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

Факультет інформатики та обчислювальної техніки

(повне найменування інституту, факультету)

Автоматизованих систем обробки інформації і управління

(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (ініціали, прізвище)
“ ” 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення інформаційних управляючих систем та технологій»

спеціальності «121 Інженерія програмного забезпечення»

на тему *Програмне забезпечення з аналізу результатів інтеграційних тестів*

Виконав: студент IV курсу, групи *ІП-63 Пилипенко Володимир Олександрович* _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *ст.в. кафедри АСОІУ Халус О.А.* _____
посада, науковий ступінь, вчене звання, прізвище, і ім’я, по батькові (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук К.І.* _____
посада, науковий ступінь, вчене звання, прізвище, і ім’я, по батькові (підпис)

Рецензент:

проф. каф. ТК, д.т.н., проф. Стенін О.А.
посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – *121 Інженерія програмного забезпечення*

Освітньо-професійна програма – *Програмне забезпечення інформаційних управляючих систем та технологій*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис)
“___”_____2020 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Пилипенка Володимира Олександровача
(прізвище, ім’я, по батькові)

1. Тема проєкту « Програмне забезпечення з аналізу результатів інтеграційних тестів »
керівник проєкту *Халус О.А., ст. викладач*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “07” травня 2020 р. №1081-с

2. Термін подання студентом проєкту «08» *червня 2020 року*

3. Вихідні дані до проєкту
Технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог, математичне забезпечення
2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, архітектура програмного забезпечення, конструювання програмного забезпечення, аналіз безпеки даних
3) Аналіз якості та тестування програмного забезпечення: аналіз якості ПЗ, процесів тестування, опис контрольного прикладу
4) Впровадження та супровід програмного забезпечення: розгортання програмного забезпечення, робота з програмним забезпеченням

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема структурна класів програмного забезпечення

3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «10» *березня 2020 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	19.03.2020	
2.	*Аналіз існуючих методів розв'язання задачі*	26.03.2020	
3.	*Постановка та формалізація задачі*	26.03.2020	
4.	*Аналіз вимог до програмного забезпечення*	02.04.2020	
5.	*Алгоритмізація задачі*	02.04.2020	
6.	*Моделювання програмного забезпечення*	09.04.2020	
7.	*Обґрунтування використовуваних технічних засобів*	16.04.2020	
8.	*Розробка архітектури програмного забезпечення*	23.04.2020	
9.	*Розробка програмного забезпечення*	30.04.2020	
10.	*Налагодження програми*	07.05.2020	
11.	*Виконання графічних документів*	*14.05.2020*	
12.	*Оформлення пояснювальної записки*	*21.05.2020*	
13.	*Подання ДП на попередній захист*	*28.05.2020*	
14.	*Подання ДП рецензенту*	*06.06.2020*	
15.	*Подання ДП на основний захист*	*08.06.2020*	

Студент _____ Володимир ПИЛИПЕНКО
(підпис)

Керівник _____ Олена ХАЛУС
(підпис)

№з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-6323.045490.02.81	Пояснювальна записка	58	
3	А4	КПІ.ІП-6323.045490.03.91	Технічне завдання	11	
4	А4	КПІ.ІП-6323.045490.04.51	Програма та методика тестування	6	
5	А4	КПІ.ІП-6323.045490.05.34	Керівництво користувача	6	
6	А4	КПІ.ІП-6323.045490.06.13	Опис програми	17	
7	А3	КПІ.ІП-6323.045490.07.99	Схема структурна варіантів використань	1	
8	А3	КПІ.ІП-6323.045490.07.99	Схема структурна класів програмного забезпечення	1	
9	А3	КПІ.ІП-6323.045490.07.99	Креслення вигляду екранних форм	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-6323.045490.01.90		
Розробн.		Пилипенко В.О.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Халус О.А.					
Консульт						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-63	
Н/контр.		Ліщук К.І.					
В.о.зав.каф.		Павлов О.А.					

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 8 рисунків, 30 таблиць та 12 джерел – загалом 58 сторінок.

Об’єкт дослідження: програмне забезпечення аналізу помилок за результатами інтеграційних тестів.

Мета дипломного проєкту: підвищити швидкість розробки, поліпшити сприйняття звітності інтеграційного тестування для всіх користувачів, не залежно від ступеню знань в системі, надання менеджерам більш детальну інформацію для визначення або перевизначення пріоритету та терміновості нових і вже існуючих помилок, а також отримувати більш точних результатів тестів після внесення своїх змін до коду, навіть у випадках коли гілка являється нестабільною.

У першому розділі було проведено аналіз відомих технічних рішень та розроблені вимоги до програмного продукту.

У другому було розроблено архітектуру програмне забезпечення. Побудовано структурну схему класів та BPMN діаграму.

У третьому розділі було проведено тестування програмне забезпечення аналізу помилок за результатами інтеграційних тестів за розробленим планом тестування.

У четвертому розділі було зроблено детальний опис розгортання та користування розробленим програмним забезпеченням.

КЛЮЧОВІ СЛОВА: ІНТЕГРАЦІЙНЕ ТЕСТУВАННЯ, МИГАЮЧА ПОМИЛКА, ЗВІТ, ГРУПУВАННЯ

ABSTRACT

The explanatory note of the diploma project consists of four sections, 30 tables, 8 illustrations and 12 sources – totally 58 pages.

The object of study: error analysis software based on the results of integration tests.

The aim of the diploma project: increase the speed of development, improve the perception of integration testing reporting for all users, regardless of the level of knowledge in the system, provide managers with more detailed information to determine or redefine the priority and urgency of new and existing errors, and get more accurate test results after making changes in the code, even when the branch is unstable.

In the first section analyzes the known technical solutions and develops requirements for the software product.

In the second section the software architecture was described and developed. The block diagram of classes and the BPMN diagram are constructed.

In the third section the software of error analysis based on the results of integration tests was tested according to the developed testing plan.

In the fourth section provides a detailed description of the deployment and use of the developed software.

KEYWORDS: INTEGRATION TESTING, FLASHING ERROR, REPORT, GROUPING

Пояснювальна записка
до дипломного проєкту

на тему: Програмне забезпечення з аналізу результатів інтеграційних тестів
_____
_____

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CI (Continious integration) – неперервна інтеграція.

HTML (HyperText Markup Language) – стандартизована мова розмітки документів.

CLI (Command Line Interface) – текстовий інтерфейс користувача, в якому інструкції можна дати тільки введенням із клавіатури текстових рядків.

XML (Extensible Markup Language) – стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками.

equal – аналогічний.

nochange – незмінний.

replace – заміна.

insert – вставка.

delete – видалення.

ВСТУП

На сьогодні бізнес вимагає від інженерів програмного забезпечення максимально швидко та якісно виконувати поставленні перед ними завдання.

Для досягнення цих вимог бізнесу інженерів програмного забезпечення бути впевненим в своєму продукті та оптимізувати й автоматизувати максимально можливу частину тестування.

Актуальність теми: Під час інтеграційного тестування окремі модулі програми комбінуються та тестуються у взаємодії між собою, що надає впевненості у відповідності програмного забезпечення відносно передбачених вимог.

Проте аналіз результатів тестування може ускладнитись у випадку, коли система неперервної інтеграції не реалізована повністю. А саме - створення коментаря не потребує підтвердження позитивними результатами тестів. Основною причиною такої реалізації системи неперервної інтеграції являється великий розмір продукту та сильна зв’язність його компонентів (повний процес тестування у таких випадках займає більше 20 годин).

Також аналіз може ускладнити наявність помилок які “мигають”. Саме такі помилки найбільше уповільнюють процес аналізу інтеграційного тестування, оскільки причина помилки, зазвичай, не прозора і це означає, що для цієї помилки необхідно приділити набагато більше часу.

Отже, для якісного життєвого циклу програмного забезпечення яке потребує інтеграційне тестування, аналіз результатів цього тестування повинен бути максимально оптимізований і автоматизований.

Мета розробки: підвищити швидкість розробки, поліпшити сприйняття звітності інтеграційного тестування для всіх користувачів, не залежно від ступеню знань в системі, надання менеджерам більш детальну інформацію для визначення або перевизначення пріоритету та терміновості нових і вже існуючих помилок. Також метою роботи являється – отримувати більш точних результатів

тестів після внесення своїх змін до коду, навіть у випадках коли гілка являється нестабільною.

Завдання розробки: обробка файлів реєстрації інтеграційного тестування, аналіз кожної помилки, групування помилок, автоматичне визначення мигаючих помилок та аналіз результатів інтеграційних тестів різних гілок розробки та відносно попередніх тестувань. А також генерування детального звіту.

Практичне значення одержаних результатів: розроблено програмне забезпечення у вигляді CLI додатку, що надає можливість аналізувати результати інтеграційного тестування. Особливість програмного забезпечення полягає в можливості групуванні помилок та визначення мигаючих помилок.

Публікації: Пилипенко В.О. ОПТИМІЗАЦІЯ ЗНАХОДЖЕННЯ ВІДСОТКОВОГО ЗІСТАВЛЕННЯ ТЕКСТУ ШЛЯХОМ ВИКОРИСТАННЯ «ВІДСТАНІ ЛЕВЕНШТЕЙНА» // IV Всеукраїнська науково-практична конференція молодих вчених та студентів «Інформаційні системи та технології управління – ІСТУ-2020».

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

На сьогоднішній день однією з головних вимог до програмного продукту – можливість швидкого розширення. Проте будь які розширення не можна вважати успішними без підтвердження працездатності програмного продукту та його якості. Для відстеження якості програмного продукту використовується такий процес технічного дослідження як тестування програмного забезпечення.

Тестування поділяється на три рівні:

– модульне тестування – тестується найменш можливий для тестування компонент або частина компонента, наприклад, окремий клас або функція. Найчастіше тестування цього рівня виконується розробниками програмного забезпечення;

– інтеграційне тестування – це тестування програмного забезпечення, при якому, окремі модулі чи їх частини об'єднуються для тестування в групі. При наявності певного резерву часу на цій стадії тестування – ці тести проводиться ітераційно, з поступовим розширенням функціоналу модуля або підключенням наступного модуля. Тестування цього рівня виконується частіше за все системами безперервної інтеграції. Проте розробниками програмного забезпечення також виконують це тестування, переважно у випадках коли ведеться розробка у декількох компонентах одночасно;

– системне тестування – це тестування програмного забезпечення, що виконується на повністю інтегрованій сукупності модулів чи системи. Метою даного тестування являється перевірка відповідності усієї системи поставленим вимогам. Системне тестування виконується за методом тестування чорного ящика, і, тим самим, не вимагає знань про внутрішню роботу системи, до означає що тестуванням цього рівня може займатись розробник, спеціаліст з тестування і навіть кінцевий користувач.

1.2 Змістовний опис і аналіз предметної області

При реалізації програмного продукту на базі мікро-сервісної архітектури або просто наявності декількох компонентів які повинні взаємодіяти між собою – обов'язково потрібно реалізовувати інтеграційне тестування. Цей тип тестування дозволить бути впевнений в коректній взаємодії між компонентами.

Проте, результати інтеграційного тестування можна вважати найскладнішим для аналізу, оскільки:

– найчастіше таке тестування проводиться в автоматичному режимі;

– результат тестування мають сильну залежність на:

1) стан кожного з компонентів чи модулів;

2) завантаженість середовище в якому відбувається тестування;

3) швидкість передачі даних.

– розробники проводять інтеграційне тестування.

Також складність аналіз результатів інтеграційного тестування сильно підвищується при наявності так званих 'мигаючих' помилок.

Мигаючими помилок називають такі помилки, які відтворюються не постійно як в одній і ті ж версії програмного забезпечення, так і в різних.

Мигаючі помилки можуть сильно уповільнювати розробку, оскільки можуть виникати у гілках розробки які відділились від гілки, де було створено цю помилку. Що означає що для однієї й тієї ж помилки потрібно буде двічі витратити робочий час розробника.

Приклад мигаючої помилки зображений на рисунку 1.1.

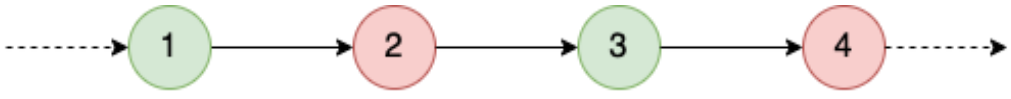


Рисунок 1.1 – Приклад мигаючої помилки

Детальний покроковий опис прикладу зображеного на рисунку 1.1:

– вносяться зміни до програмного забезпечення які приховують в собі одну або декілька мигаючих помилок, проте під час тестування помилка або помилки не проявили себе. На рисунку 1.1 – зелене коло з цифрою «1»;

– вносяться певні зміни до програмного забезпечення і під час тестування було виявлено нову помилку чи їх сукупність, проте внесені зміни не являються причиною цієї помилки або помилок. Причина – попередні зміни; На рисунку 1.1 – червоне коло з цифрою «2»;

– вносяться зміни до програмного забезпечення, які не стосуються помилки описаної в попередньому пункті, проте під час тестування не було виявлено описаної раніше помилки чи їх сукупність. На рисунку 1.1 – зелене коло з цифрою «3»;

– вносяться певні зміни до програмного забезпечення і під час тестування було виявлено таку ж помилку, що й в другому пункті(червоне коло з цифрою «2»). Проте, у цьому випадку, також внесені зміни не являються причиною помилки чи помилок. Причина – зміни описані в першому пункті; На рисунку 1.1 – червоне коло з цифрою «4».

Приклад поширення мигаючої помилки на нові гілки розробки зображений на рисунку 1.2.

