

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ
ВАДИМА ГЕТЬМАНА

Навчально-науковий інститут «Інститут інформаційних технологій в економіці»

Кафедра системного аналізу та кібербезпеки

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА

«Системний аналіз»

СПЕЦІАЛЬНІСТЬ

124 «Системний аналіз»

Форма навчання: денна

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

На тему: «Системний аналіз автоматизованого тестування програмного забезпечення»

Здобувача: Деменка Івана Олександровича

(підпис)

Науковий керівник: д.е.н., доцент кафедри Камінський О.Є.

(підпис)

Робота допущена до захисту перед екзаменаційною комісією з атестації здобувачів вищої освіти (ЕК)

Завідувач кафедри: д.ф.-м.н., проф. Джалладова І.А.

(підпис)

Київ 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА
Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»
Кафедра системного аналізу та кібербезпеки
ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА **«Системний аналіз»**
СПЕЦІАЛЬНІСТЬ **124 «Системний аналіз»**

ПОГОДЖЕНО Керівник проєктної групи (гарант) освітньо-професійної програми Джалладова І.А. <i>(підпис)</i> 2024 р.	ЗАТВЕРДЖУЮ Завідувач кафедри Джалладова І.А. <i>(підпис)</i> 2024 р.
---	---

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
здобувачу вищої освіти Деменку Івану Олександровичу
денної форми навчання
на підготовку кваліфікаційної магістерської роботи
На тему: «Системний аналіз автоматизованого тестування програмного забезпечення»
Тему затверджено наказом ректора Університету від «__»_____2024р.
№__
Кваліфікаційна магістерська робота виконується на матеріалах

План кваліфікаційної магістерської роботи	
Розділ 1	Теоретичні засади використання автоматизованого тестування програмних додатків.
Розділ 2	Аналіз моделей та методів тестування програмних додатків.
Розділ 3	Проєктні рішення щодо впровадження системи автоматизованого тестування на підприємстві.
Об'єкт дослідження:	Процес автоматизованого тестування програмного забезпечення.

Предмет дослідження:	Методи системного аналізу, що використовуються в процесі автоматизованого тестування програмного забезпечення.
Мета кваліфікаційної магістерської роботи:	Здійснити системний аналіз автоматизованого тестування програмного забезпечення.

Конкретні завдання, які здобувач повинен виконати для досягнення поставленої мети:

У розділі 1

1. Здійснити аналіз акторів та функцій процесу автоматизованого тестування.
2. Проаналізувати та порівняти основні види і напрямки автоматизованого тестування.
3. Ідентифікувати основні переваги та недоліки автоматизованого тестування.

У розділі 2

1. Здійснити системний аналіз ІТ-проекту «Dropbox». Виділити головні аспекти та зони для майбутнього тестування.
2. Здійснити аналіз та розробку тестової документації.
3. Продемонструвати та проаналізувати основні етапи впровадження автоматизованого тестування.

У розділі 3

1. Розробити тестовий фреймворк та написати автоматизовані тести для системи.
2. Інтегрувати автоматизоване тестування в CI/CD процес.
3. Здійснити аналіз отриманих результатів та оптимізувати процес тестування.

**Завдання підготував
науковий керівник**

(підпис)

Камінський О.Є.
(ініціали, прізвище)

«__»_____2024 р.

**Завдання одержав
здобувач**

(підпис)

Деменко І.О.
(ініціали, прізвище)

«__»_____2024 р.

Реферат

Кваліфікаційна магістерська робота містить 81 сторінку, 31 рисунок, список використаних джерел з 54 найменувань.

«Системний аналіз автоматизованого тестування програмного забезпечення»

Об'єктом дослідження є процес автоматизованого тестування програмного забезпечення.

Предметом дослідження є теоретико-методичні засади, методи, підходи та інструменти, що використовуються в процесі використання та впровадження автоматизованого тестування програмного забезпечення.

Метою кваліфікаційної магістерської роботи – здійснення системного аналізу автоматизованого тестування програмного забезпечення та розроблення науково-практичних рекомендацій щодо використання і впровадження даного типу тестування для ІТ-проектів.

Відповідно до поставленої мети були визначені такі *завдання*:

- визначити сутність і принципи автоматизованого тестування;
- дослідити та порівняти основні види автоматизованого тестування;
- визначити ключові переваги та недоліки автоматизованого тестування;
- здійснити системний аналіз ІТ-проекту «Dropbox» для виділення головних аспектів та зон для покриття автоматизованим тестування;
- здійснити аналіз впливу тестової документації на процес впровадження автоматизованого тестування;
- дослідити та визначити головні етапи впровадження автоматизованого тестування;
- розробити тестовий фреймворк для створення автоматизованих скриптів для перевірки якості та ефективності програмного забезпечення;
- інтегрувати процес автоматизованого тестування в цикли розробки ІТ-проекту;
- здійснити аналіз впливу автоматизованого тестування на розробку програмного забезпечення;

– провести огляд літератури та досліджень щодо сучасних методів тестування програмного забезпечення.

Практичне значення отриманих результатів. Можливість використання результатів роботи як інструкції для ІТ-проєктів, які вже використовують або мають намір використовувати автоматизоване тестування для програмного забезпечення.

Рік виконання кваліфікаційної магістерської роботи 2024.

Рік захисту роботи 2024.

Ключові слова: програмне забезпечення, розробка програмного забезпечення, тестування, автоматизоване тестування, тести, тестові сценарії, тестова документація, стратегія тестування, аналіз даних.

Відгук
про кваліфікаційну магістерську роботу
здобувача освітньо-професійної програми «Системний аналіз»
навчально-наукового інституту Інформаційних технологій в економіці
Деменка Івана Олександровича
на тему «Системний аналіз автоматизованого тестування програмного
забезпечення»

Актуальність теми: дипломна робота є самостійним дослідженням, яке присвячено важливій темі — здійснення системного аналізу автоматизованого тестування програмного забезпечення та розроблення науково-практичних рекомендацій щодо використання і впровадження даного типу тестування для ІТ-проектів. Широке використання інформаційних технологій у нашому житті підвищує увагу до якості програмного забезпечення, що ставить питання впровадження автоматизованого тестування на передній план, то обумовлює актуальність даного дослідження.

Позитивні риси кваліфікаційної магістерської роботи: у роботі було дано чітке визначення термінології та складових автоматизованого тестування, розкрито його види, напрямки, переваги та недоліки, досліджено ефективність виявлення та виправлення помилок за допомогою автоматизованого тестування; проаналізовано вплив автоматизованого тестування на час розробки ПЗ та на ризики, що виникають під час розробки ПЗ

Наявність самостійних розробок автора: За допомогою методів системного аналізу здійснено аналіз обраного ІТ-проекту для впровадження автоматизованого тестування, включаючи визначення головних особливостей системи, її функцій та критичних зон. Розроблено тестовий фреймворк, написано автоматизовані тести, досліджено інтеграцію автоматизованого тестування в CI/CD процес, проведено детальний аналіз отриманих результатів.

Цінність теоретичних висновків та практичних рекомендацій. Результати дослідження обумовлюють практичну значущість роботи, що

визначається у можливості використання запропонованого автором інструментарію для автоматизованого тестування програмного забезпечення в ІТ-проектах.

Наявність недоліків. Дипломна робота містить невелику кількість недоліків, які не впливають на загальну якість виконаної роботи.

Загальна оцінка кваліфікаційної бакалаврської роботи та її допущення до захисту перед ЕК: тема роботи розкрита, основні положення та висновки повністю обґрунтовані. Робота Деменко Івана на тему «Системний аналіз автоматизованого тестування програмного забезпечення» може бути допущена до захисту перед ЕК і заслуговує на високу оцінку, а її автору може бути присвоєно ступінь магістра зі спеціальності «Системний аналіз».

Науковий керівник

д.е.н., доцент кафедри системного аналізу та кібербезпеки,

_____ Камінський О.Є.

Рецензія

на кваліфікаційну магістерську роботу

здобувача вищої освіти

Деменка Івана Олександровича

Тема: «Системний аналіз автоматизованого тестування програмного забезпечення»

Актуальність теми кваліфікаційної роботи і доцільність її розроблення: Актуальність теми обумовлена стрімким зростанням складності програмних систем та необхідністю забезпечення їх високої якості і надійності в умовах сучасного ринку. Автоматизоване тестування стає критично важливим інструментом, який дозволяє значно зменшити час та ресурси на тестування, підвищуючи при цьому точність і ефективність виявлення помилок. Розроблення та вдосконалення методів автоматизації тестування сприяє покращенню процесів розробки ПЗ, зменшуючи витрати на підтримку та розвиток програмних продуктів. Це має особливе значення для компаній, що прагнуть забезпечити конкурентоспроможність на ринку та високу якість своїх продуктів. Тому дослідження у цій галузі є надзвичайно важливими та доцільними.

Якість проведеного дослідження: якість проведеного дослідження в дипломній роботі є надзвичайно високою. Автор продемонстрував вичерпний аналіз сучасних методів та інструментів автоматизованого тестування, застосовуючи як теоретичні, так і практичні підходи. Робота включає глибокий огляд літератури, що свідчить про ґрунтовне вивчення наявних джерел та актуальних досліджень у даній сфері. Практична частина дослідження базується на реальних прикладах, що підкріплює теоретичні висновки і демонструє вміння автора застосовувати отримані знання на практиці. Використані методи аналізу та інструменти відповідають сучасним стандартам, а результати дослідження представлені чітко і логічно, що робить роботу цілісною та науково обґрунтованою.

Позитивні риси кваліфікаційної роботи: кваліфікаційна робота відзначається рядом позитивних рис, які роблять її бездоганною. По-перше, автор демонструє глибоке розуміння предмету, вміло поєднуючи теоретичні аспекти з

практичними прикладами, що підкріплює його аналіз конкретними кейсами. По-друге, робота вирізняється високою структурованістю та логічністю викладу, де кожен розділ гармонійно доповнює інший, створюючи цілісне уявлення про досліджувану проблему. По-третє, ретельний огляд сучасних інструментів і методів автоматизованого тестування підкреслює актуальність і інноваційність дослідження. Нарешті, оформлення роботи відповідає всім академічним стандартам, а використання чіткої та професійної мови сприяє легкому сприйняттю матеріалу.

Зауваження: незважаючи на загальну високу якість роботи, можна було б додати більше практичних прикладів та порівнянь різних інструментів автоматизованого тестування для більш глибокого аналізу.

Практична значимість висновків і рекомендацій: практична значимість висновків і рекомендацій дипломної роботи є надзвичайно високою. Запропоновані автором методи та інструменти автоматизованого тестування можуть бути безпосередньо впроваджені в процеси розробки програмного забезпечення, що дозволить значно підвищити ефективність тестування, зменшити витрати часу та ресурсів, а також покращити якість кінцевого продукту.

Рецензент: директор ФОП «Кот Максим Геннадійович»

Кот М.Г.

(підпис, ПІБ)

ЗМІСТ

ВСТУП.....	3
РОЗДІЛ 1. ТЕОРЕТИЧНІ ЗАСАДИ ВИКОРИСТАННЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ПРОГРАМНИХ ДОДАТКІВ.....	6
1.1 Актори і функції процесу автоматизованого тестування	6
1.2 Аналіз видів і напрямків автоматизованого тестування.....	9
1.3 Переваги та недоліки автоматизованого тестування.....	15
1.4 Висновок до розділу.....	17
РОЗДІЛ 2. АНАЛІЗ МОДЕЛЕЙ ТА МЕТОДІВ ТЕСТУВАННЯ ПРОГРАМНИХ ДОДАТКІВ	19
2.1 Аналіз проєкту та визначення ключових областей для запровадження автоматизованого тестування.....	19
2.2 Аналіз вимог та створення тестової документації.....	26
2.3 Процес та етапи впровадження автоматизованого тестування на проєкті	30
2.4 Висновок до розділу.....	35
РОЗДІЛ 3. ПРОЄКТНІ РІШЕННЯ ЩОДО ВПРОВАДЖЕННЯ СИСТЕМИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ НА ПІДПРИЄМСТВІ	36
3.1 Розробка тестового фреймворку та написання автоматизованих тестів 36	
3.2 Інтеграція автоматизованого тестування в CI/CD процес.....	51
3.3 Аналіз отриманих результатів та оптимізація процесу тестування....	59
3.4 Висновок до розділу.....	73
ВИСНОВКИ	75
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....	77

ВСТУП

Системний аналіз автоматизованого тестування програмного забезпечення є важливою та актуальною частиною досліджень у сфері програмної інженерії і розробки. За останні роки швидкість випуску нових версій програмного забезпечення значно зросла, що створює величезний попит на ефективні методи тестування для забезпечення якості продуктів. Системний аналіз є важливим інструментом для дослідження та вдосконалення процесів тестування, адаптуючись до швидких змін середовища розробки ПЗ. Особливо в контексті України, яка стає все більш важливим гравцем на міжнародній ІТ-арені, впровадження системного аналізу в автоматизоване тестування може сприяти підвищенню конкурентоспроможності українських ІТ-компаній та забезпечити високу якість розроблюваного ПЗ на міжнародному ринку.

Широке використання інформаційних технологій у нашому житті підвищує увагу до якості програмного забезпечення, що ставить питання впровадження автоматизованого тестування на передній план. На сьогоднішній день існує значна кількість наукових досліджень, які розглядають різні підходи до впровадження автоматизованого тестування під час розробки ІТ-проектів, проводять їх порівняльний аналіз та виділяють особливості кожного з них. Так, наприклад, наукові роботи Дороті Грема, Марка Фьюстера «Досвід автоматизованого тестування» [1] та Гленфорда Дж. Майерса, Тома Беджетті, Корі Сендлера «Мистецтво Програмне Забезпечення Тестування» [2] присвячені детальному дослідженню найбільш популярних та ефективних методів впровадження автоматизованого тестування у процес розробки програмного забезпечення.

У світлі швидкозмінного технологічного середовища та неперервного зростання складності програмного забезпечення, вибір стратегії для впровадження автоматизованого тестування набуває важливості та актуальності. Початкова стратегія становить основу майбутнього процесу автоматизованого тестування, тому вибір її необхідно виконувати з особливою уважністю. Важливо заздалегідь

визначити найбільш критичні аспекти програмного забезпечення та спрямувати на них основні зусилля для забезпечення максимальної якості та надійності. Дана тема дуже детально розглянута в науковій роботі Метью Хойссера та Майкла Ларсена «Стратегії тестування програмного забезпечення» [3].

Вибір методу тестування виникає внаслідок різниці у вимогах, особливостях галузі та наявних ресурсах між різними ІТ-проєктами. Визначення доцільності впровадження автоматизованого тестування стає ключовим етапом у плануванні та виконанні проєкту. З одного боку, автоматизація може значно зменшити час, потрібний для проведення тестування, що дозволяє швидше виявляти та виправляти помилки. З іншого боку, це може вимагати значних витрат на впровадження та підтримку, що може бути нецільовим для деяких проєктів. Ці проблеми детально демонструються у наукових працях Кулікова С. С. «Тестування програмного забезпечення» [4] та Арнона Аксельрода «Повний посібник з автоматизованого тестування» [5].

Отже, можна зробити висновок, що вказана тема є популярною і велика кількість науковців проявляє до неї зацікавленість. Але, на превеликий жаль, у сучасному житті все дуже швидко змінюється та розвиваються, тому не можливо дати відповіді на всі запитання, і це залишає простір для подальшої наукової роботи. Системний аналіз виступає інструментом, який допоможе нам зрозуміти проблематику та особливості даної сфери, і дасть змогу відповісти на всі запитання.

Мета роботи: здійснити системний аналіз автоматизованого тестування програмного забезпечення.

Завдання дослідження:

- Дати чітке визначення терміну автоматизоване тестування, розкрити його види, напрямки, переваги та недоліки;
- Продемонструвати на реальному прикладі коли, де і для чого доцільно та ефективно використовувати автоматизоване тестування;
- Проаналізувати ефективність виявлення та виправлення помилок за допомогою автоматизованого тестування;

- Проаналізувати вплив автоматизованого тестування на час розробки ПЗ;
- Проаналізувати вплив автоматизованого тестування на ризики, що виникають під час розробки ПЗ.

Об'єкт дослідження: процес автоматизованого тестування ПЗ.

Предмет дослідження: методи системного аналізу, що використовуються в процесі автоматизованого тестування програмного забезпечення.

Методи дослідження:

- Спостереження та узагальнення власного досвіду;
- Аналіз книжок, статей і монографій;
- Аналіз і синтез предмету дослідження;
- Порівняння – проведення паралелі між підходами по реалізації автоматизованого тестування;
- Абстрагування;
- Ретроспектива;
- Опитування та анкетування колег.

Теоретична значущість: розгляд фундаментальних основ впровадження, реалізації та використання автоматизованого тестування ПЗ.

Практична значущість: можливість використання результатів роботи як інструкції для ІТ-проектів, які вже використовують або мають намір використовувати автоматизоване тестування для ПЗ.

РОЗДІЛ 1.

ТЕОРЕТИЧНІ ЗАСАДИ ВИКОРИСТАННЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ПРОГРАМНИХ ДОДАТКІВ

1.1 Актори і функції процесу автоматизованого тестування

Перед тим, як приступити до аналізу автоматизованого тестування, слід розібратися з терміном «тестування», зрозуміти його головну мету та призначення. Тестування – це процес перевірки відповідності функціональності продукту заявленим вимогам, що здійснюється через спостереження за його роботою в штучних умовах, а також за допомогою обмеженого набору тестів, підібраних специфічним чином.

Тестування – це вміння виявляти дефекти в програмному забезпеченні. Для ефективного тестування потрібні обдумані підходи та увага до якості випадків тестування. Важливо, щоб вони були ефективні в виявленні дефектів, прикладні та економічні у виконанні та обслуговуванні. Баланс між цими аспектами допомагає уникнути надмірних витрат та забезпечити ефективність тестування.

Автоматизоване тестування, також відоме як автоматизація тестування, – це метод перевірки програмного забезпечення, що використовує спеціальні програмні інструменти для виконання заданого набору тестових сценаріїв. Такий інструмент може автоматично вводити тестові дані, порівнювати очікувані результати з фактичними та генерувати докладні звіти про виконані тести.

Автоматизація тестування – це окрема навичка, відмінна від самого тестування. Для отримати вигоди від автоматизації, потрібно ретельно вибирати тести і правильно їх реалізовувати. Якість автоматизованого тесту не залежить від якості самого тесту. Автоматизація не впливає на ефективність тесту, вона лише знижує його вартість і складність обслуговування. Автоматизовані тести зазвичай економніші у виконанні, але вони вимагають більше зусиль для створення та

- **Створення тестових сценаріїв:** аналіз вимог до програмного забезпечення і розробку сценаріїв, які описують, як користувачі взаємодіють з програмою. Кожен сценарій повинен бути покритий тестовими випадками.
- **Розробка тестових випадків:** конкретизація тестових сценаріїв у вигляді послідовності дій, що повинні бути виконані під час тестування. Кожен тестовий випадок має бути детально описаний, включаючи вхідні дані, кроки виконання тесту, очікувані результати та критерії успішності.
- **Налаштування тестового середовища:** підготовка тестового середовища, що включає у себе: налаштування тестових баз даних, розгортання тестових серверів, імітацію зовнішніх систем, необхідних для тестування, і налаштування доступів.
- **Виконання тестів:** запуск тестів на різних конфігураціях і платформах та збір результатів виконання.
- **Аналіз результатів:** детальний розбір отриманих результатів від запуску автоматизованих тестів для виявлення та реєстрації дефектів, оцінки покриття тестування та якості тестових випадків.
- **Звітність:** підготовка детальної аналітики та звітів з результатами тестування для учасників проєкту, включаючи розробників, тестувальників, менеджерів проєкту та замовників. Ці звіти можуть містити інформацію про виявлені помилки, покриття тестування, тренди тестування, деталі виконання тестування тощо.
- **Інтеграція з циклами розробки:** інтеграція автоматизованих тестів у постійний процес розробки для автоматичного виконання тестів під час кожного релізу програмного забезпечення. Це допомагає виявляти проблеми якнайшвидше та забезпечує стабільність програмного забезпечення.
- **Підтримка тестів:** постійне оновлення тестових сценаріїв та випадків для врахування змін у вимогах до програмного забезпечення та для забезпечення високої якості тестів.

1.2 Аналіз видів і напрямків автоматизованого тестування

Автоматизоване тестування включає різні види та напрямки, що охоплюють різні аспекти тестування програмного забезпечення. Розгляд цього підрозділу дозволить уявити різноманіття можливостей, які відкриває використання автоматизованих тестів в індустрії розробки ІТ-проектів.

Визначимо основні типи автоматизованих тестів:

1. Функціональне тестування – це процес перевірки функціональності програмного забезпечення за допомогою автоматизованих тестових сценаріїв та інструментів. Цей вид тестування спрямований на визначення того, чи відповідає реальна реалізація програмного продукту його вимогам та специфікаціям. Можна виділити основні типи функціонального тестування:

- **Модульне тестування (Unit Testing)** – автоматизоване тестування окремих функціональних одиниць програми для перевірки їхньої коректності та взаємодії з іншими модулями. Модульне тестування є ключовою складовою процесу розробки програмного забезпечення, яке спрямоване на забезпечення якості коду. Воно визначається як перевірка найменших тестових частин програми, таких як процедури, інтерфейси, функції та класи, щоб визначити, чи вони готові до використання. Цей процес виконується розробниками для переконання, що код відповідає вимогам специфікації і дизайну.

- **Інтеграційне тестування (Integration Testing)** – автоматизоване тестування взаємодії між різними функціональними модулями, щоб переконатися в їхній відповідності специфікаціям. Дана техніка дозволяє виявляти помилки і одночасно конструювати структуру програми. Мета інтеграційного тестування – перевірити взаємодію між модулями та одиницями програми.

- **Системне тестування (System Testing)** – використання автоматизованих тестів для перевірки всієї системи як єдиної функціональної одиниці. Системне тестування виконується після завершення розробки програми за

вказаними вимогами та не потребує від знань про вихідний код. Це серія різних тестів, головна мета яких – повністю протестувати додаток. Кожен тест має власну ціль та перевіряє, чи правильно інтегрована система і чи виконує вона потрібні функції.

Після знайомства з типами функціонального тестування давайте визначимо їх головні відмінності, щоб розуміти на якому етапі розробки ПЗ нам доцільно використовувати кожен із типів. У табл. 1.1 показано головні відмінності між основними типами функціонального тестування.

Таблиця 1.1 – Головні відмінності між типами функціонального тестування

Тип	Модульне	Інтеграційне	Системне
Стадія впровадження	На початку розробки	По середині розробки	По завершенню розробки
Об'єкт тестування	Одиниці програми (процедури, функції, класи)	Групи з'єднаних модулів або компонентів	Вся система або додаток
Доступ до коду	Обов'язковий	В залежності від вимог для тестування	Відсутній
Ціль	Перевірка правильності роботи окремих частин програми	Перевірка взаємодії між різними частинами програми	Перевірка повної функціональності системи згідно вимог
Завершеність	Проводиться перед інтеграцією та системним тестуванням	Проводиться після модульного тестування та перед системним	Останній етап тестування після інтеграції всіх частин
Актори тестування	Виконується розробниками	Виконуються як розробниками, так і тестувальниками	Виконується тестувальниками

2. Навантажувальне тестування – це процес визначення меж стійкості системи та її продуктивності за допомогою автоматизованих тестових сценаріїв та інструментів. Основна мета цього виду тестування полягає в тому, щоб оцінити, як система поводить себе під різними навантаженнями та визначити, чи здатна вона

витримати очікувані об'єми роботи. Можна виділити основні типи навантажувального тестування:

- **Тестування стійкості (Stress Testing)** – це процес оцінки, наскільки програмний продукт може продовжувати працювати в умовах непередбачуваних або нестандартних ситуацій, таких як неправильний ввід даних, виток пам'яті, велике навантаження на систему або втрата з'єднання з мережею. Тестування стійкості спрямоване на виявлення помилок, що можуть виникнути в умовах стресу або поломок, і визначає, як продукт реагує на такі умови і чи може він відновлюватися після них.

