: httpswww.sciencedirect.com/topics/computer-science/clustering-method (дата звернення 28.03.2021 р.)

15. Jaccard Index // DeepAI.org URL: https://deepai.org/machine-learning-glossary-and-terms/jaccard-index (дата звернення: 04.04.2021 р.).

16. Nickhil B. The Levenshtein Distance Algorithm // Dzone. URL: https://dzone.com/articles/the-levenshtein-algorithm-1 (дата звернення: 08.04.2021 р.)

17. Multidimensional Scaling // Science Direct. URL: https://www.sciencedirect.com/topics/agricultural-and-biological-sciences/multidimensional-scaling (дата звернення: 10.04.2021 р.).

18. Introduction to T-sne / Data Camp. URL: https://www.datacamp.com/community/tutorials/introduction-t-sne (дата звернення: 15.03.2021 р.).

19. Фундукян А.А., Лановий О.Ф. Метод розстановки пріоритетів тестів // Education and Science of Today: Intersectoral Issues and Development of Science, Кембридж, 19 березня 2021 р., №2. С108 – 109.

20. The Python Tutorial / Python. URL: https://docs.python.org/3/tutorial/index.html (дата звернення: 16.04.2021 р.).

21. Basic Concepts of Robot Framework // Robotcorp docs URL: https://robocorp.com/docs/languages-and-frameworks/robot-framework/basics (дата звернення: 20.04.2021 р.).