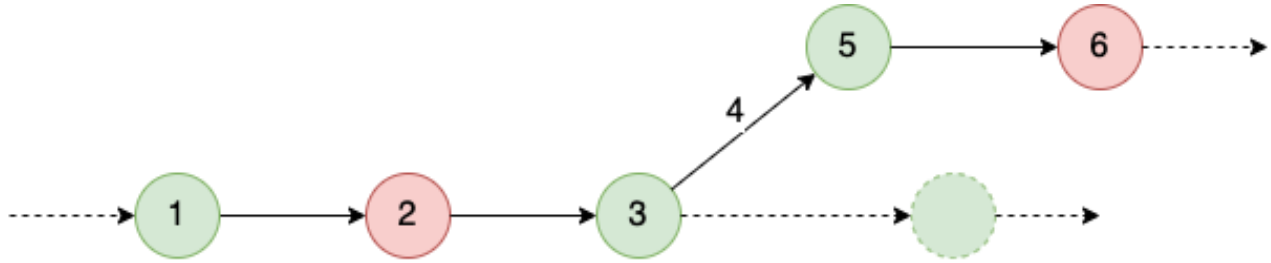


Рисунок 1.2 – Приклад поширення мигаючої помилки на нові гілках розробки

Детальний покроковий опис прикладу зображеного на рисунку 1.2:

– вносяться зміни до програмного забезпечення які приховують в собі одну або декілька мигаючих помилок, проте під час тестування помилка або помилки не проявили себе. На рисунку 1.2 – зелене коло з цифрою «1»;

– вносяться певні зміни до програмного забезпечення і під час тестування було виявлено нову помилку чи їх сукупність, проте внесені зміни не являються причиною цієї помилки чи помилок. Причина – попередні зміни; На рисунку 1.2 – червоне коло з цифрою «2»;

– вносяться зміни до програмного забезпечення, які не стосуються помилки описаної в попередньому пункті, проте під час тестування не було виявлено описаної раніше помилки чи їх сукупність. На рисунку 1.2 – зелене коло з цифрою «3»;

– створюється нова гілка розробки. На рисунку 1.2 – суцільна стрілка з цифрою «4»;

– вносяться зміни до програмного забезпечення, які не стосуються помилки описаної раніше. Під час тестування не було виявлено тієї помилки. На рисунку 1.2 – зелене коло з цифрою «5»;

– вносяться зміни до програмного забезпечення, які не стосуються помилки описаної раніше, проте під час тестування було виявлено помилки чи їх сукупність, які описані в першому пункті. На рисунку 1.2 – червоне коло з цифрою «6».

Основними причинами виникнення мигаючих помилок являється:

– побудова тестів на перервах;
– невірна конфігурація збірки;
– перевантаженість середовища тестування;
– помилки в архітектурі.

1.3 Аналіз успішних ІТ-проєктів

1.3.1 Аналіз відомих технічних рішень

Найпоширеніший спосіб аналізу результатів інтеграційного тестування – порівнювати лише результат кожного з тестів з попереднім, для того щоб визначити чи така помилка в цьому місці відбулась вперше. Такий спосіб аналізу надає лише поверхову інформацію та він не підходить при можливій наявності мигаючих помилок. Також такий спосіб аналізу має малу ефективність у випадках коли система неперервної інтеграції не реалізована повністю. Під неповною реалізацією мається на увазі, створення коментаря не потребує

підтвердження позитивними результатами тестів. Основною причиною такої реалізації системи неперервної інтеграції являється великий розмір продукту та сильна зв'язність його компонентів, у таких випадках повний процес тестування займає більше 20 годин.

Для покращення аналізу результатів інтеграційного тестування також використовують анотування при написанні тестів. Приклад анотування тесті на рисунку 1.3.

```
import org.junit.Test;
import org.junit.runner.RunWith;

import com.google.code.tempusfugit.concurrency.IntermittentTestRunner;
import com.google.code.tempusfugit.concurrency.annotations.Intermittent;

@RunWith(IntermittentTestRunner.class)
public class MultipleViaTempusFugit {
    @Test
    @Intermittent(repetition = 10)
    public void test() {
        // ...
    }
}
```

Рисунок 1.3 – Приклад анотування тесті

Цей спосіб може покрити частину випадків коли вимикають минаючі помилки, проте – не підтримується усім мовами і фреймворками для цих мов.

1.3.2 Аналіз відомих програмних продуктів

Основна маса систем аналізу помилок за результатів тестування інтегровані в системи тестування або в системи безперервної інтеграції.

Проте більшість відомих програмних продуктів аналізу помилок за результатів тестування проводять аналіз лише за результатом конкретного тесту окремо, не ведучи аналіз попередніх тестувань. Їхнє головне застосування – наглядна демонстрація результатів тестування, без глибокого аналізу.

Найбільш поширена на сьогоднішній день програмних продуктів що мають здатність аналізу помилок за результатами інтеграційних тестів являються:

– TeamCity разом з YouTrack. TeamCity – сервіс безперервної інтеграції та безперервної доставки, яких здійснює збірку запуск тестів та надає результати тестування після його закінчення. Надається звіт кожного тесту. У випадку коли помилки з різних тестів виникли через одну й теж причину і мають одне й теж повідомлення про помилку, в TeamCity вони відображатимуться як різні. YouTrack – сервіс для відстеження помилок знайдених під час тестування. Проте помилка створюється для кожного тесту своя. Також цей сервіс використовує анотування – як єдиний спосіб визначення мигаючих помилок;

– GitLab – також являється сервіс безперервної інтеграції та безперервної доставки. Проте на відміну від TeamCity разом з YouTrack, він взагалі не підтримує таке поняття як мигаючі помилки, що і являється його основною відмінністю.

1.4 Аналіз вимог до програмного забезпечення

Програмного забезпечення повинно взаємодіяти з таким типом користувачів як розробник.

Розробник – це користувач, який може, в ручному режимі, запускати розроблене програмне забезпечення.

Також дане програмне забезпечення можна запускати в автоматичному режимі після завершення інтеграційного тестування, якщо розширити систему неперервної інтеграції додатковим сценарієм.

Результатом роботи програмного забезпечення – детальний звіт в форматі HTML.

1.4.1 Розроблення функціональних вимог

Програмне забезпечення передбачає наступні варіанти використання, які описані в таблицях 1.1 – 1.4.

Таблиця 1.1 - Варіант використання UC001

Номер	UC001
Назва	Ручний запуск аналізу відносно попередньої проаналізованої версії, зі збереженням результатів
Опис	Розробник, в ручному режимі, запускає програмне забезпечення для детального аналізу результатів інтеграційного тестування відносно попередньої проаналізованої версії. Результати аналізу зберігаються до бази даних
Учасники	Розробник
Передумови	Вдало закінчене тестування та наявні файли реєстрації тестування
Постумови	Результати були успішно проаналізовано та згенеровано звіт. Результати аналізу зберігаються до бази даних
Основний сценарій	Розробник запускає програмне після закінчення тестування. Вхідні параметри: - ідентифікатор тесту, який аналізується; - папка де знаходяться файл реєстрації тестування; - прапорець необхідності збереження результатів встановлюється – ‘True’.

Таблиця 1.2 - Варіант використання UC002

Номер	UC002
Назва	Ручний запуск аналізу відносно попередньої проаналізованої версії, без збереження результатів
Опис	Розробник, в ручному режимі, запускає програмне забезпечення для детального аналізу результатів інтеграційного тестування відносно попередньої проаналізованої версії. Результати аналізу не зберігаються до бази даних
Учасники	Розробник
Передумови	Вдало закінчене тестування та наявні файли реєстрації тестування
Постумови	Результати були успішно проаналізовано та згенеровано звіт. Результати аналізу не зберігаються до бази даних
Основний сценарій	Розробник запускає програмне після закінчення тестування. Вхідні параметри: - ідентифікатор тесту, який аналізується; - папка де знаходяться файл реєстрації тестування.

Таблиця 1.3 - Варіант використання UC003

Номер	UC003
Назва	Ручний запуск аналізу відносно обраної версії
Опис	Розробник, в ручному режимі, запускає програмне забезпечення для детального аналізу результатів інтеграційного тестування відносно обраної версії. Результати аналізу не зберігаються до бази даних

Продовження таблиці 1.3

Учасники	Розробник
Передумови	Вдало закінчене тестування та наявні файли реєстрації тестування
Постумови	Результати були успішно проаналізовано та згенеровано звіт
Основний сценарій	Розробник, в ручному режимі, запускає програмне після закінчення тестування. Вхідні параметри: - ідентифікатор тесту, який аналізується; - шлях до папки, де знаходяться файл реєстрації тестування; - версія відносно якої аналізуються результати.

Таблиця 1.4 - Варіант використання UC004

Номер	UC004
Назва	Запуск аналізу за допомогою системи неперервної інтеграції
Опис	Розробник запускаю тестування за допомогою системи неперервної інтеграції, яка в свою чергу, в автоматичному режимі, запускає програмне забезпечення для детального аналізу результатів інтеграційного тестування відносно попередньої проаналізованої версії. Результати аналізу зберігаються до бази даних
Учасники	Розробник
Передумови	Вдало закінчене тестування та наявні файли реєстрації тестування

Продовження таблиці 1.4

Постумови	Результати були успішно проаналізовано та згенеровано звіт.
Основний сценарій	Розробник, в ручному режимі, тестування за допомогою системи неперервної інтеграції, яка в свою чергу запускає програмне забезпечення для аналізу результатів тестів після закінчення тестування. Вхідні параметри: - ідентифікатор тесту, який аналізується; - шлях до папки де знаходяться файл реєстрації тестування; - прапорець необхідності збереження результатів встановлюється – ‘True’.

На основі проаналізованої моделі варіантів використання була побудована схема структурна варіантів використання. Дану схему можна побачити на рисунку 1.4.

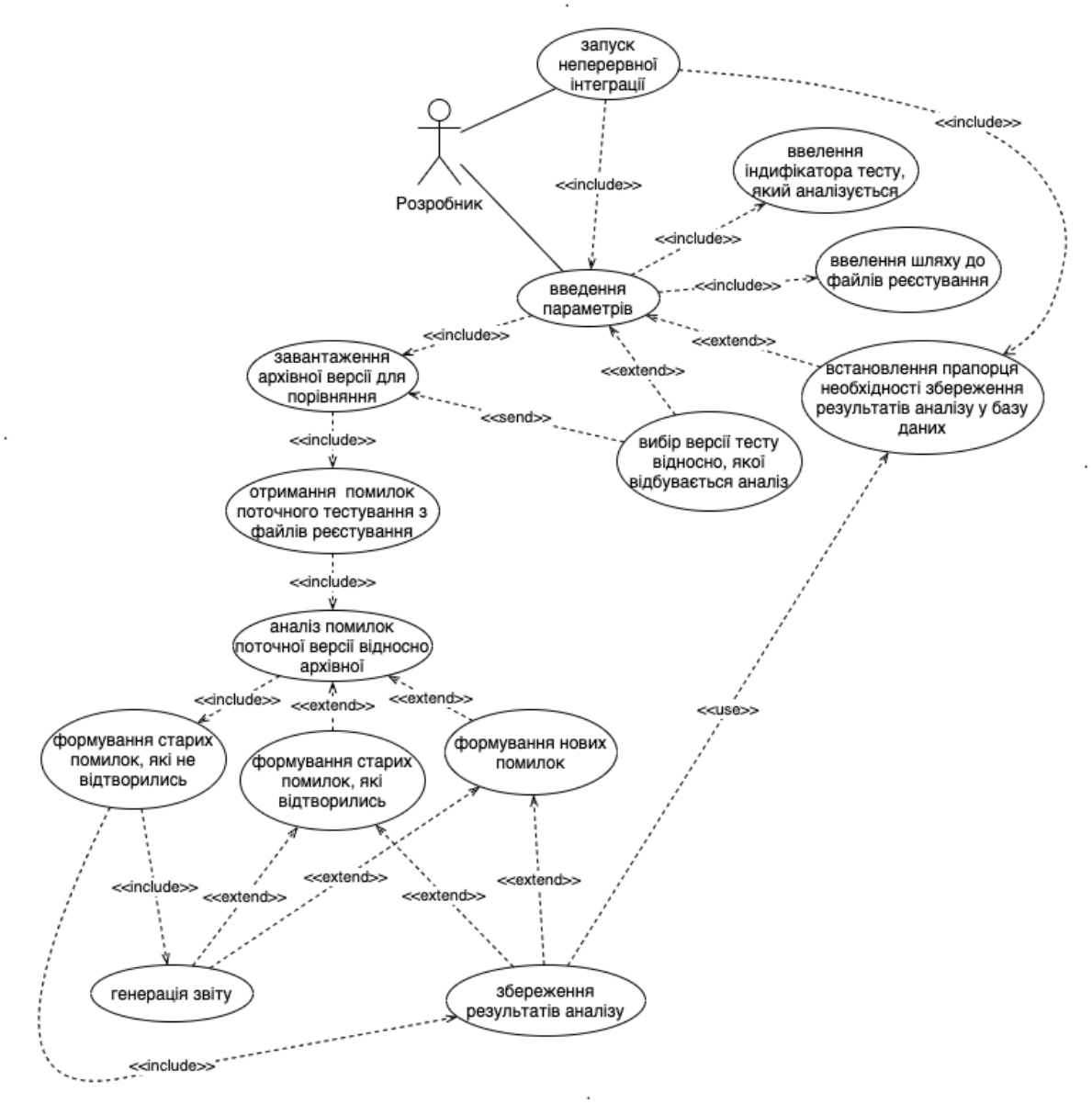


Рисунок 1.4 – Схема структурна варіантів використання

Функціональні вимоги до програмного забезпечення детально описано в таблицях 1.5 – 1.16.

Таблиця 1.5 - Опис функціональної вимоги REQ001

Номер	REQ001
Назва	Введення ідентифікатор тесту, який аналізується.
Опис	Вводиться версії інтеграційного тесту, як вхідний параметр, для ідентифікатор програмного забезпечення, яке буде аналізуватись.

Таблиця 1.6 - Опис функціональної вимоги REQ002

Номер	REQ002
Назва	Введення шляху до файлів реєстрації тестування.
Опис	Вводиться джерела файлів реєстрації тестування для аналізу, як вхідний параметр.

Таблиця 1.7 - Опис функціональної вимоги REQ003

Номер	REQ003
Назва	Встановлення прапорця необхідності збереження результатів аналізу в базу даних.
Опис	Позначається необхідність збереження результатів аналізу в базу даних, як вхідний параметр.

Таблиця 1.8 - Опис функціональної вимоги REQ004

Номер	REQ004
Назва	Введення ідентифікатора тесту, відносно якого відбувається аналіз.
Опис	Вводиться версії інтеграційного тесту, як вхідний параметр, для ідентифікатор програмного забезпечення, яке буде аналізуватись.