- **Тестування продуктивності (Performance Testing)** – це процес визначення швидкості, масштабованості та ефективності програмного продукту або системи за різних умов використання. Основна мета полягає в тому, щоб з'ясувати, як продукт веде себе під великим навантаженням або в умовах обмежених ресурсів, таких як обсяг даних, кількість одночасних користувачів або потужність обчислювального обладнання. Тестування продуктивності допомагає виявити слабкі місця системи та ідентифікувати можливі проблеми зі швидкістю або масштабованістю перед випуском в експлуатацію.

Після знайомства з основними типами навантажувального тестування давайте визначимо їх головні відмінності та особливості, це все детально та лаконічно продемонстровано у табл. 1.2.

Таблиця 1.2 – Головні відмінності між типами навантажувального тестування

Тип	Тестування стійкості	Тестування продуктивності
Основна мета	Виявлення помилок та непередбачуваних поведінок	Визначення швидкості та ефективності під навантаженням
Умови виконання	Стресові умови, несправності, неправильний ввід даних	Великий обсяг даних, багатокористувацьке середовище
Ціль	Визначення, як продукт веде себе під стресом	Виявлення слабких місць та ідентифікація проблем з швидкістю та масштабованістю

Умови проведення	Вимушене спотворення умов роботи, нестандартні сценарії	Симуляція великого обсягу робочих навантажень
Об'єм даних	Може бути обмежений, може бути використано тестові дані	Зазвичай використовуються великі обсяги даних
Ресурси	Може включати обмеження пам'яті, процесора та мережі	Тестування на різних обсягах навантаження та ресурсах
Залежність від коду	Зазвичай не потребує доступу до вихідного коду	Може вимагати аналізу коду для оптимізації
Актори тестування	Виконується тестувальниками	Виконується тестувальниками

3. Тестування безпеки – це процес визначення вразливостей та оцінки рівня захищеності програмного забезпечення, застосовуючи автоматизовані засоби тестування та сценарії. Основна мета цього виду тестування полягає в ідентифікації можливих точок вразливості та виявленні потенційних загроз для безпеки системи. Можна виділити основні типи тестування безпеки:

- **Тестування на проникнення (Penetration Testing)** – це процес активного аналізу системи або програмного забезпечення з метою виявлення потенційних слабкостей в безпеці, які можуть бути використані зловмисниками для несанкціонованого доступу або атак на систему. Включає в себе спроби злому, використання вразливостей, аналіз вимог до безпеки та інших методів, щоб перевірити, наскільки добре захищена система. Тестування на проникнення допомагає виявити слабкі місця в захисті даних і інфраструктури, щоб їх можна було виправити перед тим, як їх використають зловмисники.

- **Аналіз вразливостей (Vulnerability Analysis)** – це процес виявлення, оцінки та класифікації потенційних вразливостей в програмному забезпеченні, інформаційних системах або мережевих інфраструктурах. Це включає в себе сканування, тестування та аналіз коду для ідентифікації слабких місць, які можуть бути використані зловмисниками для атак або несанкціонованого доступу. Метою аналізу вразливостей є забезпечення безпеки інформаційних активів та запобігання можливим загрозам шляхом виправлення виявлених проблем безпеки.

Після знайомства з основними типами тестування безпеки давайте визначимо їх головні відмінності та особливості, це все детально та лаконічно продемонстровано у табл. 1.3.

Таблиця 1.3 – Головні відмінності між типами тестування безпеки

Тип	Тестування на проникнення	Аналіз вразливостей
Основна мета	Виявлення потенційних вразливостей через атаки	Виявлення та оцінка вразливостей в системі
Підхід	Активні атаки на систему або програмне забезпечення	Аналіз і виявлення слабких місць у системі
Проведення	Спроби злому, експлуатація вразливостей, збір інформації	Сканування, аналіз коду, тестування на вразливості
Час проведення	Часово-обмежені сеанси тестування з підсумковою звітністю	Зазвичай проводиться періодично або при кожній зміні системи
Переваги	Покращує безпеку шляхом виявлення реальних слабкостей	Виявлення вразливостей до їх експлуатації, дозволяє попереджати потенційні загрози
Обмеження	Може бути обмежений впливом на реальну систему	Може не виявити всі можливі вразливості, особливо ті, що залежать від контексту системи
Актори тестування	Зазвичай виконується командою зовнішніх експертів	Може бути виконаний внутрішньою командою безпеки організації

4. Тестування інтерфейсу користувача – це процес визначення правильності та ефективності взаємодії користувача з програмним забезпеченням, використовуючи автоматизовані тестові сценарії та інструменти. Основна мета цього виду тестування полягає в перевірці, чи відповідає інтерфейс користувача зазначеним вимогам, чи забезпечує зручність та задоволення користувача при використанні програмного продукту. Можна виділити основні типи тестування інтерфейсу користувача:

- **Тестування відповідності дизайну (Layout Testing)** – це процес перевірки, чи відповідає розроблений програмний продукт або система затвердженому дизайну чи специфікаціям. Його мета полягає в тому, щоб переконатися, що реалізація програми відповідає вимогам, що були визначені на етапі проєктування. Таке тестування допомагає виявити відхилення від заданих стандартів, дизайну чи інших вимог, що можуть вплинути на якість або функціональність продукту.

- **Тестування взаємодії користувача (User Interaction Testing)** – це процес перевірки користувацького інтерфейсу програмного забезпечення для забезпечення його зручності, доступності та відповідності вимогам користувачів. Це включає в себе перевірку елементів інтерфейсу, таких як кнопки, поля вводу, меню, діалогові вікна тощо, на предмет їх правильності, відповідності дизайну, простоти використання та інші аспекти, які можуть вплинути на зручність взаємодії користувача з програмою.

Після знайомства з основними типами тестування інтерфейсу користувача давайте визначимо їх головні відмінності та особливості, це все детально та лаконічно продемонстровано у табл. 1.4.

Таблиця 1.4 – Головні відмінності між типами тестування інтерфейсу користувача

Тип	Тестування відповідності дизайну	Тестування взаємодії користувача
Мета	Перевірка відповідності програмного забезпечення дизайну	Перевірка зручності, доступності та ефективності інтерфейсу
Об'єкт тестування	Елементи інтерфейсу, графічні компоненти	Елементи інтерфейсу, користувацькі дії та взаємодія
Переваги	Визначення відповідності стандартам дизайну	Забезпечення оптимального досвіду користувача
Залежність від вхідного коду	Може не вимагати доступу до вихідного коду	Може потребувати доступу до вихідного коду для аналізу

Обмеження	Може не виявити проблем, пов'язаних зі зручністю використання	Може не виявити проблем, пов'язаних з графічним виглядом
Актор тестування	Зазвичай виконується командою розробників	Може виконуватися командою тестувальників

Розглянувши різноманітні види і напрямки автоматизованого тестування, можна зробити висновок, що кожен з них відіграє важливу роль у забезпеченні якості програмного забезпечення. Розмаїття підходів дозволяє розробникам вибирати оптимальні стратегії тестування відповідно до специфіки їхнього проєкту та його вимог.

Функціональне тестування забезпечує перевірку окремих компонентів та їх взаємодії, надаючи впевненість у правильності реалізації функціональності. Навантажувальне тестування дозволяє визначити межі стійкості системи під навантаженням, тим самим підтверджуючи її надійність у реальних умовах експлуатації. Тестування безпеки та тестування інтерфейсу користувача важливі для забезпечення захисту від потенційних загроз та задоволення потреб користувачів.

Кожен напрямок має свої особливості та завдання, і в їх комбінації вони створюють повноцінний набір тестів, спрямованих на визначення різноманітних аспектів якості програмного продукту. Враховуючи це різноманіття, розробники можуть ефективно впроваджувати автоматизоване тестування, забезпечуючи високу ефективність, швидкість та надійність в процесі розробки.

1.3 Переваги та недоліки автоматизованого тестування

Автоматизоване тестування є потужним та важливим елементом сучасного процесу розробки ІТ-проєктів, проте його використання пов'язане із специфічними

перевагами та недоліками. Розгляд цього підрозділу дозволить глибше зрозуміти вплив автоматизованого тестування на якість та ефективність розробки.

Для початку ознайомимося з перевагами автоматизованого тестування:

- **Швидкість виявлення помилок:** процес запуску та виконання автоматизованих тестів відбувається набагато швидше, в порівнянні з ручним тестуванням. Це дозволяє швидше і ефективніше знаходити дефекти в роботі програми та економить дорогоцінний час для команди розробки.
- **Економія коштів:** за рахунок регулярного та швидкого запуску автоматизованих тестів, дефекти в роботі ІТ-проєкту знаходяться та виправляються на ранніх стадіях, що в свою чергу економить кошти замовника.
- **Точність та надійність:** запуск та виконання автоматизованих тестів відбувається за допомогою комп'ютера, що в свою чергу, мінімізує ризик людських помилок та впливає на якість та ефективність ІТ-проєкту.
- **Універсальність:** автоматизоване тестування дозволяє здійснювати операції на тисячах пристроїв, що неможливе під час ручного тестування, це збільшує область тестування нашого ІТ-проєкту.
- **Ефективність:** автоматизоване тестування дозволяє економити час та ресурси, сприяючи підтримці стабільної якості продукту.
- **Покриття:** автоматизоване тестування дозволяє забезпечити високий рівень покриття, охоплюючи різноманітні тестові сценарії та шари програмного забезпечення.

Але на превеликий жал, не існує нічого ідеального і автоматизоване тестування не є виключенням. Давайте ознайомимося та проаналізуємо недоліки автоматизації:

- **Вартість впровадження:** автоматизоване тестування не підходить для малих проєктів, або проєктів з обмеженим фінансуванням, адже вимагає значних витрат на впровадження, включаючи вибір інструментів, навчання команди та розробку тестових скриптів.
- **Складність:** ручне тестування є набагато простішим в порівнянні з автоматизованим та вимагає меншого рівня кваліфікації робітників. Також

автоматизоване тестування вимагає більш прискіпливого планування та управління ризиками.

- **Підтримка та обслуговування:** автоматизовані тести вимагають регулярного догляду та підтримки, навіть невелика зміна в програмі може вплинути на процес запуску та виконання тестів.

- **Вибір інструмента:** існує велика кількість засобів для реалізації автоматизованого тестування, це ускладнює проблему вибору інструмента, що у свою чергу, може впливати на планування та вибір стратегії тестування.

Після детального розгляду переваг та недоліків автоматизованого тестування можна зробити висновок, що вибір цього підходу до тестування важливий та повинен базуватися на конкретних вимогах та умовах проєкту.

Переваги автоматизованого тестування, такі як ефективність, універсальність, точність, надійність та швидкість, роблять його незамінним інструментом для забезпечення якості програмного забезпечення. Зокрема, на великих проєктах та в умовах постійних змін цей метод може допомогти економити час та ресурси.

Однак важливо враховувати недоліки автоматизованого тестування, такі як вартість впровадження, складність, підтримка та обслуговування. Ці аспекти вимагають уважної стратегії та розгляду альтернативних методів тестування.

Отже, ефективне використання автоматизованого тестування полягає в обережному аналізі всіх його переваг та недоліків в контексті конкретного проєкту, щоб досягти оптимального балансу між якістю, витратами та часом.

1.4 Висновок до розділу

У першому розділі дипломної роботи розглянуто теоретичні аспекти автоматизованого тестування ПЗ. Було проаналізовано сутність поняття

автоматизованого тестування, визначено його основні функції та місце у життєвому циклі тестування ПЗ.

Крім того, проведено докладний аналіз різновидів автоматизованого тестування. Надано чітке визначення кожного типу автоматизованого тестування та здійснено порівняльну характеристику їх особливостей, переваг та недоліків. Цей аналіз дозволив розкрити широкий спектр можливостей, які надає автоматизоване тестування, а також визначити області його найбільш ефективного застосування.

Окрім того, у дослідженні були детально описані переваги та недоліки автоматизованого тестування. Зазначено, що серед переваг можна виділити підвищену швидкість виконання тестів, зниження витрат на тестування у довгостроковій перспективі, збільшення точності та надійності результатів тестування. Проте, також відзначено, що для успішного впровадження автоматизованого тестування необхідно враховувати його обмеження, такі як складність створення та підтримки тестових скриптів, вартість розробки та підтримки автоматизованих тестів, а також обмежену можливість виявлення деяких видів дефектів.

Отже, перший розділ дипломної роботи висвітлює основні аспекти теорії автоматизованого тестування, зокрема його типи, переваги та недоліки, що становлять теоретичну основу для подальшого дослідження та впровадження автоматизованого тестування на проєкті.

РОЗДІЛ 2.

АНАЛІЗ МОДЕЛЕЙ ТА МЕТОДІВ ТЕСТУВАННЯ ПРОГРАМНИХ ДОДАТКІВ

2.1 Аналіз проєкту та визначення ключових областей для запровадження автоматизованого тестування

Для реалізації процесу автоматизованого тестування був обраний ІТ-проєкт «Dropbox», адже даний проєкт надає можливість продемонструвати широкий спектр функцій та аспектів, які потрібно тестувати в реальному середовищі з великою кількістю користувачів та вимогами до безпеки та стабільності.

Dropbox – це хмарний сервіс зберігання та синхронізації файлів, який дозволяє користувачам зберігати свої дані в Інтернеті і отримувати до них доступ з будь-якого пристрою з підключенням до Інтернету. Основна ідея Dropbox полягає в тому, щоб зробити зберігання та обмін файлами максимально простим і зручним для користувачів.

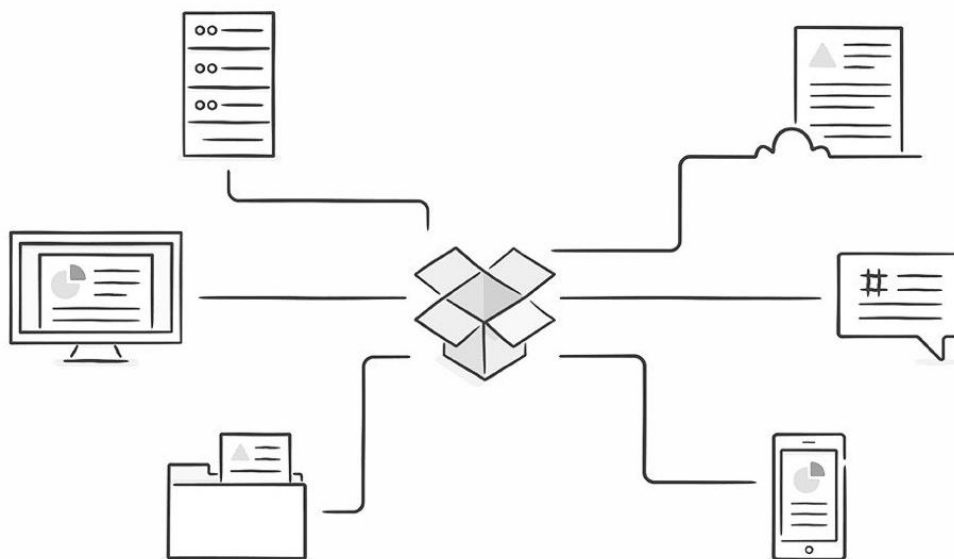


Рисунок 2.1 – Процес зберігання та обмін файлами в Dropbox

Dropbox – це універсальний інструмент для зберігання, синхронізації та обміну файлами через хмару, який забезпечує зручний доступ до даних з будь-якого пристрою. Користувачі можуть використовувати офіційні додатки Dropbox на комп'ютерах, смартфонах та планшетах, а також отримувати доступ через веб-версію на будь-якому пристрої без встановлення спеціального програмного забезпечення. Такий різноманітний спектр підтримуваних пристроїв робить Dropbox зручним і універсальним інструментом для роботи з файлами у будь-яких умовах та на будь-яких пристроях.

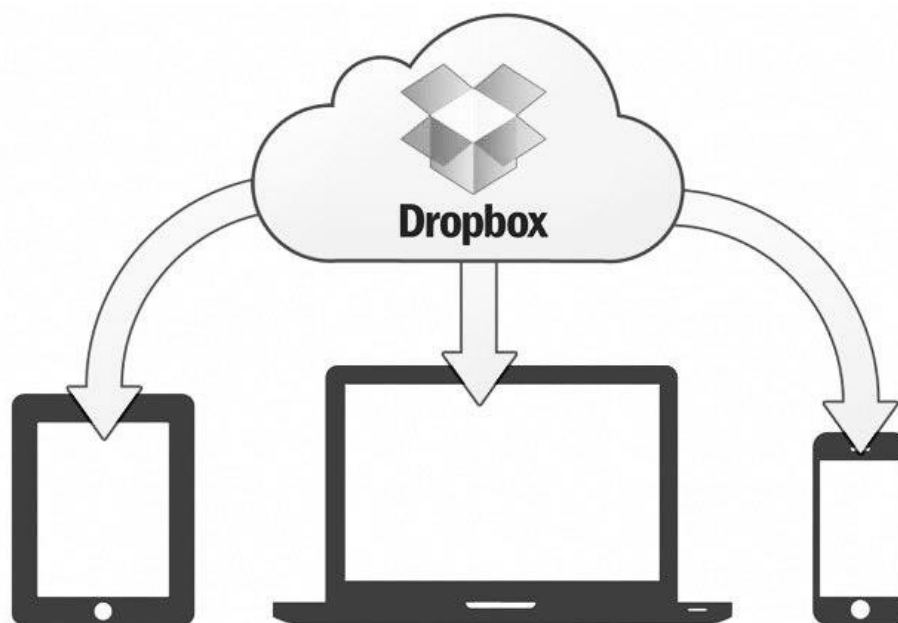


Рисунок 2.2 – Доступ до Dropbox з різних пристроїв

Dropbox пропонує широкий спектр функцій та переваг, які роблять його популярним серед користувачів та бізнесів. Ось деякі з них:

- **Зберігання та синхронізація файлів.** Dropbox дозволяє користувачам зберігати свої файли в хмарі і синхронізувати їх між різними пристроями, щоб мати до них доступ з будь-якого місця.
- **Спільна робота.** Dropbox надає можливість ділитися файлами та папками з іншими користувачами, що дозволяє спільно працювати над проєктами та документами.

- **Резервне копіювання і відновлення даних.** Dropbox автоматично резервує копії файлів, що дозволяє відновлювати видалені або змінені файли на певну дату чи версію.
- **Мобільний доступ.** Існують офіційні мобільні додатки Dropbox для різних платформ, що дозволяють користувачам отримувати доступ до своїх файлів навіть з мобільних пристроїв.
- **Захист даних.** Dropbox використовує шифрування та інші заходи безпеки для захисту конфіденційності даних користувачів, що робить його надійним середовищем для зберігання файлів.
- **Інтеграція з іншими сервісами.** Dropbox може інтегруватися з іншими популярними сервісами та програмами, що дозволяє розширити його функціональність та зручність використання.
- **Віддалений доступ до файлів.** Користувачі можуть отримувати доступ до своїх файлів через будь-який браузер на будь-якому пристрої, що підключений до Інтернету.
- **Робота в автономному режимі.** Dropbox дозволяє працювати з файлами навіть без Інтернет-з'єднання, а підключившись до мережі, автоматично синхронізує зміни.

Дані функції та переваги роблять Dropbox потужним інструментом для зберігання, обміну та роботи з файлами в онлайн-середовищі.

Після знайомства з проектом та визначення його головного призначення, можна перейти до більш глибокого аналізу нашої системи. Для початку ми здійснимо побудову SIPOC-діаграми для визначення важливих аспектів процесу.

SIPOC-діаграма – це інструмент для моделювання та аналізу процесу або системи. SIPOC-діаграма допомагає визначити ключові елементи процесу або системи, з'ясувати залежності між ними та зрозуміти взаємозв'язки з клієнтами та постачальниками.

Перед побудовою діаграми нам варто визначити ключові елементи:

1. **Постачальники:**

- **Інтернет-провайдери** – постачальники інтернет-з'єднання, які забезпечують доступ користувачів до сервісу через інтернет.

- **Розробники програмного забезпечення** – компанії, що створюють та підтримують програмне забезпечення, необхідне для роботи системи.

- **Технологічні компанії** – постачальники хмарних технологій, такі як AWS або Google Cloud, які забезпечують інфраструктуру для зберігання даних та обробки запитів.

2. Вхід:

- **Користувачі** – особи або організації, що завантажують файли та дані на хмарне сховище системи.

- **Дані** – файли різних форматів, які користувачі завантажують у свої облікові записи системи.

- **Інформація про обліковий запис користувача** – дані, що ідентифікують користувача та включають інформацію про його обліковий запис.

3. Процес:

- **Завантаження файлів на сервери** – передача файлів з пристроїв користувачів на сервери через інтернет.

- **Зберігання та організація файлів у хмарному сховищі** – зберігання та індексація файлів на серверах для подальшого доступу користувачів.

- **Забезпечення безпеки та конфіденційності даних** – застосування технологій шифрування та інших заходів для захисту даних користувачів.

- **Синхронізація даних між пристроями користувачів** – автоматична синхронізація файлів між пристроями користувачів для доступу з будь-якого пристрою.

- **Надання доступу до файлів для спільної роботи** – надання можливості користувачам ділитися файлами та спільно працювати з ними.

4. Вихід:

- **Доступ до файлів з будь-якого пристрою з інтернет-підключенням** – можливість отримання доступу до файлів з будь-якого пристрою через веб-сайт або мобільний додаток.

- **Поділ файлів та тек між користувачами** – функціонал, який дозволяє користувачам ділитися файлами для спільної роботи або перегляду.

- **Резервне копіювання файлів та можливість відновлення** – автоматичне резервне копіювання файлів на серверах та можливість відновлення втрачених файлів.

5. Клієнти:

- **Корпоративні клієнти** – бізнеси та організації, які використовують систему для спільної роботи над проектами.

- **Індивідуальні користувачі** – особи, які використовують систему для особистих цілей, таких як зберігання фотографій чи документів.