Таблиця 1.9 - Опис функціональної вимоги REQ005

Номер	REQ005
Назва	Завантаження архівної версії для порівняння.
Опис	Завантажується необхідна версія результатів аналізу з бази даних для порівняння.

Таблиця 1.10 - Опис функціональної вимоги REQ006

Номер	REQ006
Назва	Отримання помилок поточного тестування з файлів реєстрування.
Опис	Файли реєстрації інтеграційного тестування зчитаються та аналізуються, в результаті чого отримуються всі помилки які були виявленні під час цього тестування.

Таблиця 1.11 - Опис функціональної вимоги REQ007

Номер	REQ007
Назва	Аналіз помилок поточної версії відносно архівної.
Опис	Відбувається аналіз помилок які були виявлені під час поточного тестування відносно результатів попереднього тестування.

Таблиця 1.12 - Опис функціональної вимоги REQ008

Номер	REQ008
Назва	Формування старих помилок, які не відтворились.
Опис	За результатами аналізу формується вибірка старих помилок, які не відтворились під час тестування що аналізується.

Таблиця 1.13 - Опис функціональної вимоги REQ009

Номер	REQ009
Назва	Формування старих помилок, які відтворились.
Опис	За результатами аналізу формується вибірка старих помилок, які відтворились під час тестування що аналізується.

Таблиця 1.14 - Опис функціональної вимоги REQ010

Номер	REQ010
Назва	Формування нових помилок.
Опис	За результатами аналізу формується вибірка нових помилок які відтворились під час тестування що аналізується.

Таблиця 1.15 - Опис функціональної вимоги REQ011

Номер	REQ011
Назва	Збереження результатів аналізу.
Опис	Зберігаються результати аналізу в базу даних.

Таблиця 1.16 - Опис функціональної вимоги REQ012

Номер	REQ012
Назва	Генерування звіту з результатами аналізу
Опис	Після закінчення аналізу, автоматично генерується детальний звіт у форматі HTML.

Взаємозв’язки між варіантами використання та вимогами програмного забезпечення відображені на рисунку 1.5.

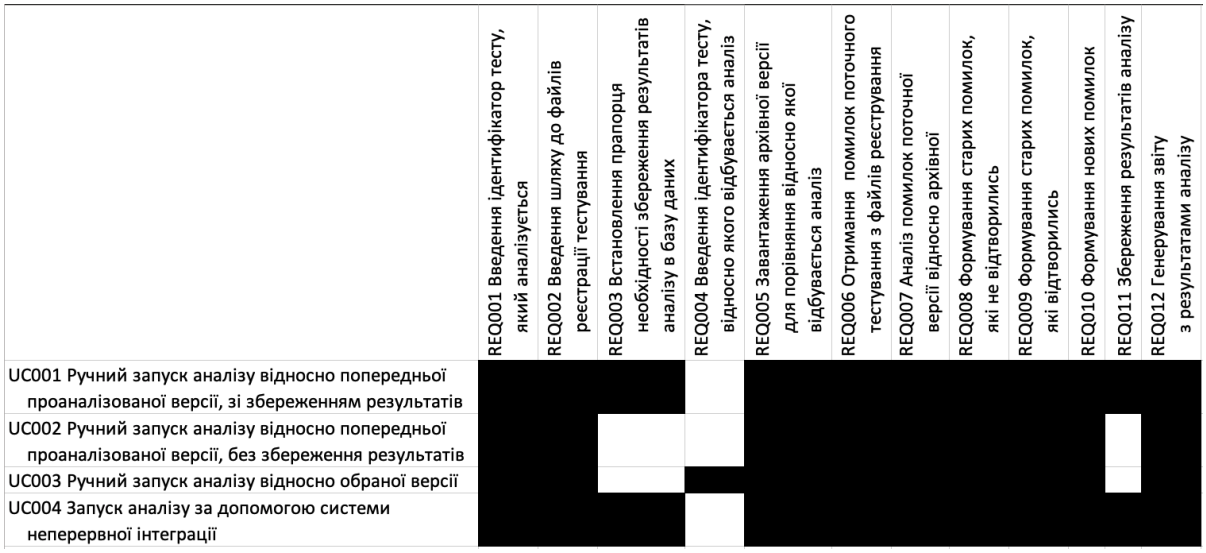


Рисунок 1.5 - Матриця залежності між варіантами використання та вимогами програмного забезпечення

1.4.2 Розроблення нефункціональних вимог

Була проведена розробка нефункціональних вимог до програмного забезпечення. Виявлені нефункціональні вимоги описані в таблицях 1.17 – 1.21.

Таблиця 1.17 – Опис нефункціональної вимоги NFR001

Номер	NFR001
Назва	Підтримка версій Python
Опис	Програмне забезпечення повинен підтримувати версії Python 3.6 та вище

Таблиця 1.18 – Опис нефункціональної вимоги NFR002

Номер	NFR002
Назва	Підтримка версій MongoDB
Опис	Програмне забезпечення повинен підтримувати версії MongoDB 4.2 та вище

Таблиця 1.19 – Опис нефункціональної вимоги NFR003

Номер	NFR003
Назва	Підтримка версій бібліотеки python-Levenshtein
Опис	Програмне забезпечення повинен підтримувати версії python-Levenshtein 0.12 та вище

Таблиця 1.20 – Опис нефункціональної вимоги NFR003

Номер	NFR003
Назва	Швидкодія
Опис	Час аналізу результатів тестування не повинен перевищувати 60 секунд.

Таблиця 1.21 – Опис нефункціональної вимоги NFR005

Номер	NFR005
Назва	Кросплатформенність
Опис	Програмне забезпечення має функціонувати на операційних системах: - Windows 10 1607 x64 та вище; - Ubuntu 16.04 x64 та вище; - MacOS Catalina та вище.

1.4.3 Постановка комплексу завдань модулю

Згідно з вище перечисленими функціональними та нефункціональними вимогами встановимо комплекс завдань, що повинен вирішувати розроблюване програмне забезпечення.

Метою для створення даної роботи являється:

– спрощення аналізу помилок для розробників та інженерів із забезпечення якості;

– покращення розуміння пріоритету та терміновості нових і вже існуючих помилок для менеджера;

– отримувати більш точних результатів тестів після внесення своїх змін до коду, навіть у випадках нестабільної гілки.

Для досягнення поставленої мети, програмне забезпечення повинно вирішувати наступні задачі:

– автоматичне визначення мигаючих помилок на базі попередніх результатів тестів;

– аналіз результатів інтеграційних тестів різних гілок розробки;

– можливість аналізу результатів інтеграційних тестів відносно попередніх результатів тестів;

– групування результатів за помилками;

– генерувати детальний звіт.

1.5 Математичне забезпечення

Визначення причини помилки повинно відбуватись за допомогою аналізу повідомлення цієї помилки. Для ефективного порівняння повідомлень помилок було обрано – «відстань Левенштейна».

Відстань Левенштейна – це міра, яка являє собою мінімальну кількість операцій вставки, видалення та заміни, які необхідні для перетворення одної послідовності символів в іншу. Ця міра дозволяє визначити «схожість» двох рядків.

Для розрахунку відстані Левенштейна частіше за все використовується простий алгоритм, в якому застосовується матриця розміром (M+1)*(N+1), де M і N - довжини порівнюваних рядків. Окрім цього вартість операцій вилучення, заміни та вставки дорівнює одиниці. Конструювання матриці виконується за формулою

$$d(S_1, S_2) = D(M, N), \text{де} \tag{1.1}$$

S_1 і S_2 – два рядка, з довжинами M та N відповідно;

$$D(i,j) = \begin{cases} 0, \text{при } i = 0, j = 0 \\ i, \text{при } i > 0, j = 0 \\ j, \text{при } i = 0, j > 0 \\ \min\begin{cases} D(i,j-1)+1, \\ D(i-1,j)+1, \\ D(i-1,j-1)+m(S_1[i],S_2[j]) \end{cases}, \text{при } i > 0, j > 0 \end{cases}, \text{ де} \quad (1.2)$$

m(a, b) дорівнює 0, якщо a = b, інакше 1.

Відстань Левенштейна має кілька простих верхніх і нижніх меж. До них належать:

– різниця розмірів двох рядки;

– не є більшою довжини найдовшого рядка;

– дорівнює 0, тоді і лише тоді, коли рядки рівні.

На детальному прикладі, розглянемо покрокове знаходження відстані Левенштейна між словами «субота» та «неділя». Приклад представлений в таблиці 1.22.

Таблиця 1.22 – Приклад знаходження відстані Левенштейна між словами «субота» та «неділя»

		з	**о**	**л**	**о**	**т**	**о**
	0	1	2	3	4	5	6
к	1	1	2	3	4	5	6
о	2	2	1	2	3	4	5
п	3	3	2	2	3	4	5
и	4	4	3	3	3	4	5
т	5	5	4	4	5	3	4
о	6	6	5	5	5	4	3

Отже, відстані Левенштейна між словами «субота» та «неділя» дорівнює трійці.

Відстань Левенштейна — ефективна міра для визначення «схожості» двох рядків. Даний підхід може бути застосований для великих рядків (близько 10^5

символів, тобто фактично для текстів) при одержанні не тільки оцінки «схожості», а й послідовності змін для перекладу одного рядка в іншу.

1.6 Висновки по розділу

У першому розділі була проведено визначення постановки задачі та освітлення основних проблем.

Були розглянуті успішні рішення наближених задач і було проведено порівняння цих рішень з розроблюваного програмного продукту та виявлені певні переваги та недоліки.

Також була проведено аналіз вимог до програмного продукту, а саме розроблені функціональні та нефункціональні вимог. І на останок, було встановлено комплекс завдань для розроблюваної системи згідно функціональних та нефункціональних вимог.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Перед створенням програмного продукту необхідно проаналізувати та спроєктувати бізнес-процеси. Однією з кращих методологій для проєктування бізнес-процесів являється BPMN.

BPMN – це графічне зображення для конкретизації бізнес-процесів у моделі бізнес-процесів.

На рисунку 2.1 можна побачити детальну BPMN-діаграма аналізу результатів інтеграційних тестів, що являється основним бізнес-процесом програмного забезпечення.

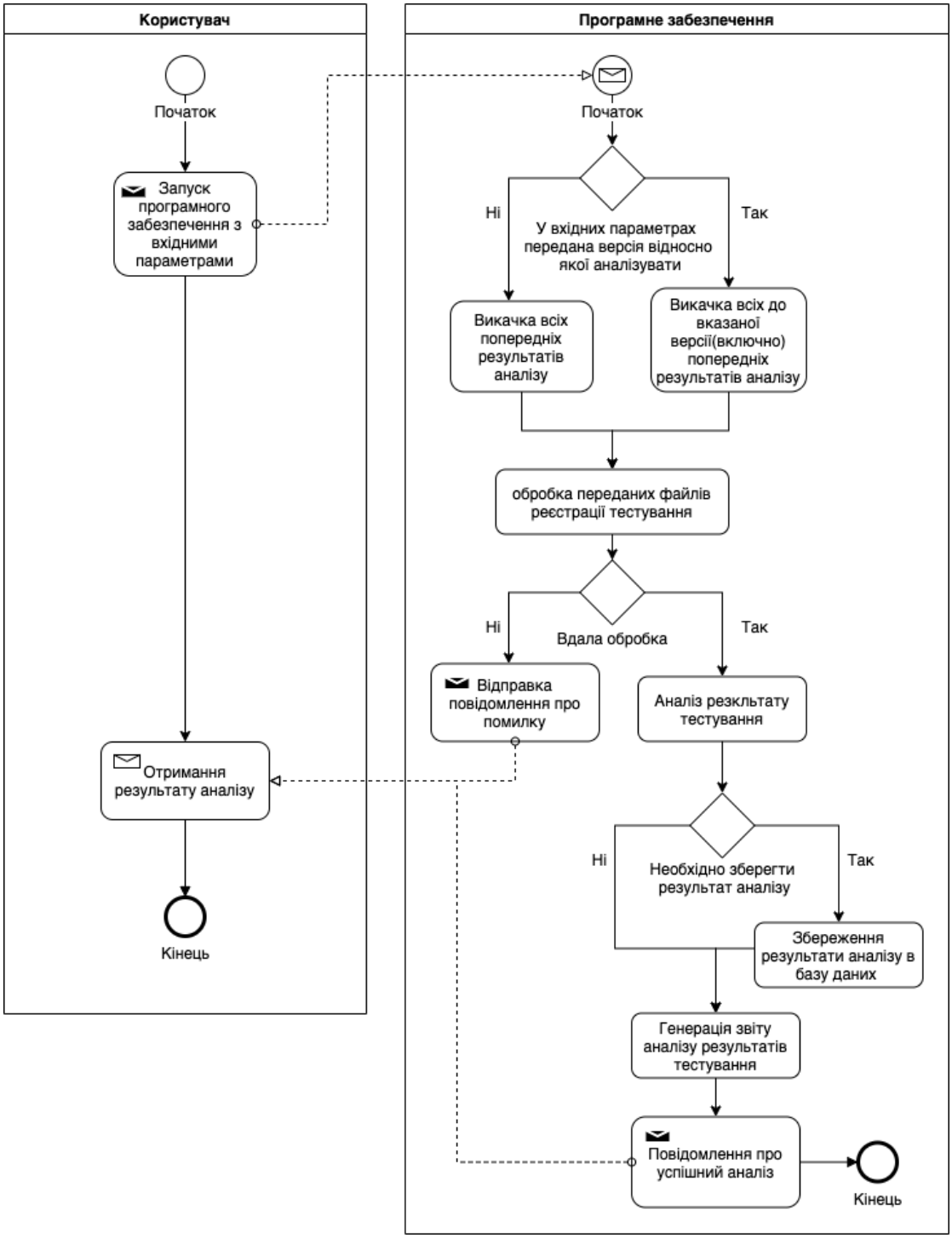


Рисунок 2.1 – Схема бізнес-процесу аналізу результатів інтеграційних тестів

Також під час моделювання програмного забезпечення була розроблена схема бази даних, що дозволяє зберігати усю інформацію про версії аналізу та їх результати. В процесі моделювання було обрано використовувати нереляційну базу даних. Схема бази даних представлена на рисунку 2.2.

Рисунок 2.2 – Схема бази даних

Таблиця 2.1 - Опис об’єкту Test

Поле	Тип	Опис
file	string	Назва файлу, де відбулася помилки.
test_suite	string	Назва набору тестів, де відбулася помилки.
test_case	string	Назва галузі тестів, де відбулася помилки.

Таблиця 2.2 - Опис об’єкту Failure

Поле	Тип	Опис
tests	Array<Test>	Масив з об’єктів Test, у яких однакова причина помилки.
count	int	Кількість відтворення помилки.
first_version	string	Версія інтегрованого тестування, де вперше виникала помилка.

Продовження таблиці 2.2

all_versions	Array<string>	Всі версії інтегрованого тестування, де виникала помилка.

Таблиця 2.3 - Опис об’єкту Version

Поле	Тип	Опис
failures	Map<string, Array<Failure>>	Колекція формату ключ – значення, де ключем виступає помилка, тоді як значенням – масив з об’єктів Failure.
version	string	Версія інтегрованого тестування.

Таблиця 2.4 - Опис об’єкту Versions

Поле	Тип	Опис
versions	Array<Version>	Масив з об’єктів Version.

2.2 Архітектура програмного забезпечення

Під час розробки архітектури програмного забезпечення була побудовано детальну діаграму класів програми. Ця діаграма зображена на рисунку 2.3.

Рисунок 2.3 – Схема структурна класів програмного забезпечення

На таблицях 2.5 та 2.6 детально описано класи програмного продукті та їх функції.

Таблиця 2.5 - Опис класів

Клас	Опис
Runner	Базовий клас, що викликає послідовно інші класи для аналізу результатів інтеграційного тестування.
ArgsManager	Клас для роботи з вхідними параметрами. Реалізований за допомогою фреймворку argparse.
ReportGenerator	Клас для генерування звіту після вдалого аналізу результатів.
ParserLogsManager	Клас для отримання інформації з файлів реєстрації тестування.
DBManager	Клас для роботи з не реляційною базою даних MongoDB.

Продовження таблиці 2.5

Analyzer	Клас для отримання інформації з файлів реєстрації тестування.
Test	Клас для збереження інформації про певний тест і роботу з цією інформацією.
Failure	Клас для групування тестів та їх статистику. Також забезпечує роботу з цією інформацією.
Error	Клас для збереження інформації про певну помилку і роботу з цією інформацією.

Таблиця 2.6 - Опис методів класів

Клас	Метод	Опис
Main	__main__	Точка входу в програму. Вхідні параметри відсутні.
Runner	__init__	Конструктор. Вхідні параметри відсутні.
Runner	run	Точка входу в програму. Вхідні параметри відсутні.
ArgsManager	__init__	Конструктор. Обробка переданих в програмне забезпечення атрибутів. У разі некоректності вхідних параметрів – виводить відповідне повідомлення і зупиняє виконання програмного продукту. Вхідні параметри відсутні.
ArgsManager	get_version	Отримання обов’язкового вхідного параметру, котрий відповідає версії тесту який аналізується. Вхідні параметри відсутні.

Продовження таблиці 2.6

ArgsManager	get_path_logs	Отримання обов’язкового вхідного параметру, котрий вказує на шлях до директорії в якій знаходяться файли реєстрації тестування. Вхідні параметри відсутні.
ArgsManager	get_version4analyze	Отримання не обов’язкового вхідного параметру, котрий відповідає версії тесту відносно якої буде відбуватися аналіз. У разі відсутності параметру, функція повертає ‘last’ Вхідні параметри відсутні.
ArgsManager	get_path2save	Отримання не обов’язкового вхідного параметру, котрий вказує на шлях, куди необхідно згенерувати звіт аналізу. У разі відсутності параметру, функція повертає шлях, де знаходиться програмне забезпечення. Вхідні параметри відсутні.
ArgsManager	get_need_save	Отримання не обов’язкового вхідного параметру, котрий відповідає на необхідність збереження результатів аналізу до бази даних. У разі відсутності параметру, функція повертає ‘False’ Вхідні параметри відсутні.
DBManager	__init__	Конструктор. Вхідні параметри відсутні.

Продовження таблиці 2.6

DBManager	connection	Під’єднання до бази даних. У разі невдачі – виводить відповідне повідомлення і зупиняє виконання програмного продукту. Вхідні параметри: - host_name – string – назва вузла з базою даних, за замовчуванням ‘localhost’; - port – string – назва порту для під’єднання до базою даних, за замовчуванням ‘27017’.
DBManager	get_old_analyze	Отримання інформації попередніх аналізів результатів інтеграційних тестів. У разі невдачі – виводить відповідне повідомлення і зупиняє виконання програмного продукту. Вхідні параметри: - version – string – назва версії попереднього аналізу. Цей параметр може бути відсутнім, тоді береться остання проаналізована версія.
DBManager	save_analyze	Збереження інформації аналізу результату інтеграційних тестів. У разі невдачі – виводить відповідне повідомлення. Вхідні параметри: - version – string – назва версії аналізу; - errors – map<Error, collection<Failure>> – результат аналізу.

Продовження таблиці 2.6

ParserLogsManager	__init__	Конструктор. Ініціалізація початкових значень атрибутів. Вхідні параметри відсутні.
ParserLogsManager	check_logs	Отримання інформації з файлів реєстрації тестування та перетворення їх у зручний для роботи формат, тобто колекцію об’єктів-python, а саме collection<Error>. Вхідні параметри: - version – string – назва версії аналізу; - path – string – шлях до файлів реєстрації тестування; - gERRORS – map<Error, collection<Failure>> – шлях до файлів реєстрації тестування.
ParserLogsManager	get_fail_message	Обробка усього повідомлення помилки та отримання причини помилки. Вхідні параметри: - message – string – повідомлення помилки.

Продовження таблиці 2.6

ParserLogsManager	add_failure_preliminary	Додавання помилки в колекцію для подальшої обробки. Вхідні параметри: - file – string – назва файлу, де відбулася помилки; - test_suite_name – string – назва набору тестів**,** де відбулася помилки; - test_case_name – string – назва галузі тестів**,** де відбулася помилки; - failure – string – помилка; - failed_version – string – версія тесту; - gERRORS – map<Error, collection<Failure>> – колекція усіх помилок.
Analyzer	__init__	Конструктор. Ініціалізація початкових значень атрибутів. Вхідні параметри відсутні.

Продовження таблиці 2.6

Analyzer	analyze	Проаналізувати помилки які були отримані з файлів реєстрації тестування. У разі невдачі – виводить відповідне повідомлення і зупиняє виконання програмного продукту. Вхідні параметри: - version – string – назва версії аналізу; - new_errors – map<Error, collection<Failure>> – колекція помилок які були отримані з файлів реєстрації тестування; - old_errors – map<Error, collection<Failure>> – колекція історичних помилок.
ReportGenerator	__init__	Конструктор. Ініціалізація початкових значень атрибутів. Вхідні параметри відсутні.
ReportGenerator	generate_report	Створення звіту після аналізу результату інтеграційного тестування. У разі невдачі – виводить відповідне повідомлення. Вхідні параметри: - data_save_path – string – місце куди зберегти звіт, за замовчуванням являється кореневою папкою програмного продукту; - version – string – назва версії аналізу; - all_errors – map<Error, collection<Failure>> – колекція всіх помилок.

Продовження таблиці 2.6

ReportGenerator	save_html	Збереження детального звіту в форматі HTML документу. Вхідні параметри: - data_save_path – string – місце куди зберегти звіт; - html_data – string – HTML документ в форматі рядка.
ReportGenerator	add_style	Генерування стилю для звіту. Вхідні параметри відсутні.
ReportGenerator	create_new_table	Формування таблиці в форматі HTML документ для нових помилок. Вхідні параметри: - new_errors – map<Error, collection<Failure>> – колекція нових помилок.
ReportGenerator	create_replay_table	Формування таблиці в форматі HTML документ для помилок, які відтворились. Вхідні параметри: - replay_errors – map<Error, collection<Failure>> – колекція помилок, які відтворились.
ReportGenerator	create_old_table	Формування таблиці в форматі HTML документ для помилок, які не відтворились. Вхідні параметри: - replay_errors – map<Error, collection<Failure>> – колекція помилок, які не відтворились.

Продовження таблиці 2.6

Test	__init__	Конструктор. Ініціалізація початкових значень атрибутів. Параметри: - file – string – назва файлу; - test_suite – string – назва вибірки тестів; - test_case – string – назва ящику.
Test	__eq__	Функція порівняння з іншим об’єктом. Параметри: - other – Object – інший об’єкт.
Test	__hash__	Функція знаходження хешу від об’єкту. Параметри відсутні.
Test	__str__	Функція отримання рядку з об’єкту. Параметри відсутні.
Test	__lt__	Функція перевірки на менше з іншим об’єктом. Параметри: - other – Object – інший об’єкт.
Failure	__init__	Конструктор. Ініціалізація початкових значень атрибутів. Параметри: - tests – collection<Test> – колекція всіх помилок; - count – int – кількість відтворень; - first_version – string – версія першого відтворення; - all_versions – collection<string> – колекція всіх версій відтворення.

Продовження таблиці 2.6

Failure	__eq__	Функція порівняння з іншим об’єктом. Параметри: - other – Object – інший об’єкт.
Failure	__str__	Функція отримання рядку з об’єкту. Параметри відсутні.
Failure	count_up	Функція для інкримінації кількості відтворень та додавання нової версії відтворень. Параметри: - new_version – string – нова версія відтворень.
Failure	add_test	Функція додавання нового тесту. Параметри: - new_test– Test – новий тест.
Error	__init__	Конструктор. Ініціалізація початкових значень атрибутів. Параметри: - msg – string – повідомлення помилки.
Error	__eq__	Функція порівняння з іншим об’єктом. Параметри: - other – Object – інший об’єкт.
Error	__hash__	Функція знаходження хешу від об’єкту. Параметри відсутні.
Error	apply_transformations	Функція обробки повідомленя. Параметри: - msg – string – повідомлення помилки.

Продовження таблиці 2.6

Error	messages_definitely_different	Функція для швидкої перевірки на ймовірність схожості повідомлень. Використовується для оптимізації. Параметри: - msg1 – string – перше повідомлення; - msg2 – string – друге повідомлення.
Error	msg_no_ignored_chars	Функція видалення символів які потрібно ігнорувати під час порівняння повідомлень, для підвищення коректності їх порівняння. Параметри: - msg – string – повідомлення помилки.
Error	msg_no_queries	Функція обробки запиту, як причина помилки. Параметри: - msg – string – повідомлення помилки.
Error	msg_no_prefix	Функція видалення непотрібних префіксів в повідомлені помилки. Параметри: - msg – string – повідомлення помилки.
Error	msg_no_GUID	Функція видалення індифікатора в повідомлені помилки. Параметри: - msg – string – повідомлення помилки.

2.3 Конструювання програмного забезпечення

Продукт має тільки один варіантів використання – запуск програмного забезпечення з введенням необхідних вхідних параметрів та отримання результатів аналізу в форматі HTML сторінки.

До обов’язкових вхідних параметрів налужать:

– ідентифікатор тесту, який аналізується;

– шлях до папки, де знаходяться файл реєстрації тестування.

До необов’язкових вхідних параметрів налужать:

– ідентифікатор тесту, відповідно якої проводитиметься аналіз;

– прапорець, необхідності збереження результату аналізу до бази даних;

– шлях, куди необхідно зберегти звіт аналізу.

2.4 Аналіз безпеки даних

Не потребує підвищеної безпеки оскільки використовується лише для аналізу даних та не використовує під’єднання до інтернету.

2.5 Висновки по розділу

В другому розділі виконано моделювання та аналіз розробленого програмного продукту, а також було описано архітектуру цього продукту в діаграмі класів. Також за допомогою BPMN діаграми проілюстровано бізнес-процеси програмного забезпечення.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Якість програмного забезпечення вважається здатність програмного продукту при заданих умовах задовольняти встановленим або передбачуваним потребам які заздалегідь визначенні.

З огляду на функціональні вимоги, визначимо план тестування який описано в таблиці 3.1.

Таблиця 3.1 - План тестування

№	Розділ	Елементи
1	Вступ	У цій таблиці описується тестування програмного забезпечення аналізу помилок за результатами інтеграційних тестів
2	Функціонал, що підлягає тестуванню	– завантаження файлів; – читання файлів; – групування помилок; – отримання інформації з бази даних; – збереження інформації в базу даних.
3	Функціонал, що не підлягає тестуванню	

№	Розділ	Елементи
4	Тестові елементи	– завантаження файлів реєстрації тестування; – читання файлів реєстрації тестування; – групування помилок за їх повідомленням помилки; – отримання певної, заздалегідь підготовленої, інформації з бази даних; – збереження інформації в базу даних.
5	Тестові підходи	Тестування буде проведено способом black-box, на основі функціонального підходу
6	Критерій успішно/не успішно	– всі вимоги повинні бути покриті позитивними тестами та негативними тестами; – не повинно бути жодної критичної помилки.
7	Критерій призупинення/відновлення	Програмне забезпечення не запускається або запускається.
8	Вимоги до середи	Повинен бути пристрій (комп'ютер, ноутбук, серверна машина), з операційною системою Windows/MacOS/Ubuntu, а також необхідна клавіатура.
9	Вимоги до навичок персоналу	Необхідні середні або високі навички роботи з комп’ютером. Також навички роботи з CLI програмним забезпеченням.

№	Розділ	Елементи
10	Задачі тестування	– пошук дефектів, що викликають відмову додатку; – створення сценаріїв для перевірки функціоналу програми; – створення звіту про виконану роботу.
11	Супровід тестування	– документ тестового плану; – тестові сценарії; – журнал помилок та виконання; – звіти про помилки та коригувальні дії.
12	Розклад тестування	Для кожного з тестів виділяється дві години робочого часу
13	Відповідальні	Уся відповідальність присвоюється автору тестового плану
14	Підтвердження	Етап тестування може затвердити виключно автора тестового плану
15	Ризики	При зменшені часу для тестування зростає ймовірність зниження якість тестування всього програмного продукту

3.2 Опис процесів тестування

Для модульного тестування було обрано фреймворком – pytest. Цей фреймворк спрощує створення простих та масштабованих тестів.