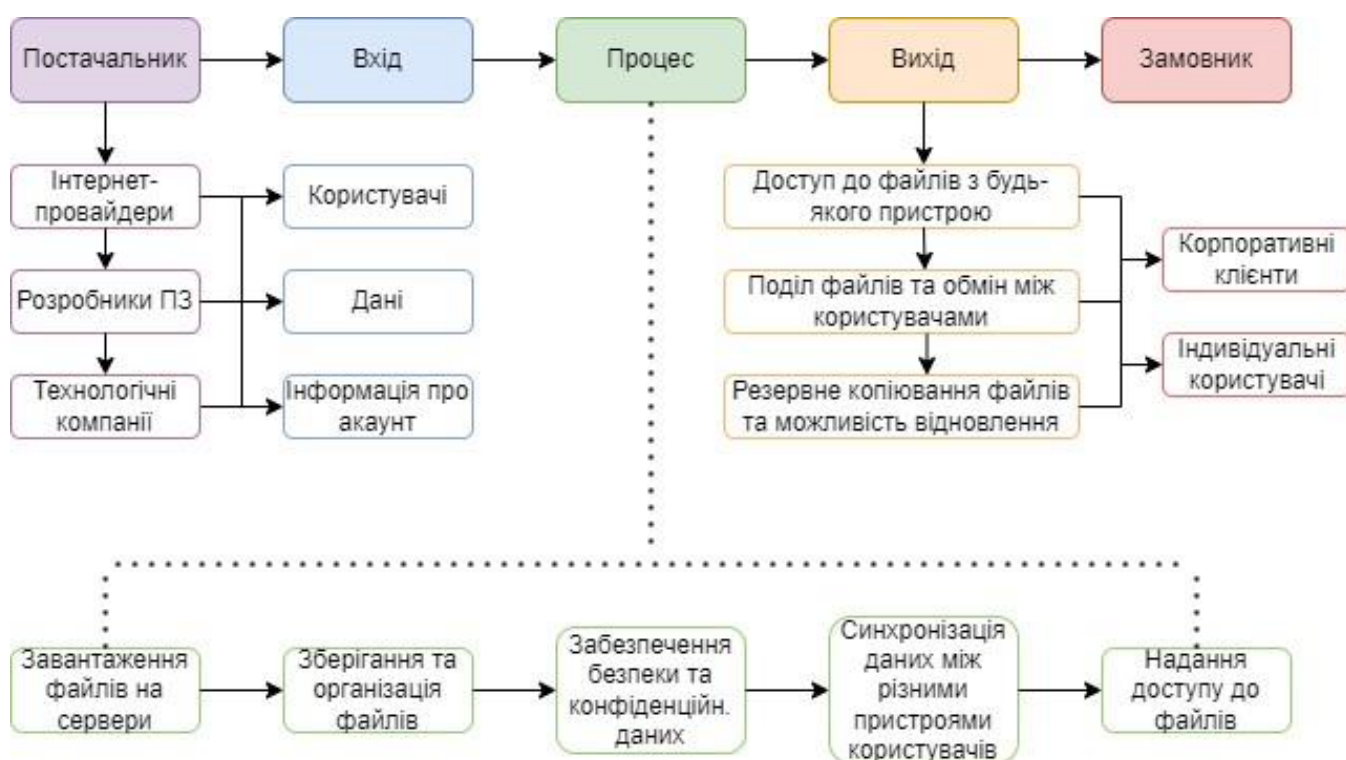


Рисунок 2.3 – SIPOC-діаграма для системи Dropbox

Розроблена SIPOC-діаграма ідентифікує всі ключові елементи, що становлять процес функціонування проекту, від постачальників до клієнтів, та надає детальні пояснення кожного з них. Дозволяє нам зрозуміти взаємозв'язки між різними елементами процесу, визначити ролі та відповідальності різних учасників процесу, включаючи постачальників, виконавців та клієнтів. Аналіз SIPOC-діаграми дозволяє визначити головні потреби клієнтів, це все нам стане у нагоді

при плануванні та розробці тестової документації та написанні майбутніх автоматизованих тестів.

Після детального опису моделі та розробки SIPOC-діаграма, здійснимо побудувати функціональної моделі бізнес-процесу. Контекстна діаграма IDEF0 – це графічне зображення бізнес-процесу, яке показує взаємодію процесу з його зовнішнім середовищем. Вона визначає головний процес та зовнішні сутності, з якими він взаємодіє, але не деталізує сам процес. Це допомагає зрозуміти загальну структуру та взаємозв'язки між процесом та його навколишнім середовищем без зайвих деталей.

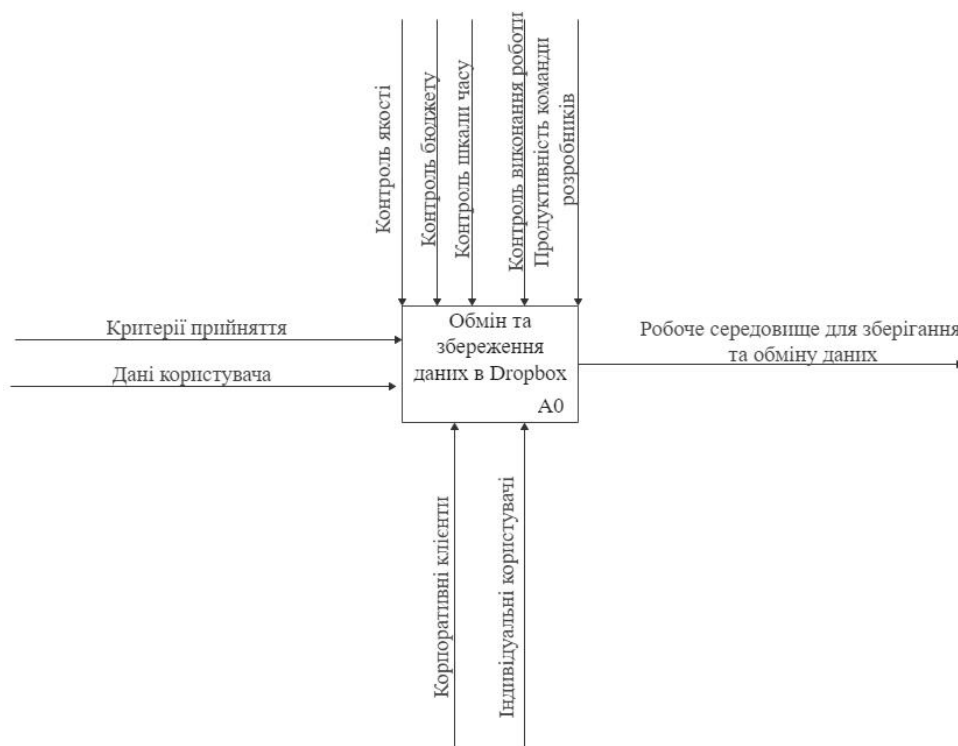


Рисунок 2.4 – Контекстна діаграма IDEF0 для бізнес-процесу «Обмін та збереження даних в Dropbox»

Після попередніх кроків у нас вже є необхідна інформаційна база для побудови діаграми бізнес-процесу в нотації BPMN. Діаграма бізнес-процесу в нотації BPMN (Business Process Model and Notation) – це графічне зображення бізнес-процесу, яке дозволяє моделювати, аналізувати та управляти бізнес-процесами в організації. Нотація BPMN надає стандартний набір символів та правил для створення діаграм, що дозволяє розуміти, комунікувати та

вдосконалювати процеси ефективним способом. Вона дозволяє візуалізувати кроки, учасників, потоки даних та подій у бізнес-процесі, що сприяє кращому розумінню та управлінню процесами в організації.

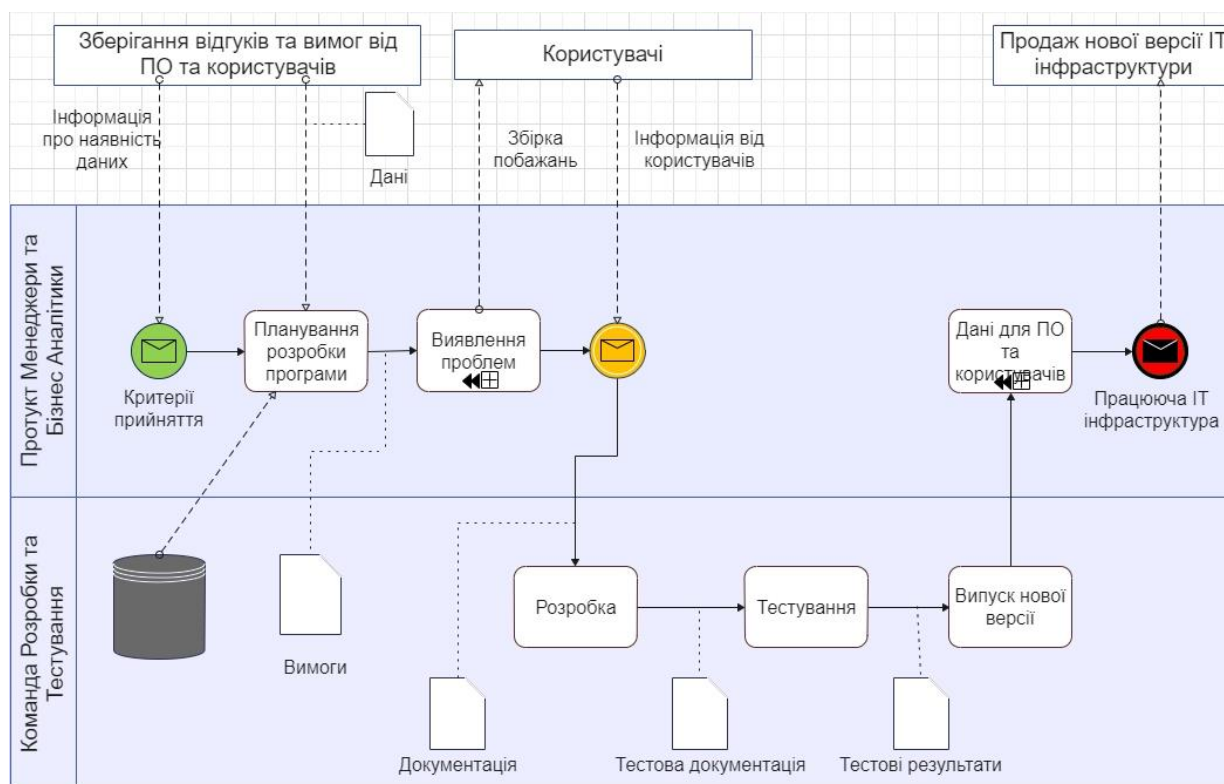


Рисунок 2.5 – BPMN-діаграма для системи Dropbox

На діаграмі BPMN продемонстровано життєвий цикл ІТ-проєкту та роль кінцевого користувача в ньому.

На попередніх кроках було здійснено детальний аналіз системи, дана інформація нам допоможе виділити головні аспекти та зони, які потрібно буде протестувати в майбутньому за допомогою автоматизованих тестів:

- **Функціональність веб-інтерфейсу (UI):** слід перевірити всі найважливіші функцій, які користувач може використовувати через веб-інтерфейс. Це включає в себе завантаження, скачування, переміщення та видалення файлів, роботу з папками, створення та видалення посилань для спільного доступу тощо.
- **Інтеграція з іншими службами (API):** слід перевірити інтеграцію нашої системи з іншими сервісами та додатками. Важливо переконатися, що ці інтеграції працюють належним чином та не впливають на загальну продуктивність та безпеку додатку. Дані перевірки будуть здійснюватися за допомогою тестування

API (Application Programming Interface). API – це набір правил і протоколів, які визначають, як різні програми можуть взаємодіяти одна з одною. API надає можливість одній програмі (або частині програми) використовувати функції або послуги, які надаються іншими програмами або сервісами.

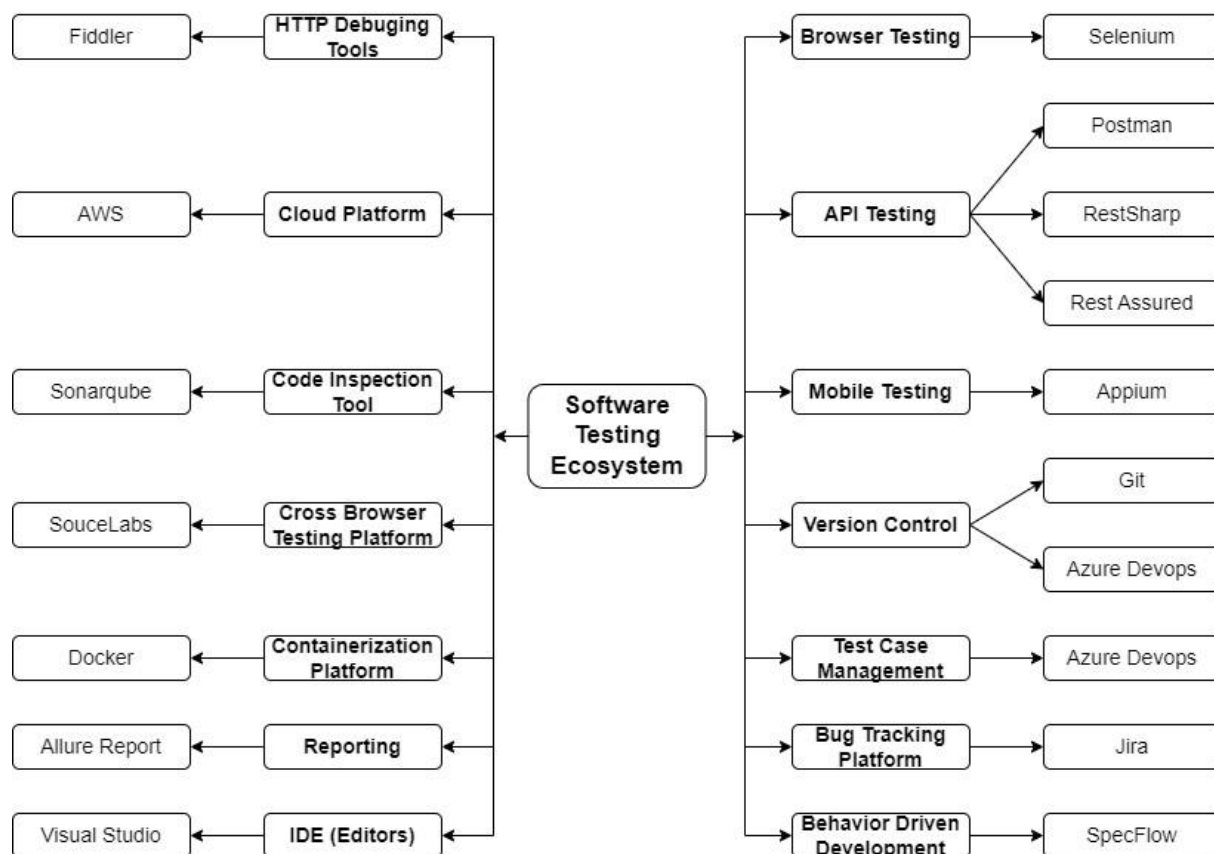


Рисунок 2.6 – Екосистема тестування

Використання автоматизованого тестування допоможе забезпечити високий рівень якості та надійності нашої системи, забезпечуючи швидке виявлення помилок при кожній зміні або вдосконаленні та ефективну реакцію на них.

2.2 Аналіз вимог та створення тестової документації

Перед початком впровадження автоматизованого тестування необхідно здійснити ретельний аналіз вимог до програми та на основі проведеного аналізу розробити якісну тестову документації. Цей етап дуже важливий, адже тестова

документація буде виступати у ролі фундаменту для нашого майбутнього процесу автоматизованого тестування.

Аналіз вимог для тестування становить ключовий етап у підготовці до автоматизованого тестування IT-проекту. Цей процес передбачає докладне вивчення як функціональних, так і нефункціональних вимог до системи з метою визначення стратегії тестування, спрямованої на забезпечення повноти та якості покриття тестами.

Під час аналізу вимог встановлюється обсяг тестування, що охоплює всі аспекти функціональності та надійності системи. Тестування має враховувати різноманітні сценарії використання системи та її взаємодію з іншими компонентами інфраструктури. Крім того, необхідно врахувати нефункціональні вимоги, такі як продуктивність, масштабованість, безпека та інші аспекти, і переконатися, що майбутні сценарії тестування відповідають цим вимогам.

На основі проведеного аналізу розробляється тестова документація, яка включає в себе стратегію тестування, план тестування, тест-кейси, сценарії тестування та інші важливі аспекти тестування програмного забезпечення. Основна мета цієї документації полягає в забезпеченні повноти тестування та визначенні відповідності програмного забезпечення вимогам та очікуванням. Такий підхід дозволяє забезпечити ефективне тестування програми та підвищити його якість та надійність.

Якісно розроблена тестова документація повинна відповідати зазначеним критеріям, щоб бути ефективною та корисною для процесу тестування. Ось деякі з цих критеріїв:

- **Чіткість та зрозумілість.** Документація повинна бути написана таким чином, щоб будь-який член команди, який виконує тестування, міг зрозуміти її з легкістю. Технічні терміни повинні бути пояснені, а вказівки та інструкції повинні бути зрозумілі для всіх.
- **Повнота та охоплення.** Документація повинна включати в себе всі необхідні елементи тестування, включаючи стратегію тестування, плани, тест-кейси, сценарії, очікувані результати та критерії завершення.

- **Адаптованість до змін.** Тестова документація повинна бути гнучкою та здатною до швидкого оновлення відповідно до змін у вимогах або функціональності програми.

- **Доступність та доступність.** Документація повинна бути доступною всім членам команди, які займаються тестуванням, і вони повинні мати легкий доступ до неї у будь-який час. Це може включати збереження документації у спільному обліковому записі або системі контролю версій.

- **Відстеження прогресу та результатів.** Документація повинна допомагати відстежувати прогрес тестування та фіксувати результати. Це дозволяє команді відстежувати, які тести були виконані, які пройшли успішно, а які потребують удосконалення.

- **Структурованість та організація.** Документація повинна бути організованою та структурованою таким чином, щоб зробити її легкою у використанні та навігації. Це може включати розділення на різні секції, використання заголовків та нумерації для підсилення логічної послідовності.

Після детального аналізу вимог було розроблено тестову документацію, що допоможе нам у запровадженні автоматизованого тестування. Основні компоненти тестової документації:

- **Тест-план (Test Plan)** – це документ, який описує загальний підхід до тестування, включаючи стратегію тестування, ресурси, розклад тестування та інші важливі аспекти. Головна мета тест-плану – забезпечити інформацію про те, як буде виконуватися тестування, і зробити його більш управлінським та контрольованим процесом.

- **Тестова стратегія (Test Strategy)** – це документ, який визначає загальний підхід до тестування в рамках проєкту чи організації. Вона визначає спосіб, яким будуть визначені, сплановані і виконані тести, а також яким буде оцінюватися якість продукту після тестування. Тестова стратегія допомагає узгоджувати різні аспекти тестування для досягнення високої якості продукту.

- **Тест-кейси (Test Cases)** – це документ, який містить конкретні інструкції та умови для проведення тестування певної функціональності чи аспекту

програмного забезпечення. Кожен тест-кейс представляє собою конкретний сценарій тестування, який включає в себе вхідні дані, дії, які повинні бути виконані, і очікувані результати.

- **Матриця відстежування (Traceability Matrix)** – це інструмент у сфері тестування програмного забезпечення, який допомагає відстежувати взаємозв'язок між різними артефактами в процесі розробки та тестування. Одним з найпоширеніших прикладів використання матриці відстежування є відстеження вимог до тест-кейсів.



Рисунок 2.7 – Послідовність розробки тестової документації

Після розробки тестової документації ми впевнено можемо приступати до впровадження автоматизованого тестування, адже ми чітко знаємо в якому напрямку нам рухатися.

Аналіз вимог та розробка тестової документації є необхідним етапом впровадження автоматизованого тестування в ІТ-проектах. Цей процес дозволяє систематично визначати ключові аспекти тестування, забезпечує ефективне вибір тест-кейсів для автоматизації та створення необхідної тестової документації. Аналіз вимог визначає обсяг тестування, встановлює основні вимоги до програмного продукту, ідентифікує критичні функціональності та етапи, які можуть бути піддані автоматизації. Це допомагає оптимізувати тестовий процес та спрямовувати зусилля на найбільш критичні частини системи.

Розробка тестової документації забезпечує структурований підхід до тестування, включаючи опис кроків тест-кейсів, очікувані результати та критерії прийняття. Це необхідно для побудови ефективних тестових скриптів та забезпечення повторюваності тестів в різних умовах.

Усі ці аспекти сприяють ефективній реалізації автоматизованого тестування, збільшуючи швидкість виявлення та виправлення помилок, полегшуючи підтримку та забезпечуючи високу якість програмного продукту. Ці етапи визначають успішну

інтеграцію автоматизованого тестування в процес розробки ІТ-проєктів, роблячи його ефективним і надійним.

2.3 Процес та етапи впровадження автоматизованого тестування на проєкті

Після аналізування всіх вимог та розробки тестової документації, настав час для впровадження автоматизованого тестування для нашої системи. Даний крок є дуже важливим в розробці програмного забезпечення, адже він спрямований на підвищення та контроль якості системи. Даний крок дозволить зменшити час, необхідний для виконання тестових сценаріїв, підвищити покриття тестування та забезпечити більш точні результати, що сприяє забезпеченню стабільності та надійності розроблюваного продукту.

Користуючись власним досвідом я можу виділити наступні етапи впровадження автоматизованого тестування:

1. Аналіз та планування. На цьому етапі проводиться детальний аналіз вимог та вибір інструментів для автоматизованого тестування. Розробляється стратегія та план дій для ефективної розробки тестових сценаріїв, що охоплюють функціональні можливості ІТ-проєкту. На даному етапі здійснюються наступні дії:

- Аналізує всі вимоги до програмного забезпечення, які включаються в проєкт. Це можуть бути функціональні та нефункціональні вимоги, специфікації, вимоги до продуктивності, безпеки та інші. аналіз вимоги до процесу автоматизованого тестування.
- На основі аналізу вимог команда вибирає набір інструментів для автоматизованого тестування, які найкраще відповідають потребам проєкту.
- На основі вибраних інструментів та стратегії тестування команда розробляє детальний план дій, який включає в себе календарний план реалізації

автоматизованого тестування, розподіл завдань між учасниками команди, призначення відповідальних осіб за кожен область автоматизації.

2. Підготовка тестового середовища та налаштування тестових даних. На цьому етапі створюється тестове середовище, включаючи підняття тестових серверів та налаштування тестових даних для майбутнього тестування. На даному етапі здійснюються наступні дії:

- Команда забезпечує наявність інфраструктури для проведення тестування, що може включати в себе сервери, середовища віртуалізації, контейнери, бази даних тощо. Тестове середовище повинне відповідати характеристикам реального середовища продукту.

- Інструменти для автоматизованого тестування встановлюються та налаштовуються на тестових серверах.

- Для проведення тестів необхідні вхідні дані, які відображають реальні сценарії використання програмного продукту. Команда створює тестові дані або генерує їх автоматично з використанням спеціалізованих інструментів. Тестові дані повинні бути репрезентативними та охоплювати різноманітні сценарії.

- Забезпечується доступ до тестового середовища для всіх учасників проєкту, які будуть проводити тестування. Це може включати налаштування прав доступу, створення облікових записів для спеціалістів та інших учасників проєкту.

- Після підготовки середовища проводиться валідація його працездатності. Це включає в себе запуск простих тестів для перевірки правильності налаштувань та функціональності середовища.

3. Реалізація тестових сценаріїв. Цей етап передбачає розробку та написання автоматизованих тестів, використовуючи обрані інструменти для автоматизації. На даному кроці здійснюються наступні дії:

- Команда аналізує вимоги до програмного забезпечення та відповідно до цього визначає набір тестових сценаріїв, які необхідно автоматизувати.

- На основі визначених тестових сценаріїв розробники створюють автоматизовані тестові скрипти. Ці скрипти містять послідовність дій, які необхідно виконати для проведення тестування, а також очікувані результати.

- Після написання тестових скриптів їх необхідно налагодити, щоб вони працювали правильно на різних конфігураціях та середовищах. Це може включати в себе вирішення проблем з ідентифікацією елементів інтерфейсу користувача, обробкою винятків та інші аспекти.

- Розроблені тестові скрипти інтегруються з системами керування версіями, щоб забезпечити зберігання, відстеження та керування версіями тестового коду.

- Кожен тестовий сценарій та відповідний йому тестовий скрипт документується, щоб забезпечити зрозумілість та доступність для всіх учасників проєкту.

4. Запуск автоматизованих тестів та аналіз результатів. На даному етапі проводиться запуск тестів та аналіз результатів для виявлення дефектів та недоліків в роботі ІТ-проєкту. На даному етапі відбуваються такі дії:

- Перед запуском тестів команда перевіряє готовність тестового середовища, включаючи доступність необхідних ресурсів, стабільність інфраструктури та правильність конфігурації.

- Обраний набір автоматизованих тестів запускається в автоматичному режимі. Це може бути виконано за допомогою спеціальних інструментів або скриптів, які автоматично запускають тестові сценарії та збирають результати.

- Під час виконання тестів проводиться моніторинг їхнього виконання. Це включає в себе відстеження прогресу, виявлення можливих проблем або помилок та реакцію на них в реальному часі.

- Після завершення запуску тестів збираються результати тестів, які включають в себе інформацію про пройдені тестові сценарії, виявлені помилки, логи виконання тестів та інші важливі дані.

- Отримані результати аналізуються з метою виявлення дефектів, помилок або незгоди з очікуваними результатами. Якщо виявляються проблеми, вони фіксуються та документуються для подальшої виправлення.

- На основі результатів тестування генерується звіт, який містить інформацію про пройдені тестові сценарії, виявлені проблеми, статус проходження тестів та інші важливі відомості.

- На основі звіту та аналізу результатів приймається рішення про подальші кроки, такі як виправлення виявлених дефектів, внесення змін до програмного забезпечення або відправлення його на наступний етап розробки.

5. Підтримка та оптимізація. Цей етап передбачає моніторинг автоматизованих тестів для їхньої підтримки та оптимізації. Також на цьому етапі виправляються дефекти та збої в тестах для забезпечення стабільної роботи. Даний етап містить наступні кроки:

- Команда проводить постійний моніторинг виконання автоматизованих тестів. Це включає в себе відстеження прогресу виконання тестів, виявлення можливих проблем або збоїв та реагування на них негайно.

- Отримані результати тестування аналізуються з метою виявлення тенденцій, або повторюваних проблем. Цей аналіз допомагає виявити слабкі місця в програмному забезпеченні та покращити якість тестів.

- На основі аналізу результатів може бути виявлено можливості для оптимізації тестових скриптів. Це може включати в себе видалення зайвих кроків, оптимізацію швидкості виконання тестів, вдосконалення структури коду та інші аспекти.

- Якщо під час моніторингу виявляються дефекти або збої в автоматизованих тестах, команда вносить відповідні виправлення для виправлення проблем та забезпечення стабільності тестового середовища.

- При необхідності тестові скрипти оновлюються для врахування змін в програмному забезпеченні. Це може бути необхідно після внесення змін до функціоналу, інтерфейсу або архітектури програмного продукту.

- Всі внесені зміни та оптимізації документуються для забезпечення належної прозорості та доступності інформації для всіх учасників проєкту.

- Тестування проводиться на регулярній основі для перевірки стабільності тестового середовища та ефективності автоматизованих тестів у нових умовах.

В процесі впровадження автоматизованого тестування на проєкті важливо керуватися чіткою інструкцією та виконувати всі кроки для отримання бажаного результату. Впровадження автоматизованого тестування на проєкті вимагає систематичного та дисциплінованого підходу на кожному етапі для забезпечення високої якості програмного продукту. Кожен новий член тестової команди повинен бути ознайомлений з вимогами та всіма тестовими документами перед початком своєї роботи.

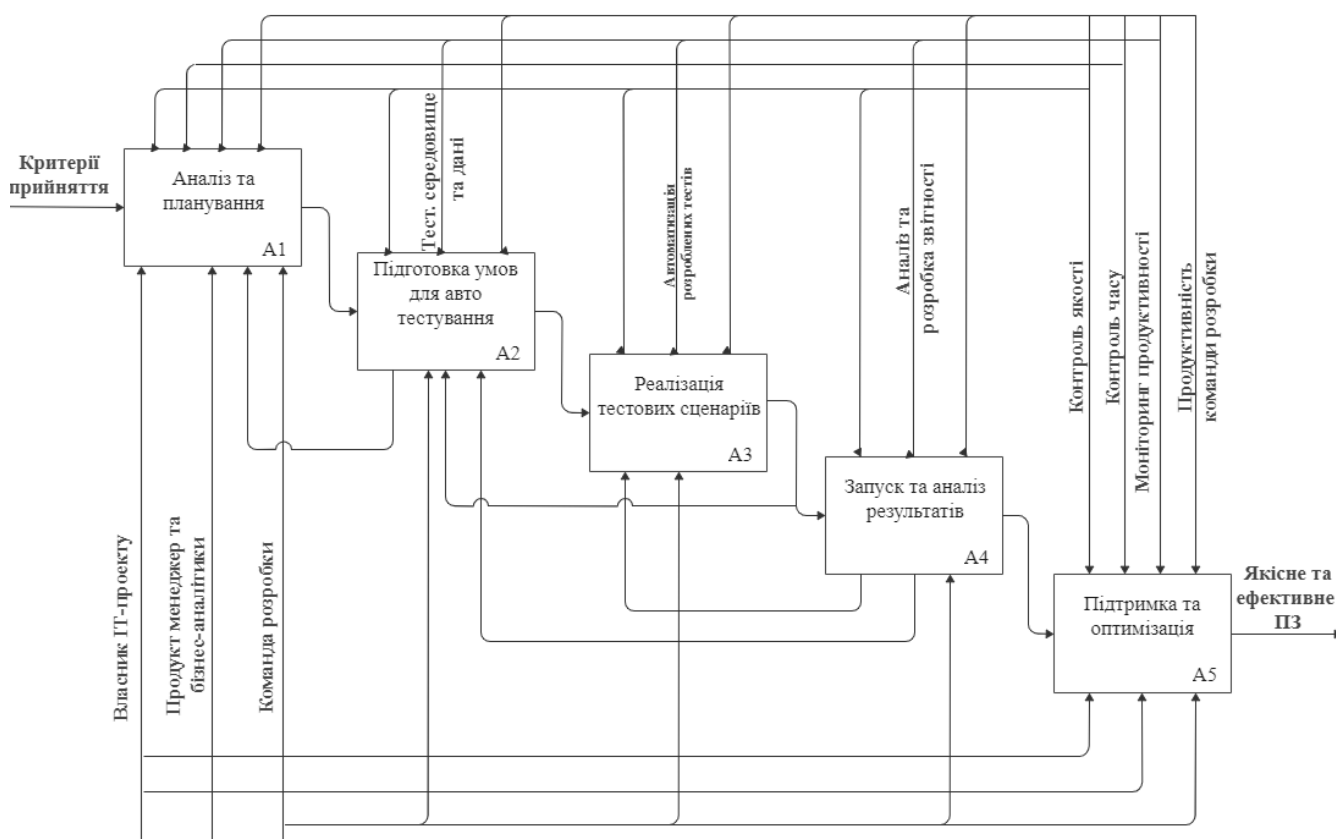


Рисунок 2.8 – Декомпозиція бізнес-процесу «Впровадження автоматизованого тестування»

Крім того, важливо регулярно оновлювати та вдосконалювати автоматизовані тести відповідно до змін у програмному забезпеченні. Це дозволить уникнути помилок і знизити ризики, пов'язані з введенням нових функцій або модифікацій. Впровадження зворотного зв'язку від команди розробників та користувачів також може сприяти підвищенню ефективності автоматизованого

тестування, що в кінцевому підсумку призведе до створення більш стабільного та надійного продукту.

2.4 Висновок до розділу

У другому розділі дипломної роботи був здійснений системний аналіз обраного ІТ-проєкту, для якого планується впровадження автоматизованого тестування. Проведений аналіз включав виділення головних особливостей системи, її функцій та критичних зон, що дозволило зрозуміти специфіку проєкту та його основні вимоги до тестування.

Крім того, в рамках дослідження було розглянуто типи тестової документації, які необхідно підготувати для успішного впровадження автоматизованого тестування.

Надалі, розглянуто та проаналізовано процес та етапи впровадження автоматизованого тестування для обраного ІТ-проєкту. Це включало в себе аналіз потреб проєкту, вибір необхідних інструментів та технологій, а також розробку та виконання плану дій для ефективного впровадження тестування. Кожен етап був детально розглянутий з метою забезпечення успішного та ефективного впровадження автоматизованого тестування на проєкті.

РОЗДІЛ 3.

ПРОЄКТНІ РІШЕННЯ ЩОДО ВПРОВАДЖЕННЯ СИСТЕМИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ НА ПІДПРИЄМСТВІ

3.1 Розробка тестового фреймворку та написання автоматизованих тестів

Після проведення аналізу вимог та розробки тестової документації, ми можемо впевнено приступити до планування та розробки тестового фреймворку для автоматизованого тестування.

Тестовий фреймворк для автоматизованого тестування – це набір інструментів, бібліотек, правил та керівних принципів, які допомагають автоматизувати процес тестування програмного забезпечення. Його мета – забезпечити ефективне, систематичне і повторюване тестування, зменшити людські помилки, прискорити процес розробки та забезпечити високу якість продукту.

Для створення тестового фреймворку за основу було взято середовище .Net, мову програмування c# та набір наступних бібліотек:

- **NUnit** – це популярний фреймворк для тестування, який використовується для створення автоматизованих тестів у .NET середовищі. Він дозволяє розробникам писати та виконувати тести для .NET додатків, забезпечуючи високу якість коду і знижуючи ризик появи помилок.

- **SpecFlow** – це інструмент для автоматизованого тестування, що дозволяє використовувати підхід BDD (Behavior Driven Development) у .NET середовищі. Він інтегрує принципи BDD, надаючи можливість описувати поведінку додатка зрозумілою мовою для бізнес-аналітиків, тестувальників та розробників. SpecFlow базується на мові Gherkin, яка використовується для написання тестових сценаріїв у зрозумілому форматі.

- **Selenium** – це популярний інструмент з відкритим вихідним кодом для автоматизованого тестування веб-додатків. Він дозволяє тестувальникам і розробникам автоматизувати взаємодію з веб-браузерами для перевірки функціональності, продуктивності та поведінки веб-додатків.

- **RestSharp** — це популярна бібліотека для спрощення взаємодії з RESTful сервісами в .NET. Вона надає зручний інтерфейс для виконання HTTP-запитів і обробки HTTP-відповідей, що робить її ідеальним вибором для розробки клієнтів API і автоматизованого тестування API.

- **FluentAssertions** - це бібліотека для написання тестів на мові програмування C#, яка забезпечує зручний і читабельний синтаксис для перевірки стану об'єктів у тестах. У випадку невдачі тесту, Fluent Assertions надає детальні та інформативні повідомлення про помилки, що полегшує діагностику проблем. Бібліотека підтримує перевірку різноманітних умов і типів даних, включаючи об'єкти, колекції, строки, числа та виключення.

- **Allure** – це відкрите програмне забезпечення для генерації звітів про результати автоматизованого тестування. Цей інструмент призначений для поліпшення візуалізації результатів тестування та створення зрозумілих та інформативних звітів, які легко читати та аналізувати.

Після того, як було проаналізовано та обрано необхідні інструменти розробки, ми можемо впевнено приступити до створення тестового фреймворку для написання автоматизованих скриптів. Даний етап дуже важливий, адже тестовий фреймворк – це фундамент майбутнього процесу автоматизованого тестування. Потрібно дуже уважно та прискіпливо розробляти фреймворк, адже навіть маленька помилка буде коштувати багато часу та коштів.

Якісно розроблений тестовий фреймворк повинен відповідати таким вимогам:

- **Простота використання:** тестовий фреймворк повинен мати простий та зрозумілий синтаксис, що дозволяє швидко розробляти тести, виявляти та виправляти помилки в них.

- **Масштабованість:** тестовий фреймворк повинен бути оптимізованим для швидкого виконання великої кількості тестів та мати можливість працювати з великими обсягами даних та ресурсів без втрат продуктивності.
- **Гнучкість:** тестовий фреймворк повинен дозволяти налаштовувати різні аспекти тестування, такі як умови передумов, параметри тестів, тощо. Також повинен мати можливість написання різних видів тестів з використанням різних методик тестування.
- **Підтримка різних типів тестів:** тестовий фреймворк повинен підтримувати можливість розробляти та запускати тести для окремих модулів або компонентів програмного забезпечення.
- **Підтримка різних платформ:** тестовий фреймворк повинен бути здатним працювати на різних платформах, таких як веб, мобільні пристрої, десктопні системи, сервери, тощо.
- **Надійність:** тестовий фреймворк повинен надійно працювати під час змін у коді програмного забезпечення та тестових сценаріях.
- **Підтримка паралельного виконання:** тестовий фреймворк повинен підтримувати паралельне виконання тестів на різних пристроях, браузерах або середовищах, щоб прискорити час виконання тестів.
- **Підтримка автоматизованої звітності:** тестовий фреймворк повинен забезпечувати можливість автоматичної генерації звітів про результати тестування, включаючи детальну інформацію про пройдені та невдалі тести.

Спираючись на перераховані вимоги, ми можемо приступити до розробки тестового проєкту. При розробці тестового фреймворку для написання автоматизованих тестів часто використовують багат шарову архітектуру, що забезпечує розділення відповідальності, покращує масштабованість і підтримку. Фреймворк буде складатися з трьох основних шарів: Core, Business та Test. Кожен з цих шарів виконує свою унікальну роль та функцію.

Давайте більш детально розглянемо кожен шар нашого тестового фреймворку:

- **Core Layer (Ядро)** – це основний шар, який відповідає за базову інфраструктуру фреймворку. Це включає в себе загальні утиліти, бібліотеки, налаштування середовища, управління конфігураціями, логування, обробку виключень та інтеграцію з іншими системами.
- **Business Layer (Бізнес логіка)** – це шар, що містить реалізацію бізнес-логіки та обробку даних, необхідних для виконання тестів. Тут розміщуються сервіси та модулі, що відповідають за взаємодію з основними бізнес-компонентами системи.
- **Test Layer (Тестування)** – це верхній шар, де безпосередньо пишуться тестові сценарії, що використовують компоненти з Business та Core шарів. Тут розташовуються конкретні тести, що перевіряють різні аспекти системи.

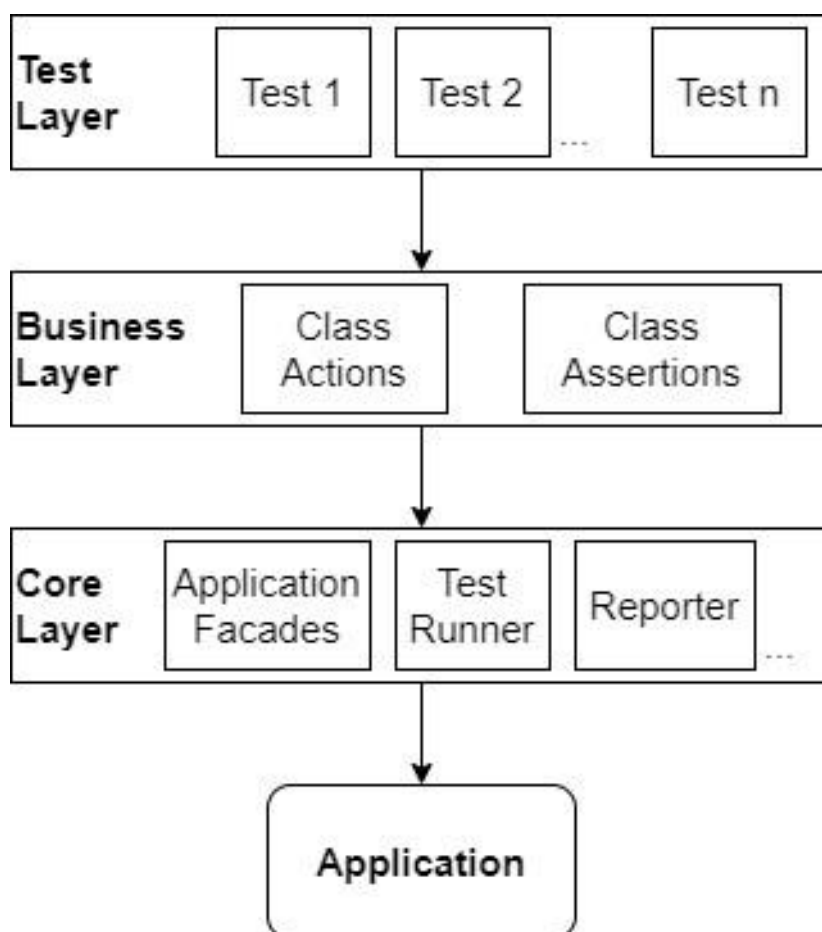


Рисунок 3.1 – Багаторівнева архітектура тестового фреймворку

Розібравшись з усіма вимогами та визначившись з архітектурою тестового фреймворку, ми можемо приступити до реалізації всіх наших думок та ідей у

вигляді коду. Тестовий фреймворк буде складатися з чотирьох проєктів, кожен проєкт буде відповідати за конкретний функціонал. Давайте більш детально розглянемо кожен із проєктів:

- **Core Project** – це фундаментальна складова, що забезпечує базову інфраструктуру та основні функціональні можливості, які використовуються іншими частинами фреймворку. Цей шар містить спільні утиліти, бібліотеки та сервіси, необхідні для створення, конфігурації та виконання тестів. Core забезпечує основну інфраструктуру для всього фреймворку, включаючи управління конфігураціями, логування та обробку виключень. Також Core абстрагує низькорівневі деталі реалізації, такі як взаємодія з веб-драйверами, базами даних та іншими зовнішніми системами, забезпечуючи прості інтерфейси для бізнес- та тестового шарів.

- **API Project** – це проєкт, що відіграє важливу роль у взаємодії з програмними інтерфейсами додатків, які тестуються. Цей шар забезпечує абстракцію від деталей реалізації HTTP-запитів, а також надає зручний та уніфікований спосіб для написання та виконання API-тестів. Проєкт полегшує написання тестових сценаріїв для перевірки бізнес-логіки, що реалізована через API, забезпечуючи швидкий зворотний зв'язок щодо коректності роботи API.

- **UI Project** – критичний компонент, що забезпечує засоби для тестування інтерфейсів користувача. Цей шар фреймворку відповідає за автоматизацію взаємодії з графічними інтерфейсами, що включає роботу з веб-додатками, мобільними додатками та іншими типами інтерфейсів. Даний проєкт містить в собі опис сторінок, елементів сторінок та методи для взаємодії з ними.

- **Test Project** – даний проєкт виконує ключову роль, оскільки саме тут безпосередньо реалізуються тестові сценарії, що перевіряють функціональність програмного забезпечення. Цей шар фреймворку відповідає за організацію тестових випадків, їх виконання, верифікацію результатів та генерацію звітів про тестування. Проєкт Test використовує компоненти з інших шарів фреймворку, таких як Core, API та UI, для здійснення всіх необхідних дій під час тестування.

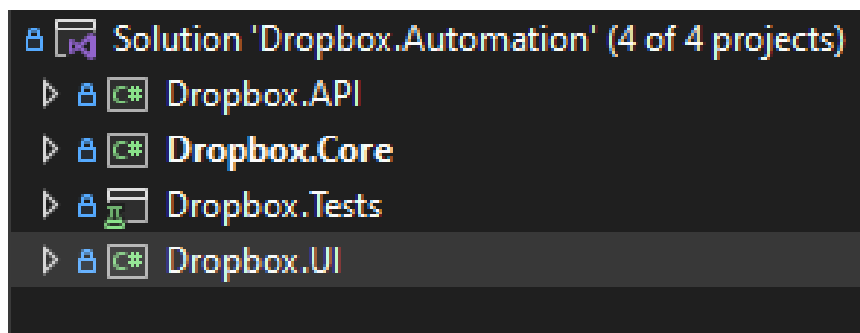


Рисунок 3.2 – Структура тестового фреймворку

Після створення тестових проєктів автоматизованого тестового фреймворку, ми можемо приступити до написання коду класів необхідних для написання та виконання тестових скриптів.

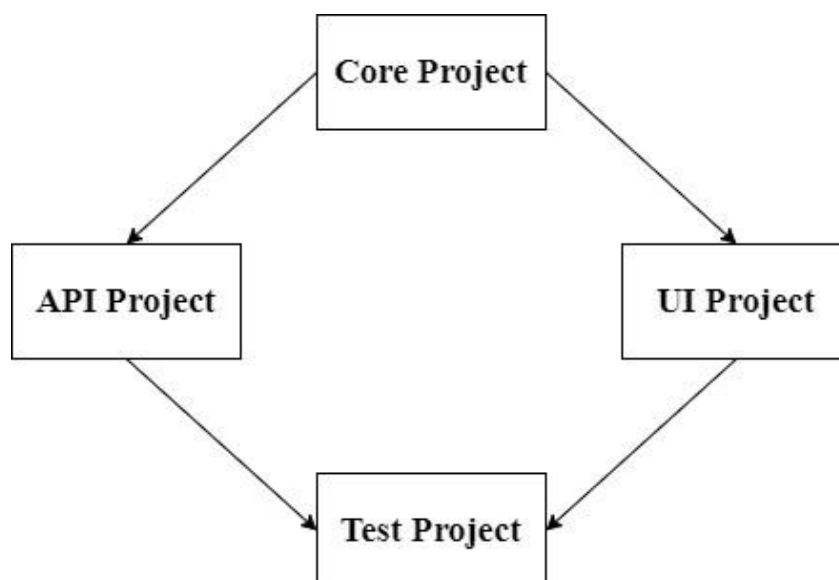


Рисунок 3.3 – Ієрархія тестових проєктів

Давайте більш детально розглянемо кожен із проєктів. Почнемо із Core проєкту, адже це найнижча ланка у ієрархії. Проєкт Core є основою, яка забезпечує стандартизовані та повторно використовувані компоненти для всього тестового фреймворку, що сприяє зменшенню дублювання коду та підвищенню ефективності розробки та підтримки тестів. Опишімо основні функції, що виконує даний проєкт:

- **Зберігання та організація констант (const):** Core проєкт містить класи, що описують значення типу const. Константи використовуються для зберігання значень, які залишаються незмінними протягом усього виконання програми, таких як математичні константи, константи конфігурації або інші значення, які не змінюються.

- **Зберігання та організація перелічувань (enum):** Core проєкт містить перелічування (enum). Enum – використовується для визначення набору іменованих констант, які називаються переліченнями. Це дозволяє групувати пов'язані значення під одним іменем, надаючи їм зрозумілі назви, що робить код більш читабельним та зрозумілим.

- **Зберігання та організація допоміжних класів (Extensions та Helpers):** допоміжні класи Extensions та Helpers створені для того, щоб спростувати роботу з великими частинами коду, реалізація методів відбувається один раз і далі ці методи повторно використовуються в інших частинах тестового фреймворку. У тестовому фреймворку для написання автоматизованих тестів класи Extensions та Helpers відіграють важливу роль у спрощенні коду тестів, підвищенні його читабельності та забезпеченні повторного використання коду.