Визначимо наступні процеси тестування які описані в таблиці 3.2.

Таблиця 3.2 – Процес тестування

№	Процес	Назва	Етапи	Очікуваний результат	Реальний результат	Висновок
1	Вибір файлів	get_all_files	– Вибір всіх XML файлів, які знаходяться просторі для тестування	Відповідь містить шляхи до кожного з файлів	Відповідь містить шляхи до кожного з файлів	Успіх
2	Читання файлів реєстрації	read_all_files	– Читання всіх XML файлів, шляхи до яких було надано	Відповідь містить зміст файлів в форматі об’єкта-python	Відповідь містить зміст файлів в форматі об’єкта-python	Успіх
3	Групування помилок	fails_grouping	– Завантажити підготовлені данні – Надати данні на аналіз та групування	Відповідь містить згруповані, за повідомленням, помилки	Відповідь містить згруповані, за повідомленням, помилки	Успіх

Продовження таблиці 3.2

4	Отримання інформації з бази даних	get_db_info	– Отримання інформації з бази даних	Відповідь містить певну підготовлену інформацію яка зберігається в базі даних	Відповідь містить певну підготовлену інформацію яка зберігається в базі даних	Успіх
5	Запис інформації до бази даних	send_db_info	– Запис інформації до бази даних – Отримання інформації з бази даних	Відповідь містить певну інформацію яка була перед цим збережена до бази даних	Відповідь містить певну інформацію яка була перед цим збережена до бази даних	Успіх
6	Невдале читання файлів реєстрації	fail_read_files	– Спроба читання XML файлу, який являється некоректним – Обробка помилки	Отримання помилки про неможливість роботи з певним файлом реєстрації	Отримання помилки про неможливість роботи з певним файлом реєстрації	Успіх

3.3 Висновки до розділу

У третьому розділі був проведено тестування програмного забезпечення. Був створено детальний тестовий план. Також був детально описаний процес тестування за яким було проведене тестування. Тестування показало задовільну якість продукту.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Для роботи розробленого програмного продукту необхідно задовольнити наступні технічні вимоги:

– пристрій під управлінням операційних систем MacOS/Windows10/Ubuntu;

– об’єм оперативної пам’яті: 512 Мб і більше;

– об’єм вільного дискового простору: 512 Мб і більше.

Для запуску розробленого програмного продукту необхідно запустити файл виконання. Запуск файл виконання відбувається за допомоги команди:

python [шлях]/run.py [версія] [шлях_до_файлів] {-version4analyze [версія_відносно]} {-path2save[місце_збереження]} {-need_save[збереження]}, де:

– [шлях] – шлях до папки, в якій знаходиться розроблений програмний продукт;

– [версія] – обов’язковий вхідний параметр, ідентифікатор тесту, який аналізується;

– [шлях_до_файлів] – обов’язковий вхідний параметр, шляху до файлів реєстрації тестування;

– {-version4analyze [версія_відносно]} – необов’язковий вхідний параметр, де [версія_відносно] – ідентифікатора тесту, відносно якого відбувається аналіз;

– {-path2save [місце_збереження]} – необов’язковий вхідний параметр, де [місце_збереження] – шлях куди збережеться звіт після завершення аналізу. За замовчуванням – [шлях];

– {-need_save [збереження]} – необов’язковий вхідний параметр, де [збереження] – прапорця необхідності збереження результатів аналізу в базу даних. За замовчуванням – False.

Дане програмне забезпечення, разом з файлом виконання, додається до дипломного проєкту та також його можна отримати з репозиторію *https://github.com/VovaPylypenko/diploma*.

На комп’ютері, де планується запуск програмного забезпечення, необхідно щоб завчасно було встановлено Python версії 3.6 і вище та MongoDB версії 4.2 і вище. Також необхідно мати, завчасно, встановлені модулі та бібліотеки: *python-Levenshtein, re, xml, os, argparse, pymongo.*

4.2 Робота з програмним забезпеченням

Детальна та покрокова інструкція користування програмним забезпеченням наведено в керівництві користувача.

4.3 Висновки до розділу

У четвертому розділі було описано необхідні умови для запуску розробленого програмного забезпечення, також – роботи з ним.

ВИСНОВКИ

Виконання дипломного проєкту здійснювалось в декілька етапів. Кожний з етапів описаний у відповідному розділі.

В першому розділі було проведено аналіз проблеми роботи з інтеграційним тестуванням та розглянуто успішні рішення наближених задач. Також було проведено аналіз вимог до програмного продукту, а саме розроблені функціональні та нефункціональні вимог. І на останок, було встановлено комплекс завдань для розроблюваної системи згідно функціональних та нефункціональних вимог.

В другому розділі було виконано моделювання та аналіз розробленого програмного продукту, а також було описано архітектуру цього продукту в діаграмі класів. Також за допомогою BPMN діаграми проілюстровано бізнес-процеси програмного забезпечення.

В третьому розділі було створено детальний тестовий план і проведено тестування програмного забезпечення. Також було детально описаний процес тестування за яким було проведене тестування. Тестування показало задовільну якість продукту.

В останньому розділі було описано необхідні умови для запуску розробленого програмного забезпечення, також було надано інструкцію користуванням даним програмним продуктом.

Отже, розроблений програмний продукт повністю задовольняє всі поставлені вимоги.

ПЕРЕЛІК ПОСИЛАНЬ

1) Пилипенко В.О. ОПТИМІЗАЦІЯ ЗНАХОДЖЕННЯ ВІДСОТКОВОГО ЗІСТАВЛЕННЯ ТЕКСТУ ШЛЯХОМ ВИКОРИСТАННЯ «ВІДСТАНІ ЛЕВЕНШТЕЙНА» // IV Всеукраїнська науково-практична конференція молодих вчених та студентів «Інформаційні системи та технології управління – ІСТУ-2020». Секція кафедри автоматизованих систем обробки інформації і управління. Матеріали конференції. – 2020.

2) Відстань Левенштейна [Електронний ресурс] : // Wikipedia. – 2019. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Відстань_Левенштейна.

3) TeamCity documentation [Електронний ресурс] // JetBrains. – 2020. – Режим доступу до ресурсу: https://www.jetbrains.com/help/teamcity/teamcity-documentation.html.

4) YouTrack documentation [Електронний ресурс] // JetBrains. – 2020. – Режим доступу до ресурсу: https://www.jetbrains.com/youtrack/documentation/.

5) GitLab documentation [Електронний ресурс] // GitLab. – 2020. – Режим доступу до ресурсу: https://docs.gitlab.com/.

6) Python 3 documentation [Електронний ресурс] // Python. – 2020. – Режим доступу до ресурсу: https://docs.python.org/3/.

7) Levenshtein Python documentation [Електронний ресурс] // Antti Haapala. – 2014. – Режим доступу до ресурсу: https://pypi.org/project/python-Levenshtein/.

8) MongoDB documentation [Електронний ресурс] // MongoDB. – 2019. – Режим доступу до ресурсу: https://docs.mongodb.com/.

9) PyMongo documentation [Електронний ресурс] // Mike Dirolf. – 2019. – Режим доступу до ресурсу: https://pypi.org/project/pymongo/.

10) Python xml documentation [Електронний ресурс] // Python. – 2019. – Режим доступу до ресурсу: https://docs.python.org/3/library/xml.etree.elementtree.html.

11) Argparse documentation [Електронний ресурс] // Thomas Waldmann. – 2015. – Режим доступу до ресурсу: https://pypi.org/project/argparse/.

12) XML [Електронний ресурс] // Wikipedia. – 2019. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/XML.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації та управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ З АНАЛІЗУ РЕЗУЛЬТАТІВ ІНТЕГРАЦІЙНИХ ТЕСТІВ

Технічне завдання

КПІ.ІП-6323.045490.03.91

“ПОГОДЖЕНО”

Керівник проекту:

О.А. Халус

Нормоконтроль:

К.І. Ліщук

Виконавець:

В.О. Пилипенко

Київ – 2020 року

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Програмне забезпечення з аналізу результатів інтеграційних тестів

Галузь застосування: Проекти, що працюються в сфері розробки та тестування програмного забезпечення

Наведене технічне завдання поширюється на розробку ManagerResult КПІ.ІП-6323.045490.03.91, котра використовується для аналізу результатів інтеграційних тестів та призначена для підвищити швидкість розробки, поліпшити сприйняття звітності інтеграційного тестування для всіх користувачів, не залежно від ступеню знань в системі, надання менеджерам більш детальну інформацію для визначення або перевизначення пріоритету та терміновості нових і вже існуючих помилок, а також отримувати більш точних результатів тестів після внесення своїх змін до коду, навіть у випадках коли гілка являється нестабільною.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки ManagerResult є завдання на дипломне проектування, затверджене кафедрою автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім.Ігоря Сікорського).

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для детального аналізу результатів інтеграційного тестування.

Метою розробки є отримувати більш детальний та наглядний звіт після інтеграційного тестування.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1.1 Для користувача:

– отримувати детальний звіт після тестування;
– робити порівняння тестів різних версій;
– знаходити мигаючі помилки.

4.1.2 Розробку виконати на платформі Windows 10.

4.1.3 Додаткові вимоги:

– коректність файлів реєстрації тестування.

4.2 Вимоги до надійності

4.2.1 Передбачити контроль введення інформації.

4.2.2 Передбачити захист від некоректних дій користувача.

4.2.3 Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

4.3.1 Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.2 Обслуговування.

Не висуваються.

4.3.3 Обслуговуючий персонал.

Не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

4.4.1 Програмне забезпечення повинно функціонувати на серверах та комп’ютерах з операційною системою Windows/Ubuntu/MacOS.

4.4.2 Мінімальна конфігурація технічних засобів:

– тип процесору ……. Pentium;
– об‘єм ОЗП ……..….. 512 Мб;
– об‘єм вільного дискового простору ……. 512 Мб.

4.5 Вимоги до інформаційної та програмної сумісності

4.5.1 Програмне забезпечення повинно працювати під управлінням операційних систем Windows 10/Ubuntu/MacOS.

4.5.2 Вхідні дані повинні бути представлені в наступному форматі: шлях до XML-файлів реєстрації тестування та версія тестування.

4.5.3 Результати повинні бути представлені в наступному форматі: звіт в HTML форматі.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не пред‘являються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не пред'являються.

4.8 Спеціальні вимоги

Згенерувати установчу версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

5.2 Програмне забезпечення повинно мати довідникову систему.

5.3 У склад супроводжувальної документації повинні входити наступні документи.

5.3.1 Пояснювальна записка не менше ніж на 100 аркушах формату А4 (без додатків 5.3.2 - 5.3.4).

5.3.2 Технічне завдання.

5.3.3 Керівництво користувача.

5.3.4 Програма та методика тестування.

5.4 Графічна частина повинна бути виконана у формату А3, котрі включаються у якості додатків до пояснювальної записки:

5.4.1 Схема структурна варіантів використань.

5.4.2 Схема структурна класів програмного забезпечення.

5.4.3 Креслення вигляду екранних форм.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

Таблиця 6.1 – Стадії і етапи розробки

№	Назва етапу	Строк	Звітність
1	Вивчення рекомендованої літератури	19.03.2020	
2	Аналіз існуючих методів розв’язання задачі	26.03.2020	
3	Постановка та формалізація задачі	26.03.2020	Технічне завдання
4	Аналіз вимог до програмного забезпечення	02.04.2020	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5	Алгоритмізація задачі	02.04.2020	
6	Моделювання програмного забезпечення	09.04.2020	
7	Обґрунтування використовуваних технічних засобів	16.04.2020	
8	Розробка архітектури програмного забезпечення	23.04.2020	Схема структурна класів програмного забезпечення
9	Розробка програмного забезпечення	30.04.2020	Тексти програмного забезпечення
10	Налагодження програми	07.05.2020	Програма та методика тестування
11	Виконання графічних документів	14.05.2020	Графічний матеріал проекту

Продовження таблиці 6.1

12	Оформлення пояснювальної записки	21.05.2020	Пояснювальна записка проекту
13	Подання ДП на попередній захист	06.06.2020	
14	Подання ДП рецензенту	07.06.2020	
15	Подання ДП на основний захист	19.06.2020	

7 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

7.1. Види випробувань

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації та управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

Програмне забезпечення з аналізу результатів інтеграційних тестів

Програма та методика тестування

КПІ.ІП-6323.045490.04.51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ О.А. Халус

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____В.О. Пилипенко

Київ – 2020 року

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Програмне забезпечення з аналізу результатів інтеграційних тестів, який являє собою інтерфейс командного рядка, створений з використанням мови програмування Python з використанням ряду зовнішніх бібліотек, таких як python-Levenshtein, re, xml, os, argparse, pymongo.

2 МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

– функціональна працездатність елементів сторінок інтерфейсу командного рядка;

– коректне зчитування вхідних параметрів;

– відповідність форматів та протоколів файлів реєстрації тестування;

– наявність доступу до бази даних та коректна робота з нею;

– зручність роботи з інтерфейсом командного рядка;

– відповідність дизайну вимогам Технічного завдання.

3 МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні модульного тестування та системного тестування.

Використовуються наступні методи:

– функціональне тестування, зокрема на рівні Critical path test (ба-зове тестування);

– тестування продуктивності програмного забезпечення, зокрема Stability testing (тестування стабільності);

– тестування інтерфейсу.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується засобами інструментарію pytest.

Працездатність інтерфейсу командного рядка перевіряється шляхом:

– динамічного ручного тестування – введенням доступних та недопустимих значень в вхідні параметри;

– динамічного ручного тестування на відповідність функціональним вимогам;

– статичного тестування коду;

– тестування коректності обчислень за допомогою підготовних наборів даних;

– тестування стабільності роботи при різних умовах;

– тестування зручності використання;

– тестування інтерфейсу.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації та управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

Програмне забезпечення з аналізу результатів інтеграційних тестів

Керівництво користувача

КПІ.ІП-6323.045490.05.34

“ПОГОДЖЕНО”

Керівник проєкту:

_____ О.А. Халус

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____В.О. Пилипенко

Київ – 2020 року

ЗМІСТ

1 ІНСТРУКЦІЯ КОРИСТУВАЧА

1.1 Запуск програмного забезпечення

Для роботи з програмним забезпеченням користувачу не потрібно бути зареєстрованим і проходити етап авторизації. Для запуска програмного забезпечення необхідно, за допомогою команди «python», в командному рядку, запустити файл виконання «run.py» зазор з вхідними параметрами:

python [шлях]/run.py [вхідні_параметри], де

– [шлях] – шлях до папки, в якій знаходиться розроблений програмне забезпечення;

– [вхідні_параметри] – сукупність вхідних параметрів.