- **Зберігання та організація класів утилітів (Utils):** класи Utils у тестовому фреймворку для написання автоматизованих тестів містять набір загальних, універсальних і часто використовуваних методів або функцій, які можуть бути використані в різних частинах тестового фреймворку. Вони подібні до класів Helpers, але можуть охоплювати більш широкий спектр функцій. У моєму випадку, це класи для роботи з кольорами та стилями шрифтів.

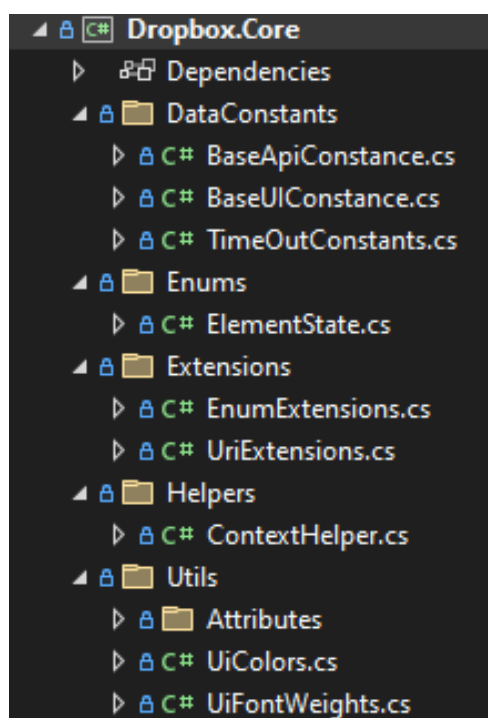


Рисунок 3.4 – Структура Core проєкту тестового фреймворку

Далі в ієрархії проєктів тестового фреймворку знаходять API та UI проєкти. Для початку детально розглянемо API проєкт. Даний проєкт забезпечує інтерфейси та методи для взаємодії з API (Application Programming Interface) тестованого додатка. Він дозволяє здійснювати HTTP-запити (GET, POST, PUT, DELETE тощо), обробляти відповіді, перевіряти статус-коди, заголовки та тіла відповідей. Розглянемо основний функціонал даного проєкту:

- **Опис та організація моделей запитів та відповідей (Models):** моделі в тестовому фреймворку використовуються для представлення даних, які передаються між клієнтом та сервером через API. Ці моделі допомагають стандартизувати дані, які передаються між різними компонентами системи, забезпечуючи чіткість та консистентність у взаємодії. Вони дозволяють визначити структуру даних, які використовуються в API запитах та відповідях, що спрощує розробку тестів та забезпечує більшу стабільність у випадку змін у структурі даних. Використання моделей також сприяє читабельності та підтримці коду, оскільки вони відображають структуру даних, яка використовується у взаємодії з API.

- **Опис реалізації класу для побудови тестових запитів та обробки відповідей від серверу (RequestBuilder):** Клас RequestBuilder в тестовому фреймворку призначений для конструювання HTTP-запитів до API тестованого додатка. Він дозволяє зручно створювати запити з необхідними параметрами, заголовками та тілом запиту, що спрощує процес написання тестів. API RequestBuilder допомагає уникнути дублювання коду та забезпечує чітку структуру тестів, що полегшує їхнє розуміння та підтримку. Використання цього класу також сприяє покращенню читабельності та підтримки коду, оскільки він абстрагує деталі конструювання запитів, дозволяючи зосередитися на логіці тестування функціональності API.

- **Опис класів для реалізації тестових запитів (ApiFacades):** у даних класах відбувається виклик класу RequestBuilder для створення та реалізація тестових запитів і обробки відповідей від сервера. Для кожного функціоналу створюється окремий клас.

- **Опис та реалізація допоміжних класів для роботи з API (Extensions):** Класи Extensions в тестовому фреймворку призначені для розширення можливостей взаємодії з API та полегшення написання автоматизованих тестів. Вони містять набір методів розширення, які додають додаткову функціональність до основних класів API, таких як класи для роботи з HTTP-запитами (наприклад, класи для виконання GET, POST, PUT, DELETE запитів). Ці методи розширення можуть включати додаткові перевірки, обробку відповідей, створення кастомних запитів тощо.

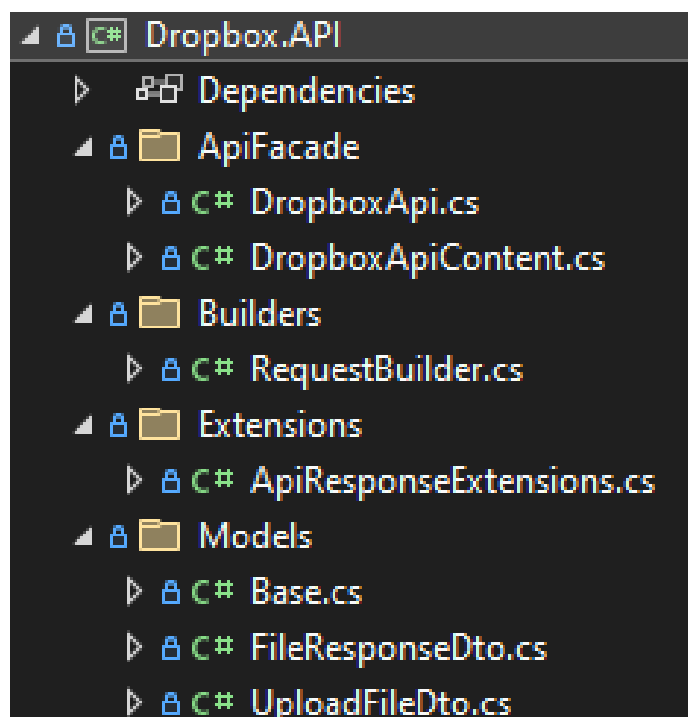


Рисунок 3.5 – Структура API проекту тестового фреймворку

Далі приступимо до розгляду UI проекту. Даний проєкт знаходить на одному рівні ієрархії з API проєктом, розроблені класи даних проєктів будуть використані в тестовому проєкті для реалізації тестових автоматизованих скриптів. Розглянемо основні функції даного проєкту:

- **Опис та реалізація класів для запуску UI тестів (DriverProvider та DriverHelper):** класи DriverProvider та DriverHelper в тестовому фреймворку призначені для управління та надання доступу до екземплярів веб-драйверів (Selenium WebDriver) для автоматизованих тестів, що перевіряють веб-інтерфейс системи. Основна мета цих класів – забезпечити централізоване створення,

ініціалізацію та управління драйверами, а також надати додатковий функціонал для роботи з ними.

- **Опис та реалізація сторінок системи (PageObjects):** класи PageObjects у тестовому фреймворку використовуються для реалізації паттерну проєктування Page Object, що спрощує управління тестами та забезпечує більшу стабільність автоматизованих тестів. Вони дозволяють абстрагувати веб-сторінки або їх функціональні частини, уникнути повторюваного коду, зменшити залежність тестів від конкретної реалізації інтерфейсу користувача, підтримувати тестування веб-сторінок та зменшити залежність від локаторів елементів. В результаті ці класи сприяють створенню структурованих, підтримуваних та менш вразливих до змін тестів, що забезпечує більшу ефективність та надійність у процесі розробки та підтримки автоматизованих тестів.

- **Опис та реалізація компонентів, що будуть використані при описі сторінок веб-додатку (Components):** класи Components в тестовому фреймворку призначені для організації та управління компонентами або елементами інтерфейсу користувача, які використовуються у тестах. Ці класи допомагають створювати абстракції для різних елементів веб-сторінок або додатків, таких як кнопки, поля вводу, таблиці тощо, що спрощує роботу з ними у тестах і забезпечує більшу стабільність автоматизованих тестів.

- **Опис та реалізація допоміжних класів для роботи з UI (Extensions):** класи Extensions в тестовому фреймворку призначені для розширення функціональності взаємодії з елементами користувацького інтерфейсу (UI) в автоматизованих тестах. Ці класи містять набір методів розширення, які додають додаткові можливості для роботи з UI елементами, такими як кнопки, поля вводу, таблиці тощо. Основна мета класів – зробити код тестів більш зрозумілим, підтримуваним та менш залежним від конкретної реалізації інтерфейсу користувача.

- **Опис та реалізація класу для ініціалізації сторінок (EntryPageFactory):** клас EntryPageFactory у тестовому фреймворку використовується для створення та ініціалізації об'єктів сторінок (Page Objects) на

початковому етапі виконання автоматизованих тестів. Основна мета даного класу – забезпечити централізований та стандартизований спосіб створення екземплярів сторінок, що спрощує роботу з ними у тестах та підтримує стабільність тестового коду.

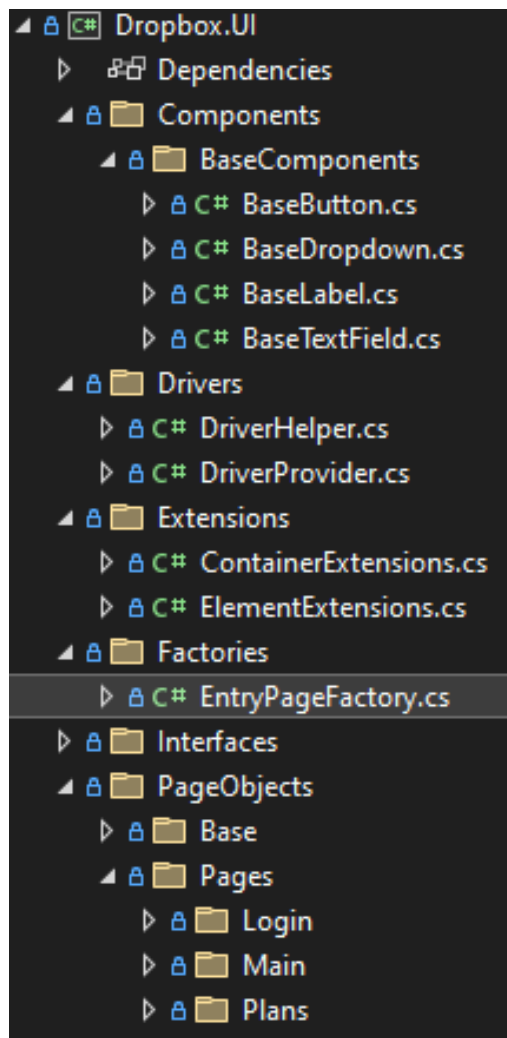


Рисунок 3.6 – Структура UI проєкту тестового фреймворку

Після того, як був здійснений опис всіх допоміжних проєктів, можна приступити до опису головного класу у ієрархії тестового фреймворку, а саме тестового проєкту (Test проєкт). Test проєкт у тестовому фреймворку використовується для написання, організації та виконання наборів тестів для перевірки різних аспектів програмного забезпечення. Основна мета тестового проєкту – забезпечити створення та виконання автоматизованих тестів, які перевіряють функціональність, надійність та відповідність вимогам програмного забезпечення. Давайте більш детально розглянемо основні функції дано проєкту:

- **Опис та реалізація тестових наборів:** тести – це основа для забезпечення якості та стабільності програмного забезпечення. Вони дозволяють автоматично перевіряти різні аспекти додатка, виявляти помилки та недоліки на ранніх етапах розробки, а також гарантувати, що нові зміни не порушують існуючу функціональність.

- **Опис та реалізація класів з тестовими кроками (Steps), що використовуються у тестах:** дані класи містять методи, що використовуються для реалізації сценаріїв тестування. Класи Steps забезпечують зв'язок між текстовими сценаріями, написаними на природній мові, та програмним кодом, що виконує ці сценарії. Це дозволяє тестувальникам і розробникам працювати спільно, використовуючи зрозумілі всім сценарії. Методи в класах Steps можуть бути повторно використані в різних сценаріях, що дозволяє зменшити дублювання коду та підвищити рівень підтримки.

- **Опис та реалізація класів функціонального шару (Contexts):** класи Contexts у тестовому проєкті тестового фреймворку відіграють важливу роль у забезпеченні збереження стану та спільного використання даних між різними кроками (Steps) під час виконання тестових сценаріїв. Вони служать як засіб для організації та управління контекстом тестів, дозволяючи ефективно взаємодіяти з методами з UI та API проєктів. Вони забезпечують доступ до спільних даних та об'єктів між різними кроками, що полегшує повторне використання інформації, наприклад, результатів API викликів, збережених даних користувача тощо. Контексти дозволяють ізолювати логіку взаємодії з UI та API у відповідних проєктах, роблячи тестовий код чистішим і менш заплутаним. Це дозволяє крокам тестів фокусуватися на сценаріях та бізнес-логіці, а не на деталях реалізації.

- **Опис та реалізація допоміжних класів для роботи з тестами (Extensions):** класи Extensions в тестовому проєкті використовуються для додавання додаткової функціональності до існуючих класів, це можуть бути допоміжні класи при роботі з файлами різних типів та при потребі робити якісь специфічні перевірки.

- **Опис та реалізація класу для налаштування запуску тестів (Hooks):**

клас Hooks в тестовому проєкті тестового фреймворку використовується для виконання спеціальних дій перед та після виконання тестових сценаріїв або окремих кроків. Ці дії включають налаштування середовища тестування, підготовку даних, очищення після тестування та інші допоміжні операції, які необхідні для правильного виконання тестів. Клас Hooks може використовуватися для ведення логів або створення звітів про виконання тестів, що може включати запис результатів, знімки екрану при помилках, збір інформації про виконання тестів.

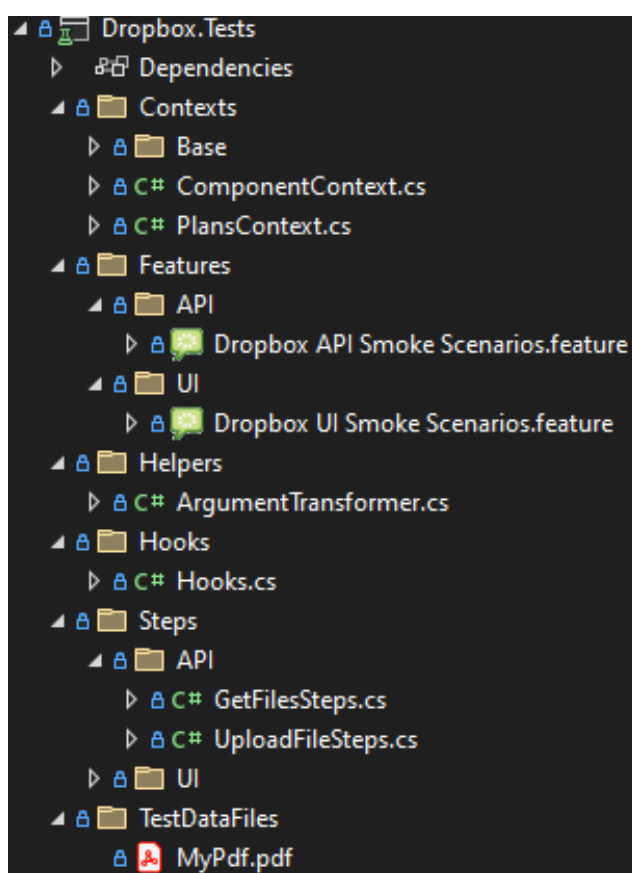


Рисунок 3.7 – Структура API проєкту тестового фреймворку

Після створення всієї необхідної інфраструктури можна приступити до написання автоматизованих тестових скриптів. Для цього скористаємося інструментом «SpecFlow». SpecFlow – це інструмент для поведінкового тестування (BDD - Behavior-Driven Development) у середовищі .NET. Він дозволяє розробникам, тестувальникам і бізнес-аналітикам описувати поведінку програмного забезпечення мовою, зрозумілою для всіх учасників проєкту. SpecFlow

базується на Gherkin – мові для написання сценаріїв тестування у форматі, схожому на природну мову.

Для початку створимо файлів з розширенням .feature та використавши мову Gherkin реалізуємо наші тестові скрипти. Після цього додаймо необхідні методи в класах Speps та Contexts.

```
@UI
Feature: Dropbox UI Smoke Scenarios

Dropbox UI Smoke Scenarios

@Smoke
Scenario Outline: User can't login/sign up with wrong Email
  Given I navigate to the Dropbox Main page
  When I click on '<Button Name>' control at the 'Header Section' container on the Main page
  And I wait for the page to be loaded
  When I set 'Test' value to 'Email' control at the 'Login Section' container on the Login page
  And I click on 'Continue' control at the 'Login Section' container on the Login page
  Then 'Error Message' control at the 'Login Section' container is 'Displayed' on the Login page
  And 'Error Message' control at the 'Login Section' container has value on the Login page
  ...
  An email address must contain a single @
  ...

Examples:
  | Button Name |
  | Login      |
  | Sign up    |

@Smoke
Scenario Outline: The correct Data is displayed for Billed Yearly test plans
  Given I navigate to the Dropbox Main page
  When I click on 'Get Started' control at the 'Header Section' container on the Main page
  And I wait for the page to be loaded
  Then The Plan with '<Name>' name has following data:
    | Type | Price | Users | Storage | Transfer Files | Restore Days |
    | <Type> | <Price> | <Users> | <Storage> | <Transfer Files> | <Restore Days> |

Examples:
  | Name | Type | Price | Users | Storage | Transfer Files | Restore Days |
  | Plus | For personal use | $9.99 / month | 1 user | 2 TB of storage | Transfer files up to 50 GB | 30 days to restore deleted files |
  | Essentials | For professionals | $16.58 / month | 1 user | 3 TB of storage | Transfer files up to 100 GB | 180 days to restore deleted files |
  | Business | For teams | $15 / user / month | 3+ users | Starts at 9 TB for the team | Transfer files up to 100 GB | 180 days to restore deleted files |
  | Business Plus | For companies | $24 / user / month | 3+ users | Starts at 15 TB for the team | Transfer files up to 250 GB | 1 year to restore deleted files |
```

Рисунок 3.8 – Реалізація UI тестів за допомогою SpecFlow

```
@API
Feature: Dropbox API Smoke Scenarios

Dropbox API Smoke Scenarios

@Smoke @Get
Scenario: Get all files
  When I try to get list of all existing files
  Then I should get valid list of files

@Smoke @Upload
Scenario: Upload a file
  Given I have 'MyPdf.pdf' file to upload
  When I upload the file
    | Path | Mode | AutoRename | Mute |
    | /MyNewFolder/MyFile.pdf | add | true | false |
  Then I should be able to get file info
    | Name |
    | MyFile.pdf |
```

Рисунок 3.9 – Реалізація API тестів за допомогою SpecFlow

Після написання тестових скриптів здійснимо запуск, у моєму випадку я користуюся Visual Studio для роботи з мовою програмування с# та платформою .Net, тому пуск тестів я буду виконувати через консоль «Test Explorer». Test Explorer – це інструмент, який використовується для запуску, перегляду та управління тестами у проєкті. Test Explorer надає зручний інтерфейс для розробників, щоб вони могли ефективно управляти процесом тестування.

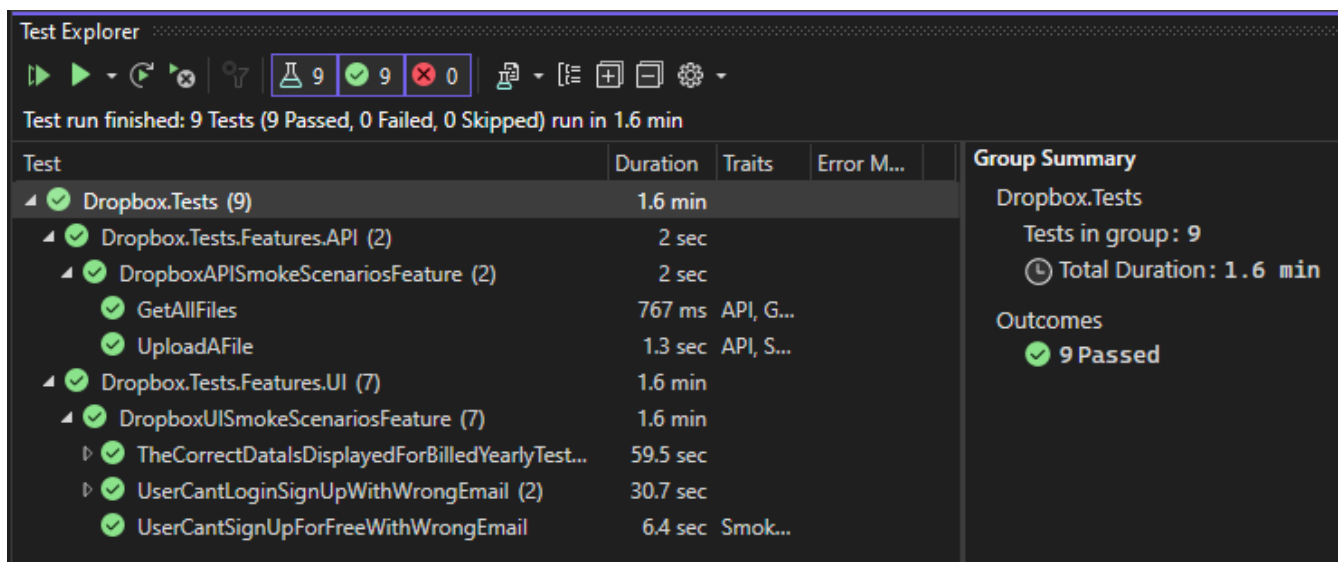


Рисунок 3.10 – Запуск автоматизованих тестів за допомогою консолі «Test Explorer»

Після запуску тестів був отриманий позитивний результат – всі тести успішно пройшли. Далі скористаємося бібліотекою «Allure» для створення звіту, який буде відображати детальну інформацію, яка була отримана під час запуску тестів.

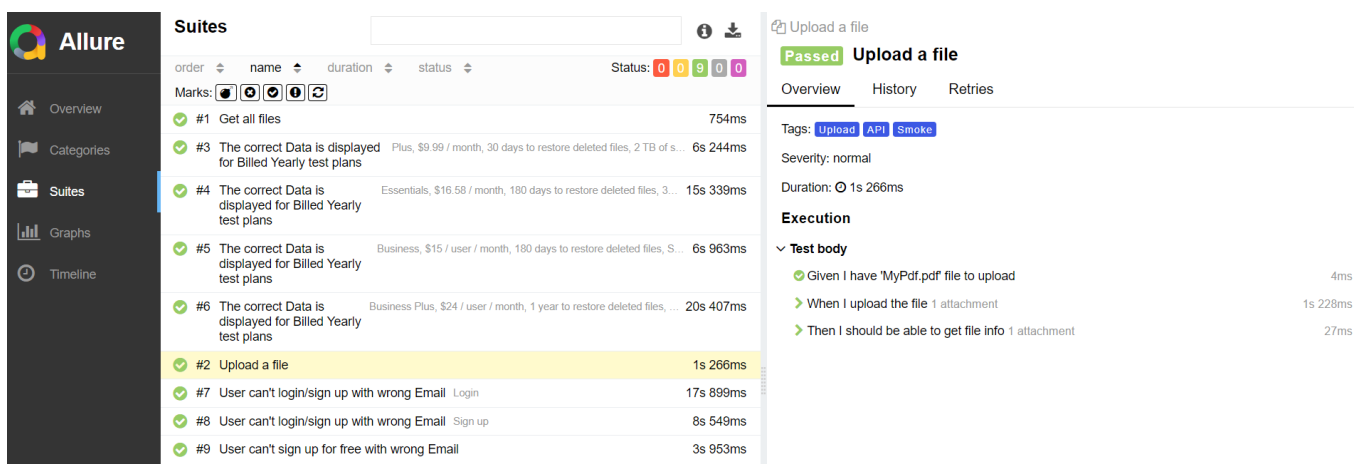


Рисунок 3.11 – Тестовий звіт створений за допомогою інструменту «Allure»

Спираючись на даний звіт, ми можемо дійти до висновку, що запуск тестів пройшов успішно – дефектів у роботі програмного забезпечення не було виявлено.