1.2 Вхідні параметри

Запуск файл виконання з вхідними параметрами виглядає наступних чином:

run.py [версія] [шлях_до_файлів] {-version4analyze [версія_відносно]} {-path2save[місце_збереження]} {-need_save[збереження]}, де:

– [шлях] – шлях до папки, в якій знаходиться розроблений програмний продукт;

– [версія] – обов’язковий вхідний параметр, ідентифікатор тесту, який аналізується;

– [шлях_до_файлів] – обов’язковий вхідний параметр, шляху до файлів реєстрації тестування;

– {-version4analyze [версія_відносно]} – необов’язковий вхідний параметр, де [версія_відносно] – ідентифікатора тесту, відносно якого відбувається аналіз;

– {-path2save [місце_збереження]} – необов’язковий вхідний параметр, де [місце_збереження] – шлях куди збережеться звіт після завершення аналізу. За замовчуванням – [шлях];

– {-need_save [збереження]} – необов’язковий вхідний параметр, де [збереження] – прапорця необхідності збереження результатів аналізу в базу даних. За замовчуванням – False;

1.3 Перегляд детального звіту

Після того як програмне забезпечення завершить аналіз результатів інтеграційного тестування, буде згенеровано детальний звіт. Приклад такого звіту на рисунку 1.1

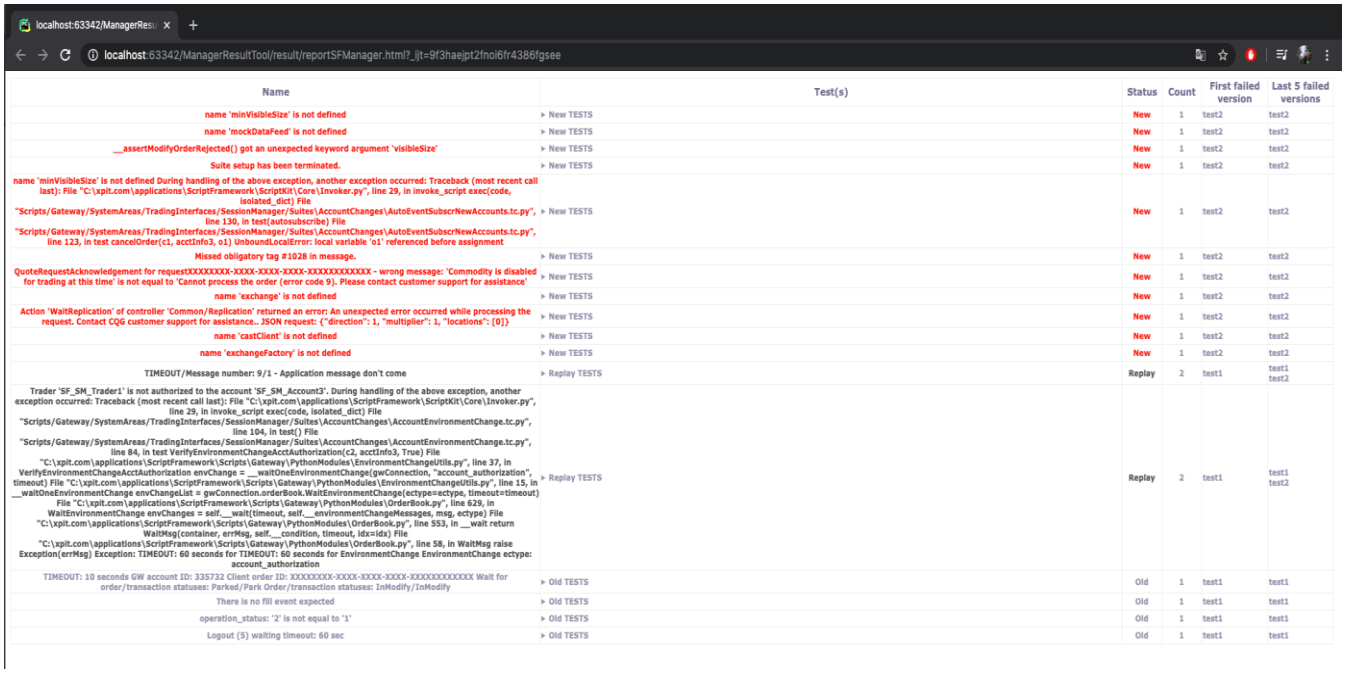


Рисунок 1.1 – Детальний звіт

Колонка «Name» містить інформацію, яка ідентифікує кожну помилка за повідомленням помилки.

Колонка «Test(s)» містить інформацію, яка вказує на вибірку тести в яких сталася помилка.

Колонка «Status» містить статус помилки.

Колонка «Count» містить цифру, яка означає кількість разів коли ця помилка виникала.

Колонка «First failed version» містить версію тесту коли вперше з’явилась ця помилка.

Колонка «Last 5 failed versions» містить вибірку з п’яти версій, де востаннє виникала ця помилка.

Червоним кольором виділяються нові помилки. Також про те, що це нова помилка свідчить надпис «New» в стовбці «Status». Приклад зображення нових помилок на рисунку 1.2

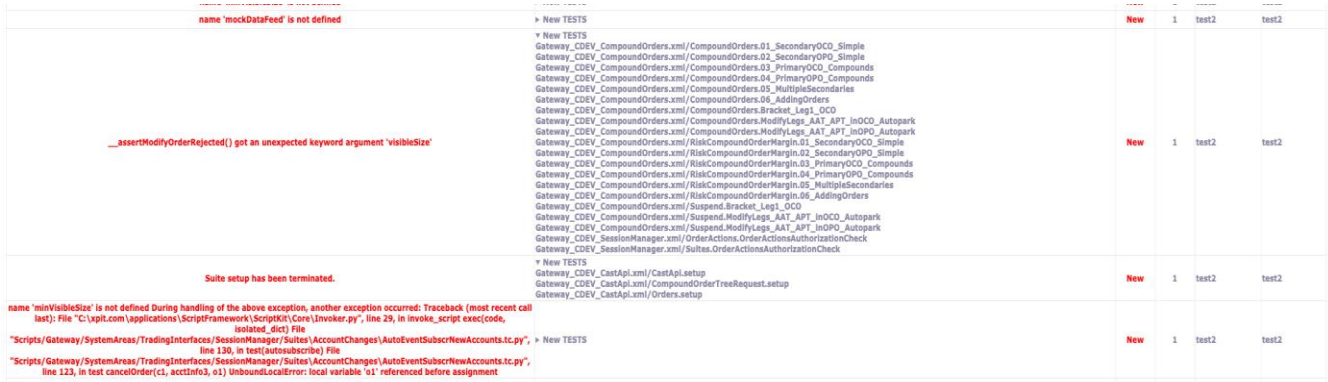


Рисунок 1.2 – Нові помилки у звіті

Чорним кольором виділяється помилка, яка відтворилась під час тестування, але ця помилка вже були зареєстрована в базі даних. Також про цей статус помилки свідчить надпис «Replay» в стовбці «Status». Приклад зображення помилок, які повторились на рисунку 1.3.

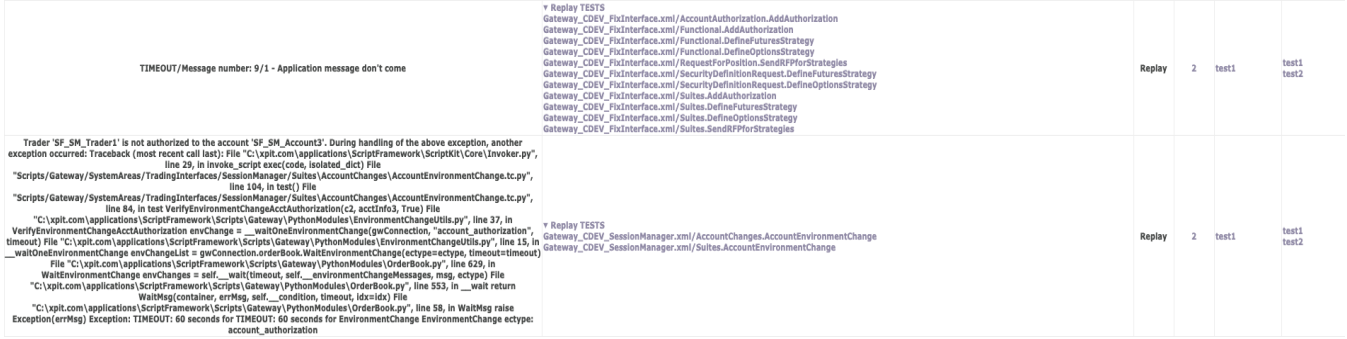


Рисунок 1.3 –Помилки, які повторились

Сірим кольором виділяється помилка, яка не відтворились під час тестування. Також про цей статус помилки свідчить надпис «Old» в стовбці «Status». Приклад зображення помилок, які не повторились на рисунку 1.4.

Рисунок 1.4 –Помилки, які не повторились

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації та управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

Програмне забезпечення з аналізу результатів інтеграційних тестів

Опис програми

КПІ.ІП-6323.045490.06.13

“ПОГОДЖЕНО”

Керівник проєкту:

_____ О.А. Халус

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____В.О. Пилипенко

Київ – 2020 року

Тексти програмного коду

Програмне забезпечення з аналізу результатів інтеграційних тестів

(Найменування програми (документа))

DVD-R

(Вид носія даних)

15 арк, 41 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ – 2020

app.argumentManager.py

```
import argparse

class ArgumentManager:

    def __init__(self):
        ap = argparse.ArgumentParser()
        ap.add_argument('version', type=str, help='Name of run version')
        ap.add_argument('path_logs', type=str, help='Path to logs')
        ap.add_argument('-version4analyze', type=str, default='last', help='.(default last)')
        ap.add_argument('-path2save', type=str, default='/', help='Path to save result.(default
/)')
        ap.add_argument('-need_save', type=bool, default=True,
                        help="Set True if this analyze need save.(default False)")

        args = vars(ap.parse_args())

        self.version = args['version']
        self.path_logs = args['path_logs']
        self.version4analyze = args['version4analyze']
        self.path2save = args['path2save']
        self.need_save = args['need_save']

    def get_version(self):
        return self.version

    def get_path_logs(self):
        return self.path_logs

    def get_version4analyze(self):
        return self.version4analyze

    def get_path2save(self):
        return self.path2save

    def get_need_save(self):
        return self.need_save
```

app.DBManager.py

```
import pymongo


class DBManager:

    def __init__(self):
        client = pymongo.MongoClient(
            "mongodb+srv://admin:admin@cluster0-
d70d1.mongodb.net/test?retryWrites=true&w=majority")
        self.db = client['diploma']
        self.versions_col = self.db['versions-col7']

    def connection(self, hast_name, port):
        pass

    def get_old_analyze(self, version=None):
        return self.versions_col.find_one(None if version is None else {'version': version})

    def save_analyze(self, version, failures):

        find_it = self.versions_col.find_one({'version': version})
        if find_it is None:
            failures_arr = []
            for error_msg, failure in failures:
                failures_arr.append({'error': str(error_msg),
                                     'failure': failure.do_serialize()})
            self.versions_col.insert_one({'version': version, 'failures': failures_arr})
        else:
            print('ERROR: version already exist.')
```

app.parserLogs.py

```
import xml.etree.ElementTree as ET
from app.models.test import Test
from app.models.failure import Failure
from app.models.error import Error
import os

class ParserLogManager:
```

```
        @staticmethod
        def get_fail_message(message):
            error_markers = ["TestCheck.Failure: ", "Exception: ", "NameError: ", "RuntimeError: ",
"IndexError: ",
                             "AssertionError: ", "TypeError: ", "AttributeError: ", "ValueError: "]
            for error_marker in error_markers:
                if error_marker in message:
                    return message.split(error_marker, 1)[1]
            return message

        @staticmethod
        def add_failure_preliminary(file, test_suite_name, test_case_name, failure,
failed_version, gERRORS):
            error = Error(failure)
            if gERRORS.get(error) is None:
                for er in gERRORS.keys():
                    if er == error:
                        gERRORS[er].add_test(Test(file, test_suite_name, test_case_name))
                        gERRORS[er].count_up(failed_version)
                        return
                gERRORS[error] = Failure({Test(file, test_suite_name, test_case_name)},
failed_version, 1, [failed_version])
            else:
                gERRORS[error].add_test(Test(file, test_suite_name, test_case_name))
                gERRORS[error].count_up(failed_version)

        @staticmethod
        def _check_for_suite_failure(file, test_suite, test_suite_name, failed_version, gERRORS):
            if test_suite.get('testsuite_failed') == 'True':
                test_case_name = 'setup'
                for failure in test_suite.findall('.//failure'):
                    ParserLogManager.add_failure_preliminary(file, test_suite_name,
test_case_name,
                                                             ParserLogManager.get_fail_message(failure.text),
                                                             failed_version, gERRORS)

        @staticmethod
        def check_logs(path, version, gERRORS):
            for file in os.listdir(path):
                if file.endswith(".xml"):
```

```
                root = ET.parse(path + '/' + file).getroot()

                for test_suite in root.findall('.//testsuite'):
                    test_suite_name = str(test_suite.get('name'))
                    if test_suite.get('skipped') != '0':
                        ParserLogManager._check_for_suite_failure(file, test_suite, test_suite_name,
version, gERRORS)
                    else:
                        for test_case in test_suite.findall('.//testcase'):
                            for failure in test_case.findall('.//failure'):
                                test_case_name = str(test_case.get('name'))
                                ParserLogManager._add_failure_preliminary(file, test_suite_name,
test_case_name,
                                                                          ParserLogManager.get_fail_message(failure.text),
                                                                          version, gERRORS)

            print('Length of known error database = {}'.format(len(gERRORS)))
            print('Update error database = {}'.format(len(gERRORS)))
```