Звіти, створені за допомогою інструменту «Allure», надають різноманітну та докладну інформацію про результати виконання тестів. Ось ключова інформація, яку можна отримати з такого звіту:

- **Загальна статистика:** загальна кількість тестів, які було запущено, кількість успішних, невдалих та пропущених тестів.
- **Графіки та діаграми:** графіки, які відображають прогрес виконання тестів, динаміку успішних і невдалих тестів, середню тривалість виконання тощо.
- **Детальні результати:** для кожного тесту можна переглянути детальну інформацію, таку як стан тесту, повідомлення про помилки, скріншоти стану інтерфейсу, якщо це тести веб-додатків, параметри, з якими був запущений тест, тощо.
- **Історія виконання тестів:** історія результатів виконання тестів дозволяє відстежувати зміни якості програмного забезпечення та виявляти тенденції у виконанні тестів.
- **Категоризація тестів:** можливість фільтрувати тести за різними категоріями або мітками, що дозволяє швидко знаходити тести за певними критеріями.

Загалом, звіти Allure надають інформацію, яка допомагає зрозуміти, які тести були успішними, де виникли проблеми та які кроки потрібно вжити для виправлення помилок та покращення якості програмного забезпечення.

3.2 Інтеграція автоматизованого тестування в CI/CD процес

Після завершення розробки автоматизованого тестового фреймворку та написання автоматизованих тестів, приступимо до впровадження процесу автоматизованого тестування в цикли розробки, а саме в CI/CD процес.

CI/CD (Continuous Integration/Continuous Delivery) процес – це сучасний підхід до розробки програмного забезпечення, який автоматизує інтеграцію, тестування та розгортання коду. У контексті CI, зміни коду від різних розробників регулярно інтегруються в загальну кодову базу. Кожна зміна автоматично збирається, тестується і перевіряється на конфлікти з іншими змінами, що дозволяє швидко виявляти і виправляти помилки. Це забезпечує більш стабільну та якісну кодову базу, зменшуючи ризик інтеграційних конфліктів і полегшуючи командну роботу.

CD включає два етапи: Continuous Delivery і Continuous Deployment. У першому етапі, після успішного проходження всіх тестів на етапі CI, зміни коду автоматично підготовляються до розгортання на будь-яке середовище (тестове, проміжне або продуктивне). У другому етапі, безперервне розгортання автоматизує процес розгортання на продуктивне середовище, що дозволяє швидко і надійно впроваджувати нові функції без ручного втручання. Це мінімізує час виведення продукту на ринок і підвищує його якість за рахунок зменшення ризиків людських помилок.

Автоматизоване тестування відіграє ключову роль в CI/CD процесі, забезпечуючи якість і стабільність програмного забезпечення на всіх етапах розробки. У контексті Continuous Integration (CI), автоматизоване тестування дозволяє негайно перевіряти кожен змін коду, яку розробники інтегрують у загальну кодову базу. Це включає юніт-тести, які перевіряють окремі функції або модулі, та інтеграційні тести, що оцінюють взаємодію між різними компонентами системи. Завдяки цьому підходу помилки можуть бути виявлені на ранніх стадіях розробки, що значно спрощує їх виправлення і запобігає накопиченню технічного боргу.

На етапі Continuous Delivery (CD), автоматизоване тестування розширюється, включаючи функціональні та регресійні тести, які перевіряють систему в цілому і забезпечують, що нові зміни не порушують існуючий функціонал. Це дозволяє автоматично підготувати зміни для розгортання на різні середовища, такі як тестове або проміжне. Крім того, такі тести забезпечують

впевненість у тому, що програмне забезпечення відповідає вимогам якості та готове до розгортання на продуктивне середовище. Важливим аспектом є швидкість виконання цих тестів, оскільки тривале тестування може затримати доставку нових функцій користувачам.

На етапі Continuous Deployment (CD), автоматизоване тестування стає ще більш критичним, оскільки зміни коду автоматично розгортаються на продуктивне середовище без ручного втручання. Тут важливо мати надійний набір тестів, який охоплює всі аспекти системи, включаючи навантажувальні тести і тести безпеки. Це забезпечує, що кожна зміна не тільки функціонує правильно, але й відповідає вимогам щодо продуктивності та безпеки. Автоматизоване тестування в CI/CD процесі значно зменшує ризик випуску дефектного коду на продуктивне середовище, підвищує швидкість розробки і доставки, а також забезпечує високу якість кінцевого продукту.

Підготовка автоматизованого тестування в CI/CD процесі – це важливий етап, який передбачає визначення стратегії тестування, вибір підходящих інструментів та налаштування тестових середовищ. Враховуючи власний досвід, можу виділити основні етапи інтеграції автоматизованого тестування в CI/CD процес:

- **Налаштування CI/CD платформи:** на даному етапі відбувається вибір відповідного інструменту і його інтеграцію з вашим репозиторієм коду. Спочатку необхідно встановити або налаштувати платформу, потім підключити її до системи контролю версій, створити налаштування для автоматичного запуску тестів при змінах в коді.

- **Відстеження та аналіз результатів тестування:** даний етап передбачає використання інструментів для збору та аналізу даних про виконані тести з метою виявлення проблем та покращення якості коду. Після завершення кожного тестового етапу результати тестування автоматично зберігаються та аналізуються за допомогою таких інструментів, як Allure для створення детальних звітів про тестування. Це допомагає відстежувати успішність тестів, знаходити помилки та створювати звіти для розробників.

- **Налаштування нотифікацій:** даний крок полягає у створенні механізму для оперативного інформування команди розробників про результати виконання інструкції CI/CD. У конфігураційних файлах інструкції налаштовуються правила для відправки нотифікацій при успішному проходженні тестів, виявленні помилок або при завершенні кожного етапу інструкції. Такі сповіщення допомагають команді своєчасно реагувати на проблеми, підтримувати високу швидкість розробки та забезпечувати прозорість процесу інтеграції та розгортання.

- **Вдосконалення та оптимізація:** даний крок спрямований на регулярний аналіз та оптимізацію процесів CI/CD для підвищення ефективності та якості розробки програмного забезпечення. Цей процес включає ретельний перегляд результатів тестування, виявлення та усунення вузьких місць, оптимізацію тестових сценаріїв для зменшення часу їх виконання, а також додавання нових тестів для покриття нових функцій і виявлених слабких місць. Команда повинна збирати зворотний зв'язок від усіх учасників процесу, проводити регулярні ретроспективи та впроваджувати покращення на основі отриманих даних. Такий підхід забезпечує неперервне вдосконалення, адаптацію до нових вимог і підвищення загальної продуктивності та якості продукту.

Спираючись на дану інформацію перейдемо до впровадження автоматизованого тестування в цикли розробки, для цього використаємо популярний та багатофункціональний інструмент «TeamCity».

TeamCity – це інтегрована платформа для безперервної інтеграції (CI) та безперервної доставки (CD), створена компанією JetBrains. Цей інструмент надає розробникам можливість автоматизувати процеси збирання, тестування та розгортання програмного забезпечення. Одна з ключових особливостей TeamCity – це гнучкість конфігурації, що дозволяє налаштовувати складні CI/CD інструкції, враховуючи особливості кожного проєкту. Вона також підтримує широкий вибір інтеграцій з популярними системами контролю версій, засобами збірки та засобами тестування.

TeamCity може використовуватися як для малих проєктів, так і для великих підприємств, завдяки своїй масштабованості. Вона надає інструменти для

моніторингу стану CI/CD інструкцій, а також для аналізу результатів тестування та метрик якості коду. Завдяки можливості налаштування прав доступу, TeamCity забезпечує безпеку використання, дозволяючи керувати доступом до різних ресурсів та функцій системи. В цілому, TeamCity є потужним інструментом для автоматизації процесів розробки програмного забезпечення, що допомагає розробникам ефективно впроваджувати принципи CI/CD та підвищувати якість та швидкість розробки.

Приступимо для створення інструкцій для запуску тестів. Проаналізувавши вимоги для нашої системи, було розроблено наступні інструкції для запуску тестів:

- **Інструкція для запуску тестів після будь-якої зміни в коді головного репозиторія:** дана інструкція для запуску автоматизованих тестів передбачає автоматичне спостереження за головною гілкою у репозиторії та виконання певних дій при кожній зміні у коді. Першим кроком є збирання проєкту, щоб переконатися, що всі зміни інтегруються без помилок. Після цього запускаються автоматизовані тести для перевірки якості коду та функціональності. Цей процес дозволяє розробникам швидко виявляти помилки та проблеми, які можуть виникнути при змінах у вихідному коді, і забезпечує високу якість програмного забезпечення на кожному етапі розробки.

- **Інструкція для автоматичного запуску тестів вночі один раз на день:** даний інструкція для запуску автоматизованих тестів передбачає щоденне автоматичне оновлення та тестування проєкту з використанням найновіших версій вихідних файлів. Цей підхід корисний для перевірки стабільності коду та виявлення потенційних проблем, а також забезпечення того, що проєкт працює правильно в умовах найновіших змін. Після оновлення вихідних файлів інструкції автоматично збирає проєкт та запускає автоматизовані тести, що допомагає виявляти будь-які проблеми або помилки, що можуть виникнути в результаті останніх змін у коді. Цей підхід дозволяє підтримувати високу якість програмного забезпечення та впевненість у стабільності проєкту за рахунок щоденного автоматичного тестування з використанням найновіших змін у вихідному коді.

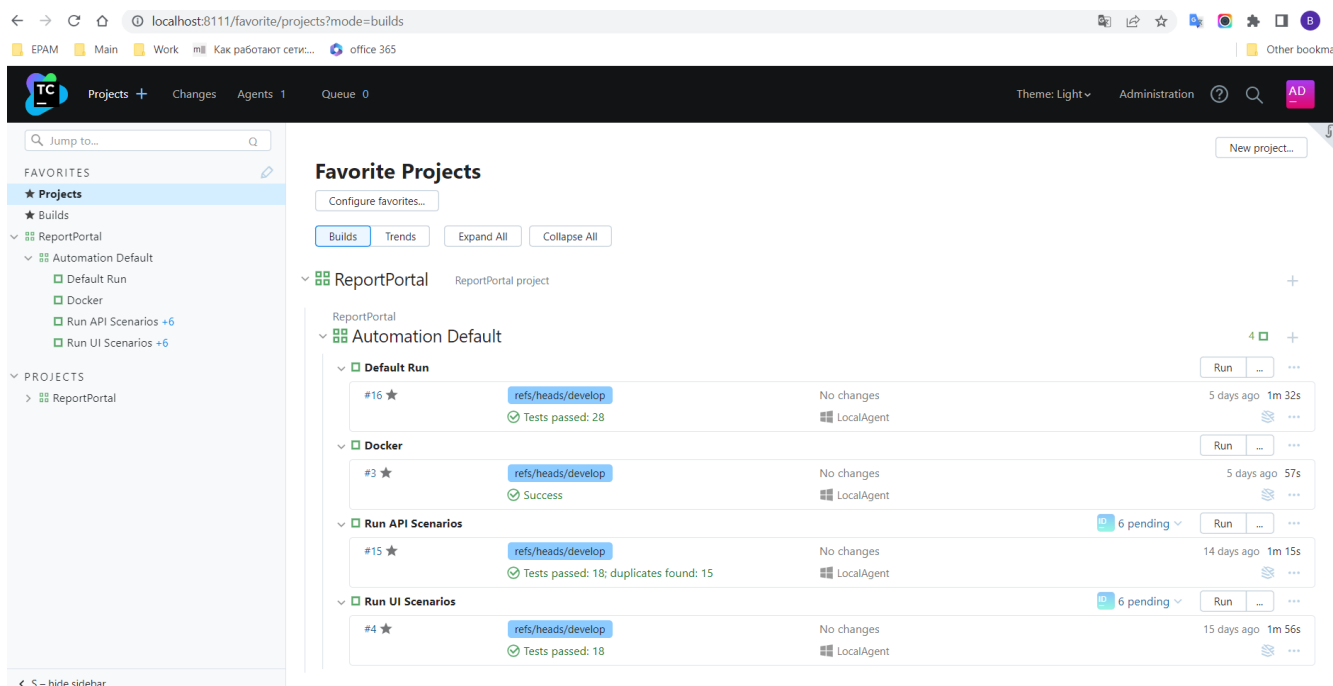


Рисунок 3.12 – Запуск тестів за допомогою додатку «TeamCity»

Також реалізуємо додаткові механізми контролю якості для нашого коду, вони дозволять автоматизовано перевіряти рівень якості програмного забезпечення на кожному етапі CI/CD інструкції. Основні механізми контролю якості включають в себе:

- **Компіляція без помилок:** даний механізм контролю якості перевіряє, чи вдалося успішно зібрати проект без будь-яких помилок компіляції. Якщо під час компіляції виникають помилки, інструкція може автоматично відмовити у виконанні наступних кроків та повідомити розробників про проблему.
- **Запуск тестів:** даний механізм контролю якості перевіряє, чи успішно пройшли всі автоматизовані тести. Якщо який-небудь тест не пройде успішно, інструкція може відхилити зміни та повідомити про це розробників.
- **Перевірка стандартів коду:** даний механізм контролю якості може перевіряти відповідність коду встановленим стандартам програмування, таким як форматування, найменування змінних, використання коментарів тощо. Невідповідність цим стандартам може вважатися причиною відхилення змін.
- **Перевірка покриття коду:** даний механізм контролю якості перевіряє покриття коду автоматизованими тестами. Якщо покриття коду тестами нижче

певного порогу, інструкція може відхилити зміни або відправити повідомлення про це розробникам.

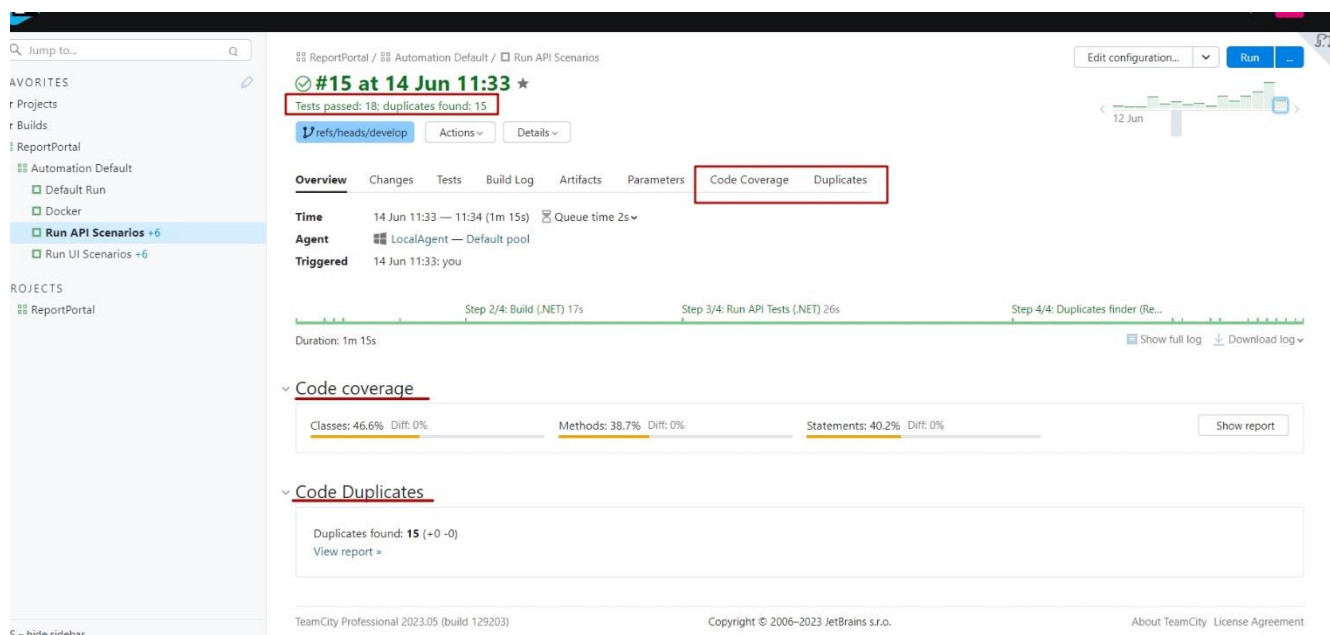


Рисунок 3.13 – Налаштування механізмів контролю якості в додатку «TeamCity»

Також налаштуємо нотифікації про стан CI/CD інструкції. Налаштування нотифікацій для CI/CD інструкцій є важливою складовою автоматизованого процесу розробки програмного забезпечення. Це дозволяє членам команди розробки отримувати сповіщення про стан та результати виконання інструкції в реальному часі. Причиною важливості цього є можливість швидко реагувати на будь-які проблеми або помилки, що виникають у процесі розробки. Наприклад, якщо інструкція виявляє помилку компіляції або невдачу автоматизованих тестів, учасники команди отримують сповіщення про це негайно, що дозволяє їм швидко реагувати та виправляти проблеми.

Крім того, налаштування нотифікацій сприяє поліпшенню комунікації в команді та підвищенню її ефективності. Це дозволяє всім учасникам проєкту бути в курсі останніх змін і результатів виконання інструкцій, навіть якщо вони працюють в різних часових поясах або мають різний графік роботи. Нотифікації також можуть включати в себе посилання на детальні звіти або журнали виконання, що дозволяє швидко знайти інформацію про проблеми та їх вирішення. У цілому, налаштування нотифікацій для CI/CD інструкцій дозволяє забезпечити оперативну

реакцію на проблеми та покращити співпрацю в команді, що веде до збільшення продуктивності та якості розробки.

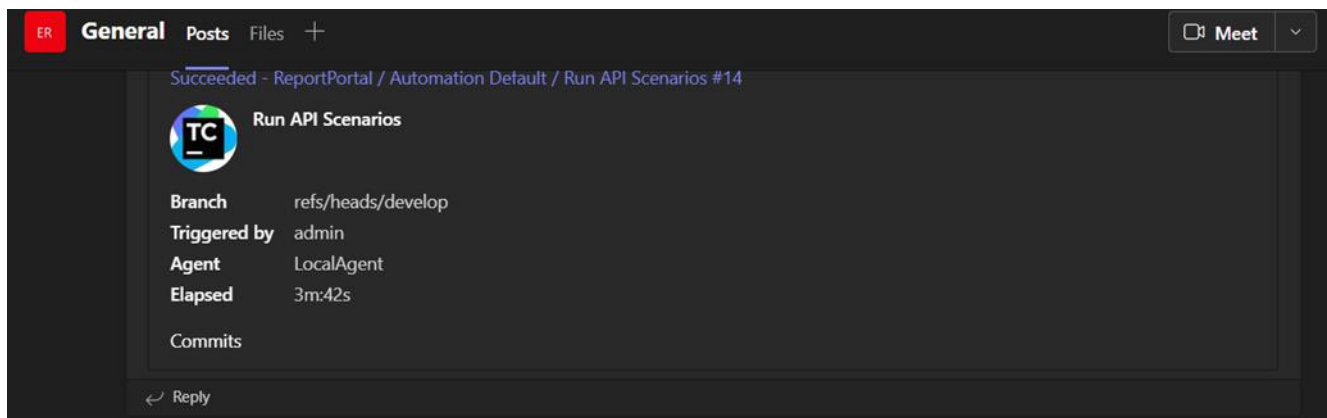


Рисунок 3.14 – Приклад нотифікації в месенджері «Teams» для CI/CD інструкції

Після здійснення всіх необхідних налаштувань, приступимо до запуску тестів та аналізу результатів у TeamCity. Аналіз результатів запуску автоматизованих тестів у TeamCity відіграє ключову роль у забезпеченні якості програмного забезпечення та виявленні можливих проблем. TeamCity надає різноманітні інструменти для аналізу результатів тестування, включаючи графіки покриття коду тестами, звіти про результати тестів, журнали виконання та інші.

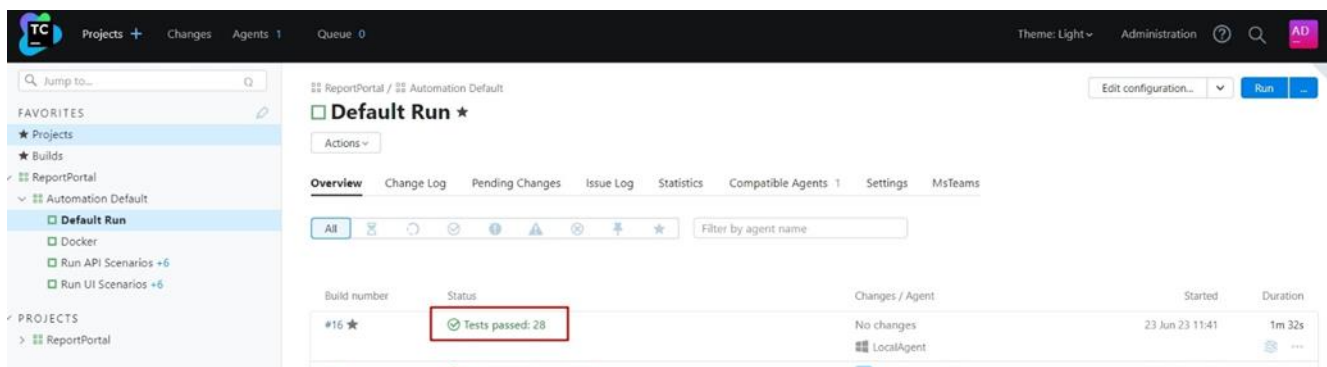


Рисунок 3.15 – Запуск тестів у додатку «TeamCity»

Один із важливих аспектів аналізу – це виявлення невдалого запуску тестів і відповідне повідомлення команди про це. Якщо автоматизовані тести не пройшли успішно, TeamCity може надіслати сповіщення про це учасникам команди, щоб вони могли вжити необхідних заходів для виправлення проблеми. Крім того, TeamCity забезпечує можливість глибокого аналізу результатів тестів, включаючи

детальні звіти про виявлені помилки та покриття коду тестами, що допомагає зорієнтуватися в проблемі та вжити відповідних заходів для її вирішення.

Загалом, аналіз результатів запуску автоматизованих тестів у TeamCity допомагає забезпечити високу якість програмного забезпечення, швидко реагувати на виявлені проблеми та забезпечувати ефективну комунікацію в команді розробників.

3.3 Аналіз отриманих результатів та оптимізація процесу тестування

У сучасному світі інформаційних технологій розробка ІТ-проектів є складним та відповідальним завданням, що передбачає високу ступінь комплексності та взаємодії різноманітних компонентів. Швидкість впровадження автоматизованого тестування не є вирішальним критерієм, адже головне досягнути максимального покриття та якості розроблених автоматизованих скриптів для отримання максимальної якості та ефективності розроблюваного ІТ-проекту. В даному підрозділ ми здійснимо збір та аналіз отриманих метрик автоматизації за допомогою мови програмування Python, дана інформація допоможе нам оптимізувати та покращити існуючий процес автоматизованого тестування для нашої системи.