app.reportGenerator.py

```
class ReportGenerator:

    @staticmethod
    def add_td(text, type=""):
        return f'<td {type}> {text} </td>'

    @staticmethod
    def create_table(error_blocks, status, style):
        error_table = ''

        for error_block in error_blocks:
            error_table += f'<tr>'
            # all failures in single block have the same errorID and failed version list
            error_table += ReportGenerator.add_td(error_block[0], style)

            tests_str = ''
            if not error_block[1].tests or len(error_block[1].tests) != 0:
                tests_str = '<details><summary>ALL TESTS</summary>'
```

```
                for test in sorted(error_block[1].tests):
                    tests_str += f'{test}<br>'
                tests_str += '</details>'

            error_table += ReportGenerator.add_td(tests_str)
            error_table += ReportGenerator.add_td(status, style)
            error_table += ReportGenerator.add_td(error_block[1].count, 'class="data"
align="left"')

            error_table += ReportGenerator.add_td(error_block[1].first_version)
            failed_versions_str = ''
            for failedVersion in error_block[1].last_versions:
                failed_versions_str += f'{failedVersion}<br>'
            error_table += ReportGenerator.add_td(failed_versions_str)
            error_table += f'</tr>'

        return error_table

    @staticmethod
    def create_new_table(new_error_list):
        return ReportGenerator.create_table(new_error_list, 'New', 'class="failed data"')

    @staticmethod
    def create_replay_table(replay_error_list):
        return ReportGenerator.create_table(replay_error_list, 'Replay', 'class="replay data"')

    @staticmethod
    def create_old_table(old_error_list):
        return ReportGenerator.create_table(old_error_list, 'Old', 'class="data"')

    @staticmethod
    def add_style():
        return f'''<style>
            table
            {'{'}
                border-collapse: collapse;
                border: 1px solid black;
                width:1900px;
            {'}'}
```

```
tr
{'{'}
   border:0;
   color:#DD0000;
{'}'}

th
{'{'}
   height:20px;
   color:#727092;
   font-size:12px;
   font-weight:bold;
   font-family:Verdana;
   border: 1px solid #F0F0F0;
{'}'}

td
{'{'}
   margin:0;
   height:20px;
   color:#9A95B0;
   font-size:10px;
   font-weight:bold;
   font-family:Verdana;
   border: 1px solid #F0F0F0;
{'}'}

noborder
{'{'}
   width: 100%;
   border: 0px;
{'}'}

p.header
{'{'}
   color:#727092;
   font-size:14px;
   font-weight:bold;
   font-family:Verdana;
{'}'}
```

```
td.name
{'{'}
   padding-left:0px;
   text-align:left;
{'}'}

td.data
{'{'}
   padding-left:0px;
   text-align:center;
{'}'}

td.failed
{'{'}
   color:#FF0000;
{'}'}

td.skipped
{'{'}
   color:#E18B6B;
{'}'}
td.passed
{'{'}
   color:#347235;
{'}'}

td.replay
{'{'}
   color:#505250;
{'}'}

td.inInspection
{'{'}
   color:#4A0AD0;
{'}'}

.description
{'{'}
   overflow:hidden;
```

```
                white-space:nowrap;
                width: 50%;
            {'}'}

            th.failed
            {'{'}
                height:12px;
                color:#FF0000;
                background:#FF0000;
            {'}'}
            th.skipped
            {'{'}
                height:12px;
                color:#E18B6B;
                background:#E18B6B;
            {'}'}
            th.passed
            {'{'}
                height:12px;
                color:#347235;
                background:#347235;
            {'}'}
        </style>'''

    @staticmethod
    def write_fails_2_HTML(gERRORS, data_save_path, version):
        # Group tests by failure + failed version list, otherwise create new entry
        # if defect failedVersion has id = len(totalFailedVersions) and len(defectFailedVersion)
== 1, this is new defect
        # if defect failedVersion has id = len(totalFailedVersions) and len(defectFailedVersion)
> 1, this is replay defect
        # otherwise this is old defect

        new_errors = []
        replay_errors = []
        old_errors = []

        for error, failure in gERRORS.items():
            if failure.first_version == version and failure.count == 1:
                new_errors.append([error, failure])
```

```
            else:
                if version in failure.last_versions:
                    replay_errors.append([error, failure])
                else:
                    old_errors.append([error, failure])

        ReportGenerator._writeFailsToHTML(new_errors, replay_errors, old_errors,
data_save_path)

    @staticmethod
    def _writeFailsToHTML(new_errors, replay_errors, old_errors, data_save_path):
        f = open(data_save_path + '/reportSFManager.html', 'w')

        message = f'''<html>
        <head>
            {ReportGenerator.add_style()}
        </head>
        <body>
            <table class = "defect">
                <tr align ="center" width="99%">
                    <th width="40%"> Name </th>
                    <th width="44%"> Test </th>
                    <th width="3%"> Status </th>
                    <th width="3%"> Count </th>
                    <th width="5%"> First failed version </th>
                    <th width="5%"> Last 5 failed versions </th>
                </tr>
                {ReportGenerator.create_new_table(new_errors)}
                {ReportGenerator.create_replay_table(replay_errors)}
                {ReportGenerator.create_old_table(old_errors)}
            </table>
        </body>
        </html>'''

        f.write(message)
        f.close()
```

app.models.error.py

```
import Levenshtein
import re

threshold = 0.9
ignored_chars = ' \t\n'


class Error:
    def __init__(self, msg):
        self.msg = Error.msg_no_HTTPError(Error.msg_no_GUID(msg))
        self.msg_transformed = Error.apply_transformations(msg)

    def __str__(self):
        return self.msg

    def __repr__(self):
        return self.msg

    def __eq__(self, other):
        # we do not need to calculate ratio for significantly different messages
        if Error.messages_definitely_different(self.msg_transformed, other.msg_transformed):
            return False
        return Levenshtein.ratio(self.msg_transformed, other.msg_transformed) > threshold

    def __hash__(self):
        return hash(self.msg_transformed)

    @staticmethod
    def apply_transformations(message):
        for transformation in [Error.msg_no_HTTPError, Error.msg_no_GUID,
Error.msg_no_GUID,
                               Error.msg_no_TIMEOUT_message_number, Error.msg_no_ignored_chars,
# , Error.msgNoNumber
                               Error.msg_no_prefix]:
            message = transformation(message)
        return message

    @staticmethod
    def messages_definitely_different(message1, message2):
```

```
            # we consider messages different, if their relative size difference is more than (1 -
threshold)
            possible_difference_percent = (1.0 - threshold) / 2
            return not (0.5 + possible_difference_percent >=
                    len(message1) / (len(message1) + len(message2)) >=
                    0.5 - possible_difference_percent)

        @staticmethod
        def msg_no_GUID(message):
            guids = re.findall(r'(\w{8}-\w{4}-\w{4}-\w{4}-\w{12})', message)
            for i in guids:
                message = message.replace(i, 'XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX')
            return message

        @staticmethod
        def msg_no_TIMEOUT_message_number(message):
            if message.find('TIMEOUT/Message number:') > -1:
                second_part = message \
                    .replace(r'*.TIMEOUT/Message number: ', '') \
                    .replace(r" - Application message don't come*.", '')
                return message.replace(second_part, 'X/XX')
            else:
                return message

        @staticmethod
        def msg_no_ignored_chars(message):
            return message.translate({ord(c): None for c in ignored_chars})

        @staticmethod
        def msg_no_queries(message):
            # we do not want to compare query outputs with each other automatically, it is
extremely slow
            if message.find('xml version') != -1:
                message = 'Query error'
            return message

        # try without nemoving number from message
        # @staticmethod
        # def msgNoNumber(message):
        #   return ''.join([i for i in message if not i.isdigit()])
```

```
        @staticmethod
        def msg_no_HTTPError(message):
            if message.find('urllib.error.HTTPError: HTTP Error 500: Internal Server Error') != -1:
                message = 'HTTP Error 500: Internal Server Error'
            if message.find('urllib.error.HTTPError: HTTP Error 401: Unauthorized') != -1:
                message = 'HTTP Error 401: Unauthorized'
            return message

        @staticmethod
        def msg_no_prefix(message):
            return message.replace('CQG_', '')
```

app.models.failure.py

```
from app.models.test import Test


class Failure:
    def __init__(self, tests=None, first_version=None, count=None, last_versions=None,
data=None):
        if data is not None:
            self.tests = set()
            for test_it in data['tests']:
                self.tests.add(Test(test_it['file'], test_it['test_suite'], test_it['test_case']))
            self.count = data['count']
            self.first_version = data['first_version']
            self.last_versions = data['last_versions']
        else:
            self.tests = tests
            self.count = count
            self.first_version = first_version
            self.last_versions = last_versions

    def __eq__(self, other):
        return self.tests == other.tests \
            and self.count == other.count \
            and self.first_version == other.first_version \
            and self.last_versions == other.last_versions
```

```
    def __str__(self):
        return '{} {} ({}, {})'.format(self.tests, self.count, self.first_version, self.last_versions)

    def __repr__(self):
        return self.__str__()

    def count_up(self, new_version):
        if new_version not in self.last_versions:
            self.count += 1
            self.last_versions.append(new_version)

    def add_test(self, new_test):
        self.tests.add(new_test)

    def do_serialize(self):
        tests_arr = []
        for test_it in self.tests:
            tests_arr.append(test_it.do_serialize())
        return {
            'tests': tests_arr,
            'count': self.count,
            'first_version': self.first_version,
            'last_versions': self.last_versions,
        }
```

app.models.test.py

```
class Test:
    def __init__(self, file, test_suite, test_case):
        self.file = file
        self.test_suite = test_suite
        self.test_case = test_case

    def __eq__(self, other):
        return self.file == other.file \
               and self.test_suite == other.test_suite \
               and self.test_case == other.test_case

    def __hash__(self):
        return hash((self.file, self.test_suite, self.test_case))
```

```
    def __str__(self):
        return '{}/{}.{}'.format(self.file, self.test_suite, self.test_case)

    def __repr__(self):
        return self.__str__()

    def __lt__(self, other):
        return self.__str__() < str(other)

    def do_serialize(self):
        return {
            'file': self.file,
            'test_suite': self.test_suite,
            'test_case': self.test_case
        }
```

run.py

```
import os
import xml.etree.ElementTree as ET
from app.models.error import Error
from app.models.test import Test
from app.models.failure import Failure
from app.reportGenerator import ReportGenerator
from app.argumentManager import ArgumentManager
from app.parserLogs import ParserLogManager
from app.DBManager import DBManager

numberOfLastFailedVersions = 5


def _clean_common_variables():
    # globals are created upon first execution
    global gERRORS

    gERRORS = {}  # map: Error <-> list<Failure>


def _get_errors_from_DB(data):
```

```
        errors = {}

        if data is not None:
            for failure in data['failures']:
                errors[Error(failure['error'])] = Failure(data=failure['failure'])

        return errors


    def get_old_fail(version4analyze):
        global gERRORS

        db_manager = DBManager()
        gERRORS = _get_errors_from_DB(db_manager.get_old_analyze(version=
                                            version4analyze if version4analyze is not None else
None))


    def save_errors(version=None):
        global gERRORS

        db_manager = DBManager()
        db_manager.save_analyze(version, gERRORS.items())


    def pars_and_check(path, version):
        global gERRORS

        ParserLogManager.check_logs(path, version, gERRORS)


    def do_check(path, version, data_save_path, version4analyze, generate_report=False):
        _clean_common_variables()

        if not os.path.isdir(path):
            print(f"Folder {path} does not exists!")
            return
        get_old_fail(None if (version4analyze is None or version4analyze == 'last') else
version4analyze)
        pars_and_check(path=path, version=version)
```

```
            save_errors(version=version)
            if generate_report:
                ReportGenerator.write_fails_2_HTML(gERRORS=gERRORS,
data_save_path=data_save_path, version=version)
        def run():
            print('Starting SF Manager Result...')
            argsManager = ArgumentManager()

            do_check(argsManager.get_path_logs(), argsManager.get_version(),
argsManager.get_path2save(),
                     argsManager.get_version4analyze(), argsManager.get_need_save())

            print('Done!')


        if __name__ == '__main__':
            run()
```

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації та управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” ___ 2020 р.