Для початку розробимо Dataset із зібраними метриками для подальшого аналізу. Dataset – це набір даних або колекцію інформації, яка групується разом з метою подальшого аналізу чи обробки. Dataset може включати в себе різні типи даних, такі як текст, числа, зображення, аудіо, відео та інше. Основні риси якісного Dataset:

- **Чіткість та консистентність.** Добре структурований та організований Dataset має чіткі та консистентні дані. Кожен елемент Dataset повинен мати однаковий формат та чітко визначені назви стовпців.

- **Повнота.** Якісний Dataset має повні дані без пропусків або значущих відсутностей. Всі необхідні або очікувані атрибути повинні бути представлені для кожного запису.
- **Актуальність.** Дані повинні бути актуальними та відображати поточний стан або період часу, який вивчається.
- **Унікальність.** Якщо в Dataset включаються ідентифікатори, то вони повинні бути унікальними для кожного запису, щоб уникнути дублікатів та забезпечити однозначну ідентифікацію.
- **Надійність.** Добре зібраний Dataset повинен бути надійним, тобто дані в ньому повинні відображати реальність та бути достовірними.
- **Метадані.** Важливо мати чітко визначені метадані, які надають інформацію про походження даних, формат, одиниці вимірювання та інші важливі аспекти.
- **Репрезентативність.** Якісний Dataset повинен адекватно відображати той аспект реальності, який вивчається, та бути репрезентативним для висновків.
- **Зручність використання.** Dataset повинен бути легким у використанні та маніпуляціях. Інтерфейс для отримання доступу до даних та їх обробки повинен бути інтуїтивно зрозумілим.
- **Захист конфіденційності.** Якщо Dataset містить конфіденційні дані, необхідно захистити їх від несанкціонованого доступу та використання відповідно до вимог законодавства та політик безпеки.

Спираючись на поставлені вимоги був проведений збір необхідних метрик та розроблено Dataset, який включає в себе наступні атрибути:

- **Number Of Sprint** – атрибут, що демонструє порядковий номер ітерації під час якої додавалися тести, адже для реалізації процесу автоматизованого тестування застосовувалася гнучка методологія Scrum, що передбачає поділ життєвого циклу проєкту на невеликі, фіксовані за часом інтервали, відомі як спринти. Кожен спринт має чітко визначений часовий рамки, що у нашому випадку становить два тижні. Спринти є ключовою одиницею в Scrum для організації роботи, вони забезпечують стабільність у розробці, фокусуючись на конкретних

завданнях і досягненні мінімальної життєздатної функціональності в кожній ітерації.

- **Total Number Of Scenarios** – атрибут, що вказує на кількість тестів, які були заплановані та виконані на момент завершення спринту. Це важлива метрика, що відображає обсяг тестування, проведеного протягом даного періоду розробки. Чим більше сценаріїв тестування виконано, тим ширше охоплення функціональності продукту і тим більше можливість виявлення потенційних дефектів.

- **Pass Rate** – атрибут, що відображає відсоток тестів, які успішно завершилися відносно загальної кількості виконаних тестів. Ця метрика є важливим індикатором якості тестування та ефективності тестового покриття. Високий «Pass Rate» свідчить про те, що більшість функціональності продукту працює належним чином, а тестування допомагає вчасно виявляти та виправляти можливі проблеми. Низький «Pass Rate» може свідчити про недостатнє покриття тестами або про наявність значних дефектів у програмному забезпеченні, що вимагають уваги та виправлення.

- **Number Of Failed Scenarios** – атрибут, що вказує на кількість тестових сценаріїв, які не вдалося успішно завершити під час їх виконання. Ця метрика важлива, оскільки вказує на кількість потенційних проблем або дефектів, які були виявлені під час тестування. Кожен невдало пройдений сценарій може вказувати на можливі проблеми у функціональності продукту, які потребують уваги та подальшого виправлення перед випуском.

- **Critical Bugs** – атрибут, що відображає кількість виявлених дефектів з найвищим пріоритетом. Ці дефекти вважаються серйозними, оскільки вони можуть значно впливати на функціональність, безпеку та інші критичні аспекти програмного забезпечення. Вони можуть призвести до серйозних проблем для користувачів, бізнес-процесів або загальної безпеки системи. Виявлення цих дефектів важливе для того, щоб їх вчасно виправити та уникнути потенційних серйозних наслідків.

- **High Bugs** – атрибут, що відображає кількість виявлених дефектів, які мають високий пріоритет у виправленні. Ці дефекти, хоча не досягли критичного рівня, все ж можуть суттєво вплинути на функціонування системи або зручність користувачів. Вони визначаються як проблеми, які потребують уваги та виправлення, але не є негайно критичними. Зазвичай ці дефекти розглядаються для виправлення у наступних версіях програмного забезпечення, але їх важливо враховувати для підвищення загальної якості продукту.

- **Medium Bugs** – атрибут, що відображає кількість виявлених дефектів з помірним пріоритетом. Ці дефекти, хоча і не досягли критичного рівня, все ж можуть мати певний вплив на функціональність, безпеку або зручність використання продукту. Вони є менш важливими, ніж критичні або високопріоритетні дефекти, але їхнє виправлення, все одно, потрібне для підвищення якості продукту та задоволення потреб користувачів. Причини цих дефектів можуть бути різноманітними, від помилок в коді до недоліків у дизайні або функціональності.

- **Low Bugs** – атрибут, що відображає кількість виявлених дефектів з низьким пріоритетом. Ці дефекти або помилки в програмному забезпеченні мають мінімальний вплив на функціональність, безпеку або використання продукту. Вони є менш важливими порівняно з критичними, високими або середніми дефектами і можуть бути відкладені для виправлення у майбутніх версіях програмного забезпечення. Помилки цього рівня можуть включати незначні недоліки, косметичні проблеми або інші мінорні аспекти, які не суттєво впливають на використання продукту, але все ж заслуговують на увагу та виправлення.

- **Test Coverage %** – атрибут, що демонструє наскільки програмний код був протестований за допомогою тестових сценаріїв. Ця метрика вказує на відсоток коду, який було протестовано під час виконання автоматизованих скриптів. Високий рівень покриття тестами свідчить про те, що багато частин програми було перевірено на наявність помилок або дефектів, що сприяє забезпеченню якості програмного забезпечення. Низький рівень покриття, навпаки, може вказувати на те, що деякі частини програми не були протестовані, що може призвести до

помилки та недоліків у функціонуванні продукту. Таким чином, підвищення рівня тестового покриття може покращити якість програмного забезпечення і зменшити його вразливість до помилок та дефектів.

Після збору метрик та розробки Dataset, ми можемо приступити до аналізу, для цього використаємо мову програмування Python та популярні бібліотеки, такі як: Pandas, Matplotlib та Seaborn.

Для початку продемонструємо всі види дефектів, які були знайдені під час процесу автоматизованого тестування. Для цього розробимо наступний код за допомогою мови програмування Python та бібліотек Pandas і Matplotlib:

Прог 3.1 – Код для побудова кругової діаграми загальної кількості знайдених багів за всі спринти

```
import pandas as pd
import matplotlib.pyplot as plt
file_path = 'TestResults.csv'
data = pd.read_csv(file_path)
total_bugs = data[['NumberOfCriticalBugs', 'NumberOfHighBugs',
'NumberOfMediumBugs', 'NumberOfLowBugs']].sum()
plt.figure(figsize=(12, 12))
plt.pie(total_bugs, labels=[""] * len(total_bugs), autopct='%1.1f%%', colors=['red',
'orange', 'yellow', 'green'])
plt.title('Distribution of Bugs Categories', fontsize=20)
plt.legend(total_bugs.index, loc="best", fontsize=14)
plt.show()
```

Даний код дозволяє нам знайти та показати загальну кількість виявлених дефектів під час автоматизованого тестування в процесі розробки. Дана діаграма відображає важливість ефективного використання автоматизованого тестування для замовника та всіх учасників команди. Крім того, результати тестування можуть допомогти визначити області, які потребують додаткової уваги або вдосконалення в процесі розробки програмного забезпечення.

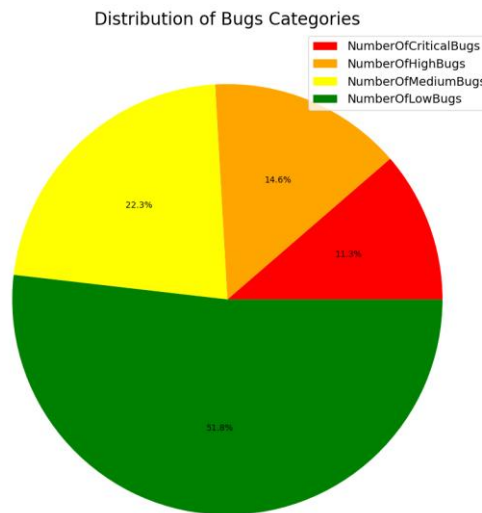


Рисунок 3.16 – Діаграма загальної кількості знайдених багів за час використання автоматизованого тестування

Наступним кроком буде представлення тенденції виявлення дефектів за допомогою автоматизованого тестування. Для цього розробимо наступний код за допомогою мови програмування Python та бібліотек Pandas, Matplotlib і Seaborn:

Прог 3.2 – Код для побудови точкової гістограми тенденції виявлення багів

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
file_path = 'TestResults.csv'
data = pd.read_csv(file_path)
plt.figure(figsize=(12, 8))
sns.pointplot(x=data['NumberOfSprint'], y=data['NumberOfCriticalBugs'],
color='red', label='Critical Bugs')
sns.pointplot(x=data['NumberOfSprint'], y=data['NumberOfHighBugs'],
color='orange', label='High Bugs')
sns.pointplot(x=data['NumberOfSprint'], y=data['NumberOfMediumBugs'],
color='yellow', label='Medium Bugs')
sns.pointplot(x=data['NumberOfSprint'], y=data['NumberOfLowBugs'],
color='green', label='Low Bugs')
plt.title('Bugs Count for Each Sprint')
plt.xlabel('Sprint Number')
```

```
plt.ylabel('Number of Bugs')
plt.ylim(-1, 25)
plt.legend()
plt.show()
```

Даний код дозволяє нам виявити та проілюструвати тенденцію виявлення дефектів за допомогою автоматизованого тестування протягом розробки. Дана аналітика є важливою для виявлення ефективності процесу тестування протягом часу та визначення можливих змін або покращень у стратегії тестування. Така інформація може допомогти команді покращити якість продукту та зменшити кількість виявлених дефектів у майбутніх релізах.

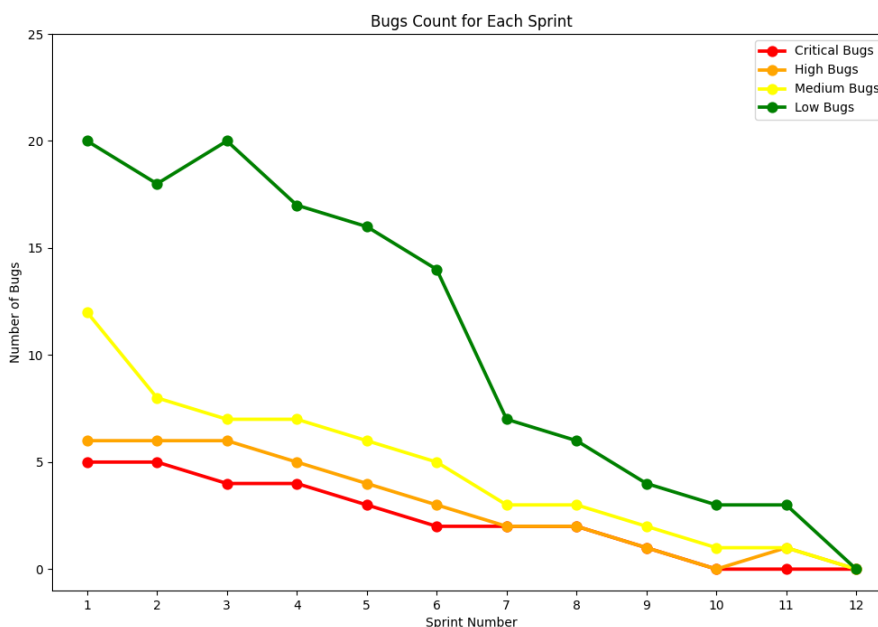


Рисунок 3.17 – Діаграма загальної кількості знайдених багів за час використання автоматизованого тестування

Далі знайдемо взаємозв'язок між кількістю виявлених дефектів та рівнем покриття ІТ-проєкту автоматизованими тестами. Дана аналітика дозволить нам з'ясувати, наскільки ефективно автоматизовані тести покривають різні частини програмного забезпечення та як це впливає на кількість виявлених дефектів. Така інформація може бути корисною для визначення пріоритетів у тестуванні та вдосконалення стратегії тестування для покращення якості продукту. Для цього ми напишемо відповідний код на мові програмування Python та бібліотек Pandas, Matplotlib і Seaborn:

Прог 3.3 – Код для побудови діаграми залежності кількості дефектів від покриття ІТ-проєкту тестами

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
file_path = 'TestResults.csv'
data = pd.read_csv(file_path)
correlation_critical = data['NumberOfCriticalBugs'].corr(data['TestCoverage'])
correlation_high = data['NumberOfHighBugs'].corr(data['TestCoverage'])
correlation_medium = data['NumberOfMediumBugs'].corr(data['TestCoverage'])
correlation_low = data['NumberOfLowBugs'].corr(data['TestCoverage'])
plt.figure(figsize=(15, 5))
plt.subplot(1, 4, 1)
sns.lineplot(x=data['NumberOfSprint'], y=data['NumberOfCriticalBugs'],
label='Critical Bugs', marker='o')
sns.lineplot(x=data['NumberOfSprint'], y=data['TestCoverage'], label='Test
Coverage', marker='o')
plt.subplot(1, 4, 2)
sns.lineplot(x=data['NumberOfSprint'], y=data['NumberOfHighBugs'],
label='High Bugs', marker='o')
sns.lineplot(x=data['NumberOfSprint'], y=data['TestCoverage'], label='Test
Coverage', marker='o')
plt.subplot(1, 4, 3)
sns.lineplot(x=data['NumberOfSprint'], y=data['NumberOfMediumBugs'],
label='Medium Bugs', marker='o')
sns.lineplot(x=data['NumberOfSprint'], y=data['TestCoverage'], label='Test
Coverage', marker='o')
plt.subplot(1, 4, 4)
sns.lineplot(x=data['NumberOfSprint'], y=data['NumberOfLowBugs'],
label='Low Bugs', marker='o')
```

```
sns.lineplot(x=data['NumberOfSprint'], y=data['TestCoverage'], label='Test
Coverage', marker='o')
plt.tight_layout()
plt.show()
```

Розглянувши дану діаграму, ми можемо зробити висновок, що спостерігається виражена обернена залежність: чим більше тестів автоматизовано, тим менше дефектів буде виявлено під час релізу продукту. Це свідчить про те, що високий рівень автоматизації тестування сприяє покращенню якості програмного забезпечення та знижує ризик виявлення дефектів під час релізу. Такий аналіз може бути корисним для планування тестування та визначення стратегії автоматизації, спрямованої на підвищення ефективності та якості розробки.

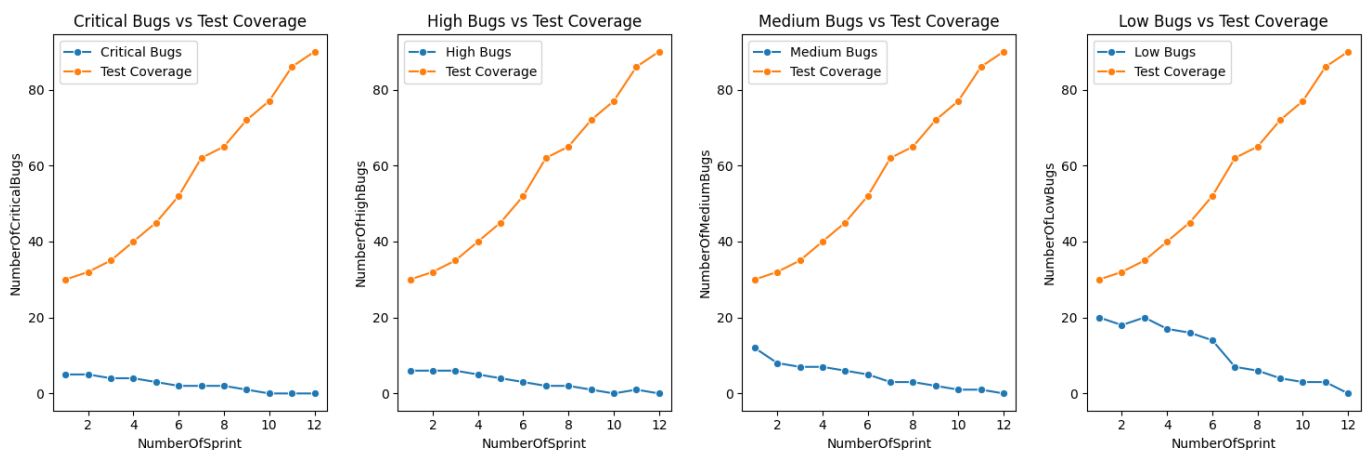


Рисунок 3.18 – Діаграма залежності кількості дефектів від покриття ІТ-проєкту тестами

Далі розглянемо метрики автоматизованого тестування. Почнемо з того, як швидко додавалися автоматизовані тести в кожному спринті. Цей аналіз дозволить нам визначити тенденції у зростанні обсягу автоматизованих тестів протягом часу, що може вказати на ефективність розробленої стратегії тестування. Також ці дані можуть служити основою для подальшого удосконалення процесу розробки та впровадження кращих практик в тестуванні. Для цього ми розробимо відповідний код на Python та бібліотек Pandas, Matplotlib і Seaborn:

Прог 3.4 – Код для демонстрації тенденція додавання автоматизованих тестів для кожного із спринтів

```
import pandas as pd
```

```

import matplotlib.pyplot as plt
import seaborn as sns
file_path = 'TestResults.csv'
data = pd.read_csv(file_path)
plt.figure(figsize=(10, 6))
sns.barplot(x=data['NumberOfSprint'], y=data['PassRate'], palette='viridis')
plt.title('Pass Rate for Each Sprint')
plt.xlabel('Sprint Number')
plt.ylabel('Pass Rate')
plt.show()

```

Після дослідження стовпчастої діаграми ми можемо зробити оцінку темпу написання тестів. Важливо врахувати, що швидкість написання тестів є ключовим фактором в процесі розробки програмного забезпечення, оскільки вона відображає загальну продуктивність команди та ефективність використання ресурсів. Додатковою інформацією може бути аналіз зміни темпу написання тестів з часом, порівняння із планованими обсягами тестування та встановленням причин за можливими коливаннями в швидкості написання тестів.

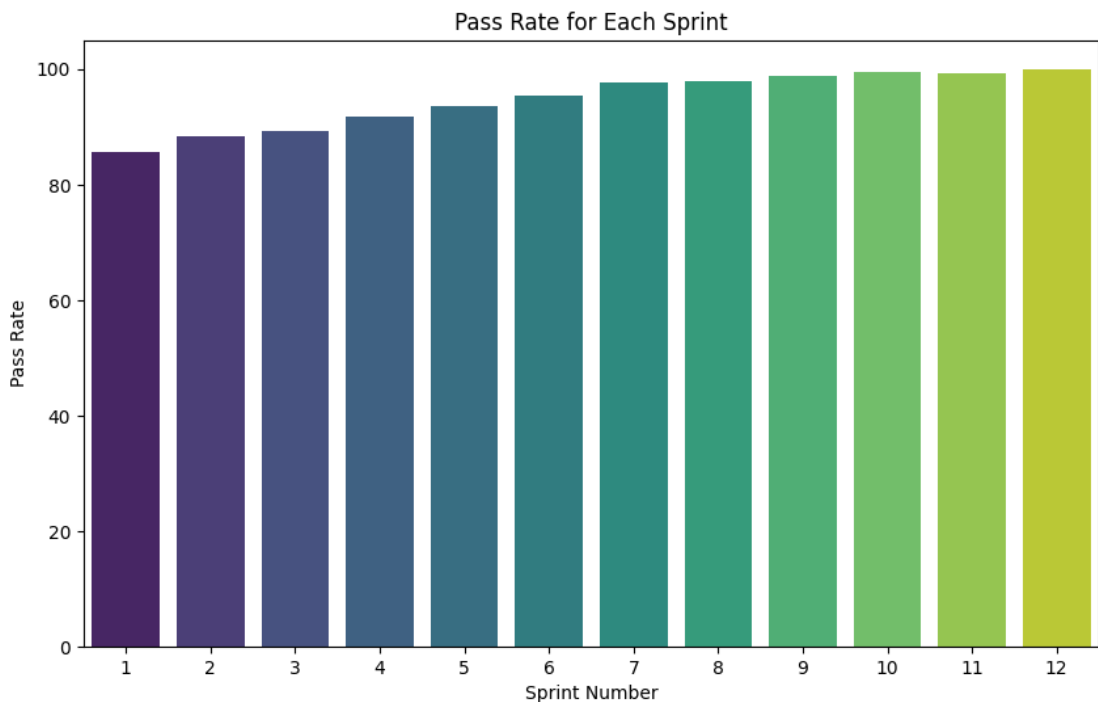


Рисунок 3.19 – Тенденція додавання автоматизованих тестів для кожного із спринтів

Наступним кроком буде відображення динаміки змін у тестовому покритті нашого IT-проєкту. Додатковою інформацією може бути аналіз тенденцій у покращенні або погіршенні покриття тестами з часом, а також виявлення факторів, які впливають на ці зміни. Розробимо код, який демонструватиме ці зміни та їх вплив на загальну якість продукту, використаємо Python та бібліотеки Pandas, Matplotlib і Seaborn:

Прог 3.5 – Код для демонстрації тенденція зміни тестового покриття для IT-проєкту

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
file_path = 'TestResults.csv'
data = pd.read_csv(file_path)
plt.figure(figsize=(10, 6))
sns.barplot(x=data['NumberOfSprint'], y=data['TestCoverage'], palette='viridis')
plt.title('Test Coverage For Each Sprint')
plt.xlabel('Sprint Number')
plt.ylabel('Test Coverage')
plt.show()
```

Відзначається, що з кожним доданим автоматизованим сценарієм збільшувалося тестове покриття. Цей процес ілюструє значний внесок у підвищення якості тестування, оскільки збільшення покриття сприяє більш повному охопленню функціональності продукту тестами, що збільшить кінцеву якість та ефективність розроблювано програмного забезпечення. Додаткова інформація може включати аналіз конкретних змін у покритті та їхній вплив на ефективність тестування, а також розгляд можливостей для подальшого розширення покриття тестами.

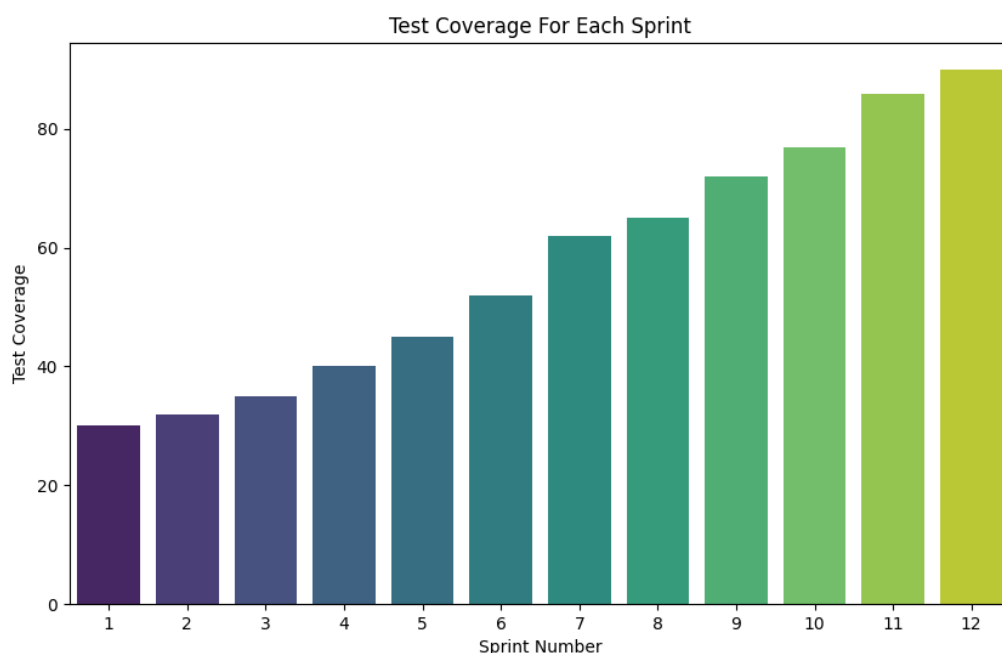


Рисунок 3.5 – Тенденція зміни тестового покриття

Далі розглянемо динаміку змін у показнику «Pass Rate» для наших автоматизованих тестів. За допомогою коду на мові програмування Python ми проаналізуємо ці зміни та їх вплив на якість тестування. Додатковою інформацією може бути оцінка причин варіацій у «Pass Rate» та виявлення можливих шляхів для покращення ефективності тестування.

Прог 3.20 – Код для демонстрації тенденція зміни показника Pass Rate для запусків автоматизованих тестів

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
file_path = 'TestResults.csv'
data = pd.read_csv(file_path)
plt.figure(figsize=(10, 6))
sns.barplot(x=data['NumberOfSprint'],          y=data['TotalNumberOfScenarios'],
palette='viridis')

plt.title('Total Number Of Scenarios For Each Sprint')
plt.xlabel('Sprint Number')
plt.ylabel('Total Number Of Scenarios')
plt.show()
```

```

correlation = data['TotalNumberOfScenarios'].corr(data['PassRate'])
print(f'Correlation between TotalNumberOfScenarios and PassRate:
{correlation}')

plt.figure(figsize=(8, 6))
sns.heatmap(data[['TotalNumberOfScenarios', 'PassRate']].corr(), annot=True,
cmap='coolwarm', fmt=".2f")

plt.title('Correlation Heatmap')
plt.show()

```

Після аналізу результатів стало очевидним, що кожен наступний спринт був успішнішим за попередній. Це пояснюється зменшенням кількості виявлених дефектів, що сталося внаслідок збільшення кількості автоматизованих тестів.