Програмне забезпечення з аналізу результатів інтеграційних тестів

Графічний матеріал

КПІ.ІП-6323.045490.07.99

“ПОГОДЖЕНО”

Керівник проєкту:

_____ О.А. Халус

Нормоконтроль:

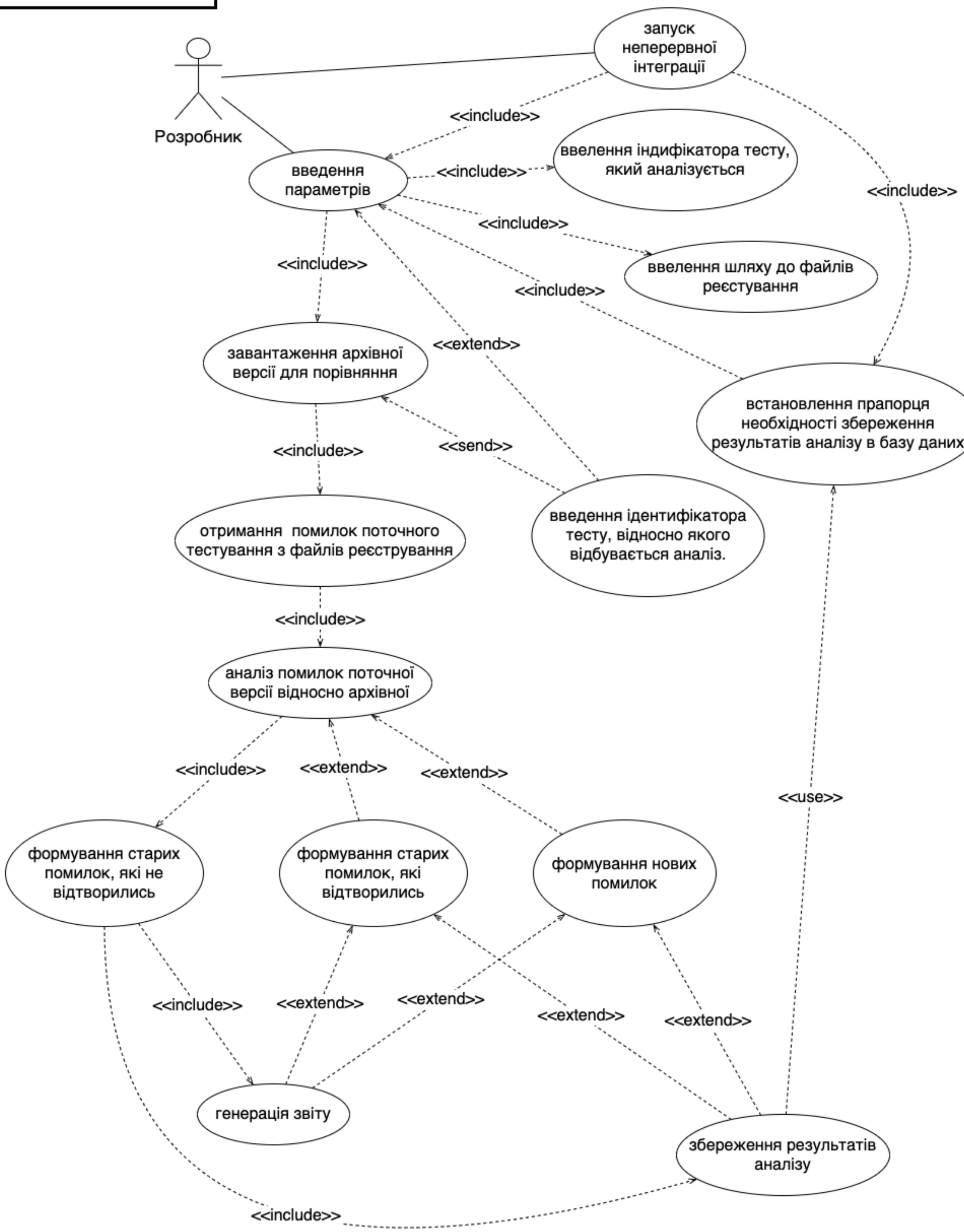
_____ К.І. Ліщук

Виконавець:

_____В.О. Пилипенко

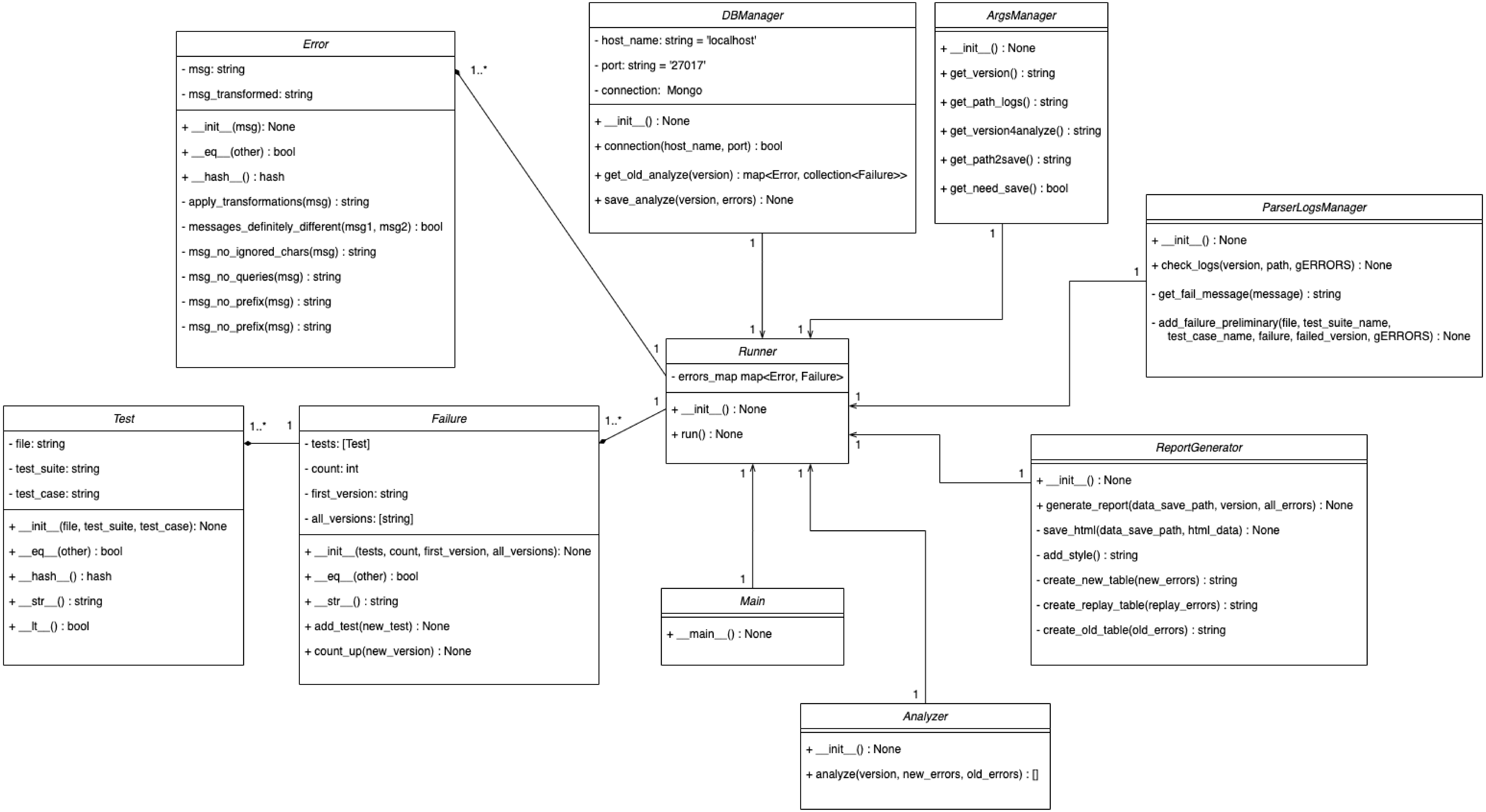
Київ – 2020 року

КПІ.ІП-6323.045490.07.99.СС



					КПІ.ІП-6323.045490.07.99.СС			
					Схема структурна варіантів використань	Лит.	Арк.	Аркушів
Зм.	Арк.	№ докум.	Підп.	Дата				1
Розроб.		Пилипенко В.О.				Аркуш		Аркушів
Перев.		Халус О.А.			Програмне забезпечення з аналізу результатів інтеграційних тестів	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63		
Т. Кон.								
Н. Кон.		Ліщук К.І.						
Затв.		Халус О.А.						

КПІ.ІП-6323.045490.07.99.СС



					КПІ.ІП-6323.045490.07.99.СС			
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна класів програмного забезпечення	Літера	Маса	Масштаб
Розробив		Пилипенко В.О.						
Перевірив		Халус О.А.				Аркуш		Аркушів
Т. кон.								
Н. кон.		Ліщук К.І.			Програмне забезпечення з аналізу результатів інтеграційних тестів	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63		
Затвердив		Халус О.А.						

localhost:63342/ManagerResu

localhost:63342/ManagerResultTool/result/reportSFManager.html?_ijt=9f3haejpt2fnoi6fr4386fgsee

Name	Test(s)	Status	Count	First failed version	Last 5 failed versions
name 'minVisibleSize' is not defined	▸ New TESTS	New	1	test2	test2
name 'mockDataFeed' is not defined	▸ New TESTS	New	1	test2	test2
__assertModifyOrderRejected() got an unexpected keyword argument 'visibleSize'	▸ New TESTS	New	1	test2	test2
Suite setup has been terminated.	▾ New TESTS Gateway_CDEV_CastApi.xml/CastApi.setup Gateway_CDEV_CastApi.xml/CompoundOrderTreeRequest.setup Gateway_CDEV_CastApi.xml/Orders.setup	New	1	test2	test2
name 'minVisibleSize' is not defined During handling of the above exception, another exception occurred: Traceback (most recent call last): File "C:\xpit.com\applications\ScriptFramework\ScriptKit\Core\Invoker.py", line 29, in invoke_script exec(code, isolated_dict) File "Scripts/Gateway/SystemAreas/TradingInterfaces/SessionManager/Suites\AccountChanges\AutoEventSubscrNewAccounts.tc.py", line 130, in test(autosubscribe) File "Scripts/Gateway/SystemAreas/TradingInterfaces/SessionManager/Suites\AccountChanges\AutoEventSubscrNewAccounts.tc.py", line 123, in test cancelOrder(c1, acctInfo3, o1) UnboundLocalError: local variable 'o1' referenced before assignment	▸ New TESTS	New	1	test2	test2
Missed obligatory tag #1028 in message.	▸ New TESTS	New	1	test2	test2
QuoteRequestAcknowledgement for requestXXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX - wrong message: 'Commodity is disabled for trading at this time' is not equal to 'Cannot process the order (error code 9). Please contact customer support for assistance'	▸ New TESTS	New	1	test2	test2
name 'exchange' is not defined	▸ New TESTS	New	1	test2	test2
Action 'WaitReplication' of controller 'Common/Replication' returned an error: An unexpected error occurred while processing the request. Contact CQG customer support for assistance.. JSON request: {"direction": 1, "multiplier": 1, "locations": [0]}	▸ New TESTS	New	1	test2	test2
name 'castClient' is not defined	▸ New TESTS	New	1	test2	test2
name 'exchangeFactory' is not defined	▸ New TESTS	New	1	test2	test2
TIMEOUT/Message number: 9/1 - Application message don't come	▾ Replay TESTS Gateway_CDEV_FixInterface.xml/AccountAuthorization.AddAuthorization Gateway_CDEV_FixInterface.xml/Functional.AddAuthorization Gateway_CDEV_FixInterface.xml/Functional.DefineFuturesStrategy Gateway_CDEV_FixInterface.xml/Functional.DefineOptionsStrategy Gateway_CDEV_FixInterface.xml/RequestForPosition.SendRFPforStrategies Gateway_CDEV_FixInterface.xml/SecurityDefinitionRequest.DefineFuturesStrategy Gateway_CDEV_FixInterface.xml/SecurityDefinitionRequest.DefineOptionsStrategy Gateway_CDEV_FixInterface.xml/Suites.AddAuthorization Gateway_CDEV_FixInterface.xml/Suites.DefineFuturesStrategy Gateway_CDEV_FixInterface.xml/Suites.DefineOptionsStrategy Gateway_CDEV_FixInterface.xml/Suites.SendRFPforStrategies	Replay	2	test1	test1 test2
Trader 'SF_SM_Trader1' is not authorized to the account 'SF_SM_Account3'. During handling of the above exception, another exception occurred: Traceback (most recent call last): File "C:\xpit.com\applications\ScriptFramework\ScriptKit\Core\Invoker.py", line 29, in invoke_script exec(code, isolated_dict) File "Scripts/Gateway/SystemAreas/TradingInterfaces/SessionManager/Suites\AccountChanges\AccountEnvironmentChange.tc.py", line 104, in test() File "Scripts/Gateway/SystemAreas/TradingInterfaces/SessionManager/Suites\AccountChanges\AccountEnvironmentChange.tc.py", line 84, in test VerifyEnvironmentChangeAcctAuthorization(c2, acctInfo3, True) File "C:\xpit.com\applications\ScriptFramework\Scripts\Gateway\PythonModules\EnvironmentChangeUtils.py", line 37, in VerifyEnvironmentChangeAcctAuthorization envChange = __waitOneEnvironmentChange(gwConnection, "account_authorization", timeout) File "C:\xpit.com\applications\ScriptFramework\Scripts\Gateway\PythonModules\EnvironmentChangeUtils.py", line 15, in __waitOneEnvironmentChange envChangeList = gwConnection.orderBook.WaitEnvironmentChange(ectype=ectype, timeout=timeout) File "C:\xpit.com\applications\ScriptFramework\Scripts\Gateway\PythonModules\OrderBook.py", line 629, in WaitEnvironmentChange envChanges = self.__wait(timeout, self.__environmentChangeMessages, msg, ectype) File "C:\xpit.com\applications\ScriptFramework\Scripts\Gateway\PythonModules\OrderBook.py", line 553, in __wait return WaitMsg(container, errMsg, self.__condition, timeout, idx=idx) File "C:\xpit.com\applications\ScriptFramework\Scripts\Gateway\PythonModules\OrderBook.py", line 58, in WaitMsg raise Exception(errMsg) Exception: TIMEOUT: 60 seconds for TIMEOUT: 60 seconds for EnvironmentChange EnvironmentChange ectype: account_authorization	▾ Replay TESTS Gateway_CDEV_SessionManager.xml/AccountChanges.AccountEnvironmentChange Gateway_CDEV_SessionManager.xml/Suites.AccountEnvironmentChange	Replay	2	test1	test1 test2
TIMEOUT: 10 seconds GW account ID: 335732 Client order ID: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX Wait for order/transaction statuses: Parked/Park Order/transaction statuses: InModify/InModify	▾ Old TESTS Gateway_CDEV_ForceCareOrders.xml/ForceCareOptionChange.ForceCare_OptionChange Gateway_CDEV_ForceCareOrders.xml/ForceCareOrders.ForceCare_OptionChange	Old	1	test1	test1
There is no fill event expected	▸ Old TESTS	Old	1	test1	test1
operation_status: '2' is not equal to '1'	▸ Old TESTS	Old	1	test1	test1
Logout (5) waiting timeout: 60 sec	▾ Old TESTS Gateway_CDEV_FixInterfaceNew.xml/SessionPersistence.TestSessionPersistenceON Gateway_CDEV_FixInterfaceNew.xml/Suites.TestSessionPersistenceON	Old	1	test1	test1

```
Terminal:  Local  +
(venv) Volodias-MacBook-Pro:ManagerResultTool vova$ python run.py test2 test/114 -version4analyze test1 -path2save result -need_save True
6: TODO  8: Services  9: Version Control  Terminal  Python Console
```

					КПІ.ІП-6323.045490.07.99.КЕ	
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм	Літера / Маса / Масштаб
Розробив		Пилипенко В.О.				
Перевірив		Халус О.А.				Аркуш / Аркушів
Т. кон.						
Н. кон.		Ліщук К.І.			Програмне забезпечення з аналізу результатів інтеграційних тестів	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63
Затвердив		Халус О.А.				