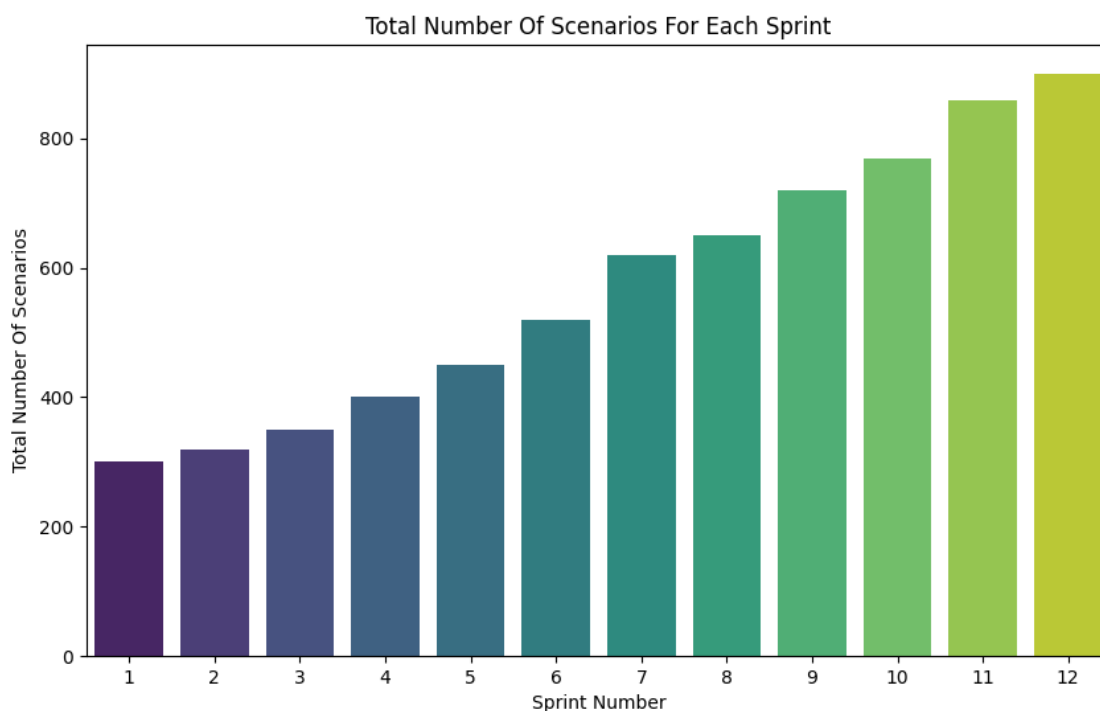


Рисунок 3.21 – Тенденція зміни показника «Pass Rate»

Хочеться відзначити дуже сильну кореляцію між показниками: із збільшенням кількості сценаріїв зростає показник «Pass Rate», оскільки значна частина дефектів була виявлена на ранніх етапах розробки. Діаграма вказує на ефективність стратегії тестування, що базується на використанні багатьох сценаріїв. Підвищення «Pass Rate» може свідчити про вдосконалення якості продукту завдяки ретельній перевірці його функціональності. З цього можна зробити висновок, що вкладення зусиль у розробку широкого спектру тестових

сценаріїв виявляється ефективним підходом до забезпечення якості програмного забезпечення.

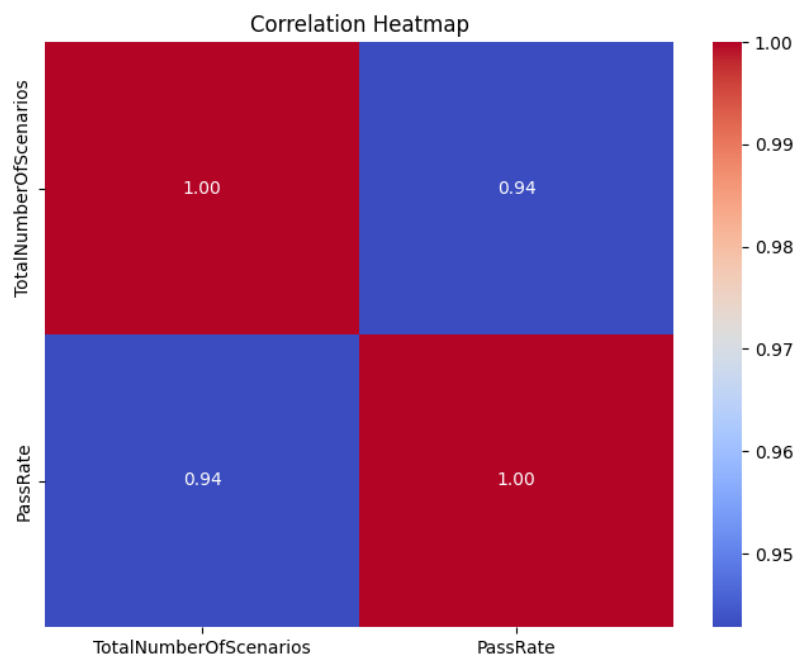


Рисунок 3.22 – Теплова карта тенденція зміни показника «Pass Rate» від кількості автоматизованих тестів

Основаючись на отриманих результатах аналізу, можна дійти до висновку, що автоматизоване тестування позитивно вплинуло на розробку системи: пришвидшився час розробки, адже менше часу витрачається на виправлення помилок на пізніх стадіях, та покращилась якість розроблюваного продукту. Можна підкреслити ключові висновки спираючись на діаграми:

- **Зменшення кількості багів:** аналіз графіків точкових гістограм багів за кожен спринт дозволяє встановити, що із збільшенням кількості автоматизованих тестів спостерігається зниження кількості багів у різних категоріях (Critical, High, Medium, Low). Діаграма кореляцій показує сильну зворотну залежність між кількістю багів та рівнем покриття тестами. Це означає, що зі збільшенням покриття тестами кількість багів зменшується.

- **Сильна кореляція між кількістю сценаріїв та показником «Pass Rate»:** аналіз кореляцій показав дуже сильний позитивний зв'язок між загальною кількістю сценаріїв та показником «Pass Rate». Це свідчить про те, що зі

збільшенням кількості тестових сценаріїв підвищується ефективність тестування, і більш великий відсоток тестів проходить успішно.

- **Теплова карта кореляцій:** теплова карта відображає взаємозв'язки між різними атрибутами проєкту. Яскраво виражені позитивні кореляції між показниками, такими як «Pass Rate» та «Test Coverage», з іншими метриками проєкту.
- **Діаграма розподілу багів за категоріями:** кругова діаграма показує загальний розподіл багів за категоріями (Critical, High, Medium, Low). Ця візуалізація спрощує сприйняття того, яку частину загальної кількості багів складає кожна категорія.
- **Стовпчикові діаграми для «Pass Rate», «Test Coverage» та «Total Number Of Scenarios»:** демонструють позитивні тенденції зменшення кількості багів та покращення якості ПЗ.

На основі даних висновків, можна впевнено сказати, що процес впровадження автоматизованого тестування для системи пройшов успішно. Даний процес працюю бездоганно та не потребує жодних кроків для оптимізації.

3.4 Висновок до розділу

У даному розділі були розглянуті та вирішені наступні питання: розробка тестового фреймворку та написання автоматизованих тестів, інтеграція автоматизованого тестування в CI/CD процес, а також аналіз отриманих результатів та оптимізація процесу тестування.

Під час реалізації проєкту було зроблено значний крок у напрямку підвищення ефективності та надійності програмного забезпечення шляхом автоматизації тестування. Розроблений тестовий фреймворк дозволяє швидко створювати та виконувати автоматизовані скрипти. Його інтеграція в CI/CD процес

забезпечує автоматичне виконання тестів під час кожної зміни вихідного коду, що дозволяє вчасно виявляти та виправляти помилки.

Аналіз отриманих результатів тестування надав важливу інформацію щодо якості програмного забезпечення та його відповідності вимогам. На основі цього аналізу були прийнято рішення залишити процес автоматизованого без змін, адже все працює ідеально, що дозволяє пришвидшити процес та розробки та покращити ефективність системи.

Отже, впровадження системи автоматизованого тестування для ІТ-проєкту є важливим кроком у забезпеченні якості програмного забезпечення. Результати даного процесу створюють основу для подальшого розвитку та вдосконалення процесу розробки програмного забезпечення, сприяючи зменшенню часу необхідного для тестування та підвищенню якості продукту.

ВИСНОВКИ

Перший розділ дипломної роботи присвячено теоретичним аспектам автоматизованого тестування програмного забезпечення. У ньому проаналізовано сутність поняття автоматизованого тестування, його основні функції та місце у життєвому циклі тестування ПЗ. Проведено детальний аналіз різновидів автоматизованого тестування, їхніх особливостей, переваг та недоліків. Серед переваг відзначено підвищену швидкість тестування, зниження витрат у довгостроковій перспективі, та збільшення точності результатів. Проте, зазначено і обмеження, зокрема складність створення та підтримки тестових скриптів, високу вартість розробки, та обмежену здатність виявлення певних видів дефектів. Таким чином, цей розділ закладає теоретичну основу для подальшого дослідження та впровадження автоматизованого тестування для системи.

У другому розділі дипломної роботи здійснено системний аналіз обраного ІТ-проєкту для впровадження автоматизованого тестування, включаючи визначення головних особливостей системи, її функцій та критичних зон. Розглянуто типи тестової документації, необхідної для успішного впровадження, та проаналізовано процес і етапи інтеграції автоматизованого тестування, такі як аналіз потреб проєкту, вибір інструментів і технологій, розробка та виконання плану дій. Детально розглянуто кожен етап для забезпечення ефективного впровадження тестування.

У третьому розділі розглянуто розробку тестового фреймворку, написання автоматизованих тестів, інтеграцію автоматизованого тестування в CI/CD процес, аналіз результатів і оптимізацію тестування. Автоматизація підвищила ефективність та надійність ПЗ, дозволяючи швидко створювати і виконувати тестові скрипти. Інтеграція в CI/CD процес забезпечила автоматичне тестування при кожній зміні коду, що допомогло своєчасно виявляти та виправляти помилки. Аналіз результатів підтвердив високу якість ПЗ, і було вирішено залишити процес без змін. Впровадження автоматизованого тестування стало важливим кроком для

забезпечення якості ПЗ, зменшення часу тестування та підвищення ефективності розробки.

Отже, проведені дослідження продемонструвало важливість і ефективність впровадження автоматизованого тестування у розробку програмного забезпечення. Теоретичний аналіз, системний огляд конкретного ІТ-проєкту та практичне впровадження тестового фреймворку показали, що автоматизація тестування сприяє покращенню якості продукту, знижує витрати і час на тестування, а також підвищує ефективність розробки. Ці висновки підтверджують доцільність подальшого використання та розвитку методів автоматизованого тестування, що забезпечить конкурентоспроможність і надійність програмних рішень у сучасних умовах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Graham D., Fewster M. Experiences of Test Automation. U.S. Corporate and Government Sales: Pearson Education, Inc, 2012. 114 p.
2. Myers G. J., Badgett T., Sandler C. The Art of Software Testing. New Jersey: John Wiley & Sons, Inc, 2012. 254 p.
3. Heusser M., Larsen M. Software Testing Strategies. Birmingham: Packt Publishing Ltd, 2023. 378 p.
4. Куліков С. С. Тестування програмного забезпечення. Мінськ: видавництво «Чотири четверті», 2020. 314 с.
5. Axelrod A. Complete Guide to Test Automation. New York: Springer Science+Business Media New York, 2018. 542 p.
6. Говорущенко Т.О., Малярчук Р.А. Аналіз сучасних методологій розробки складних програмних проєктів. Хмельницький: ХНУ, 2014. 219 с.
7. Гузенко О.П., Гійко О.М., Яковлева Л.В. Тестування програмного забезпечення: підручник. К.: Каравела, 2019. 328 с.
8. Черкаський О.В., Гаврилюк В.В. Тестування програмного забезпечення: навч. посібник. К.: НТУУ «КПІ», 2018. 352 с.
9. Jorgensen P. Software Testing: A Craftsman's Approach / P. Jorgensen – 4th Edition. Boca Raton: CRC Press, 2013. 389 p.
10. Яковенко М.В., Біляченко Ю.І., Грицай А.Ю. Методи тестування програмного забезпечення: навч. посібник. К.: НАУ, 2016. 376 с.
11. Frederick Phillips Brooks, Jr. The Mythical Man-Month: Essays on Software Engineering (Фредерік П. Брукс, Міфічний людино-місяць: Есе з програмної інженерії). – Massachusetts, Reading, Addison-Wesley Professional, 1995. 322 p.
12. Binder R. Testing Object-Oriented Systems: Models, Patterns, and Tools / R. Binder – Boston: Addison-Wesley Professional, 2000. 864 p.
13. Pol D. Software Testing: A Complete Guide / D. Pol New York: Pearson Education, 2007. 398 p.

- 14.** Graham D., Fewster M. Software Test Automation. Great Britain: Biddies Ltd, Guildford and King's Lynn, 1994. 592 p.
- 15.** Alsmadi I. Advanced Automated Software Testing: Frameworks for Refined Practice. The United States of America: Information Science Reference (an imprint of IGI Global), 2012. 289 p.
- 16.** Huizinga D., Kolawa A. Automated Defect Prevention. New Jersey: John Wiley & Sons, Inc, 2007. 454 p.
- 17.** Robert C. M. Clean Code. Boston: Pearson Education, Inc, 2009. 462 p.
- 18.** Verma N. Mobile Test Automation with Appium. Birmingham: Packt Publishing Ltd, 2017. 250 p.
- 19.** Ratan A. Practical Network Automation. Birmingham: Packt Publishing Ltd, 2017. 261 p.
- 20.** Jones R. A. Selenium WebDriver for Functional Automation Testing. Test 4 Success, LLC, 2016. 163 p.
- 21.** Sommerville I. Software Engineering: United States Edition / I. Sommerville, 9 edition., Boston: Pearson, 2010. 792 p.
- 22.** Cole R. Scotcher E. Agile project management. United Kingdom: Pearson Education Limited, 2015. 198 p.
- 23.** Burns D. Selenium 2 Testing Tools: Beginner's Guide. United Kingdom: Birmingham, 2012. 437 p.
- 24.** Hammarberg M., Sunden J. Kanban in Action. Boston: Manning Publications Co, 2014. 361 p.
- 25.** Klaus L., Kaltenecker S. Kanban change leadership. New Jersey: John Wiley & Sons, Inc., Hoboken, 2015. 313 p.
- 26.** Raj J. Software Testing and Quality Assurance: Theory and Practice / J. Raj – London: Pearson Education, 2008. 402 p.
- 27.** Spillner A. Software Testing Foundations / A. Spillner, T. Linz, H. Schaefer – 4th Edition. Rocky Hill: Rocky Nook, 2014. 319 p.
- 28.** Spillner A. Agile Software Testing: Best Practices for Large Enterprises / A. Spillner, T. Rossner, M. Winter – Berlin: Springer, 2015. 258 p.

- 29.** Fewster M. Software Test Automation / M. Fewster, D. Graham – Boston: Addison-Wesley Professional, 1999. 386 p.
- 30.** Pettichord B. Lessons Learned in Software Testing: A Context-Driven Approach / B. Pettichord, J. Bach, C. Kaner Hoboken: Wiley, 2002. 320 p.
- 31.** Rex Black. Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing. - 3rd Edition. New York: Wiley, 2009. 528 p.
- 32.** Rex Black. Critical Testing Processes: Plan, Prepare, Perform, Perfect. - 1st Edition. New York: Addison-Wesley Professional, 2003. 344 p.
- 33.** What is automation testing? URL: <https://anywhere.epam.com/en/blog/what-is-automation-testing> (Дата звернення: 25.04.2024).
- 34.** Automation Testing – Software Testing. URL: <https://www.geeksforgeeks.org/automation-testing-software-testing/> (Дата звернення: 23.04.2024).
- 35.** Automated software testing. URL: <https://www.atlassian.com/continuous-delivery/software-testing/automated-testing> (Дата звернення: 28.04.2024).
- 36.** What is Automation Testing: Benefits, Strategy, Tools. URL: <https://www.browserstack.com/guide/automation-testing-tutorial> (Дата звернення: 10.05.2024).
- 37.** Automated testing. URL: <https://www.functionize.com/automated-testing> (Дата звернення: 14.04.2024).
- 38.** What is Automation Testing? Ultimate Guide & Best Practices. URL: <https://katalon.com/resources-center/blog/what-is-automation-testing> (Дата звернення: 23.04.2024).
- 39.** Manual Testing vs Automated Testing: Key Differences. URL: <https://www.testrail.com/blog/manual-vs-automated-testing/> (Дата звернення: 05.05.2024).
- 40.** Mastering Automated Testing: Your Ultimate Guide. URL: <https://testsigma.com/automated-testing> (Дата звернення: 01.05.2024).

- 41.** Automation Testing for Web & Mobile Apps. URL: <https://www.lambdatest.com/automation-testing> (Дата звернення: 30.04.2024).
- 42.** Elevate quality engineering with the power of AI. URL: <https://www.tricentis.com/> (Дата звернення: 12.05.2024).
- 43.** Data Analysis with Python. URL: <https://www.freecodecamp.org/learn/data-analysis-with-python/> (Дата звернення: 23.04.2024).
- 44.** Data Analysis with Python. URL: <https://www.geeksforgeeks.org/data-analysis-with-python/> (Дата звернення: 25.04.2024).
- 45.** Data Analytics With Python: Use Case Demo URL: <https://www.simplilearn.com/tutorials/data-analytics-tutorial/data-analytics-with-python> (Дата звернення: 19.11.2023).
- 46.** Data Analysis with Python. URL: <https://www.coursera.org/learn/data-analysis-with-python> (Дата звернення: 07.05.2024).
- 47.** Python Programming Tutorials URL: <https://pythonprogramming.net/data-analysis-tutorials/> (Дата звернення: 23.04.2024).
- 48.** Data analysis using Pandas. URL: <https://www.geeksforgeeks.org/python-data-analysis-using-pandas/> (Дата звернення: 28.04.2024).
- 49.** Exploratory Data Analysis with Pandas. URL: <https://www.kaggle.com/code/kashnitsky/topic-1-exploratory-data-analysis-with-pandas> (Дата звернення: 29.04.2024).
- 50.** A Quick Introduction to Data Analysis with Pandas. URL: <https://medium.com/velotio-perspectives/a-quick-introduction-to-data-analysis-with-pandas-42b6e0cad260> (Дата звернення: 10.05.2024).
- 51.** Boost your Data Analysis with Pandas. URL: <https://towardsdatascience.com/boost-your-data-analysis-with-pandas-69c4be5d73bb> (Дата звернення: 12.05.2024).
- 52.** Data Visualization with Python Seaborn. URL: <https://www.geeksforgeeks.org/data-visualization-with-python-seaborn/> (Дата звернення: 15.05.2024).

53. Data Visualization using Seaborn. URL: <https://towardsdatascience.com/data-visualization-using-seaborn-fc24db95a850> (Дата звернення: 14.04.2024).

54. How to use Seaborn for Data Visualization. URL: <https://www.section.io/engineering-education/seaborn-tutorial/> (Дата звернення: 23.04.2024).

Ім'я користувача: Комп'ютерної математики та інформаційної безпеки...	ID перевірки: 1016257979
Дата перевірки: 17.05.2024 02:07:19 EEST	Тип перевірки: Doc vs Internet + Library
Дата звіту: 17.05.2024 02:09:38 EEST	ID користувача: 100005746

Назва документа: Диплом Деменко

Кількість сторінок: 77 Кількість слів: 13637 Кількість символів: 109222 Розмір файлу: 1.98 MB ID файлу: 1016045674

2.65% Схожість

Найбільша схожість: 0.65% з Інтернет-джерелом (<https://deepnote.com/app/hwu-8ca7/Coursework2-e06f0451-7443-4a68>).

1.36% Джерела з Інтернету	90	Сторінка 79
2.24% Джерела з Бібліотеки	161	Сторінка 79

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0% Вилучень

Немає вилучених джерел