МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ

Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
Аліна САВЧЕНКО.
«_____» _____ 2023р.

КВАЛІФІКАЦІЙНА РОБОТА
(ДИПЛОМНИЙ ПРОЄКТ, ПОЯСНЮВАЛЬНА ЗАПИСКА)
ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ **"БАКАЛАВР"**
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
"ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: " Програмний застосунок тестування виробничого програмного забезпечення крупного транспортного підприємства "

Виконавець: студентка групи УС-411Б Власова Катерина Євгеніївна .

Керівник: професор Віноградов Микола Анатолійович .

Нормоконтролер: ст. викл. Олександр ШЕВЧЕНКО .

Київ – 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук і технологій

Кафедра комп'ютерних інформаційних технологій

Освітній ступінь: «Бакалавр»

Галузь знань, спеціальність, освітньо-професійна програма:

12 «Інформаційні технології», 122 «Комп'ютерні науки», «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Аліна САВЧЕНКО

"_____"_____2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи студентки
Власової Катерини Євгеніївни

1. **Тема кваліфікаційної роботи:** «Програмний застосунок тестування виробничого програмного забезпечення крупного транспортного підприємства» затверджена наказом ректора № 623/ст. від 12.04.2023р.
2. **Термін виконання роботи:** 15.05.2023р. - 25.06.2023р.
3. **Вихідні дані до роботи:** ОС Windows, дані про систему Tesla Touchscreen, Pycharm, TestRail.
4. **Зміст пояснювальної записки (перелік питань, що підлягають розробці):** аналіз теорії автоматизованого тестування програмного забезпечення та розробка системи тестування для автомобільного підприємства.
5. **Перелік обов'язкового ілюстративного матеріалу:** рисунки моделей ЖЦ розробки ПЗ; рисунки індикаторів несправності, тест-кейсів, програмного коду; таблиця тест-плану тестування.

6. **Календарний план-графік**

№ п/п	Завдання	Термін виконання	Підпис керівника
1.	Аналіз предметної області дослідження	10.05.2023-11.05.2023	Виконано
2.	Провела збір і аналіз інформації щодо теми індивідуального завдання.	12.05.2023-14.05.2023	Виконано
3.	Написати перший розділ кваліфікаційної роботи.	15.05.2023-19.05.2023	Виконано
4.	Аналіз використання видів тестування.	24.05.2023-26.05.2023	Виконано
5.	Написати другий розділ кваліфікаційної роботи.	28.05.2023 - 29.05.2023	Виконано
6.	Визначення основних функцій, їх впровадження. Створення проєкту, реалізація тестування.	29.05.2023 - 30.05.2023	Виконано
7.	Написати третій розділ кваліфакційної роботи та зробити висновки.	30.05.2023	Виконано
8.	Оформлення та друк пояснювальної записки кваліфікаційної роботи.	12.06.2023-13.06.2023	Виконано
9.	Підготовка презентації та доповіді для виступу.	14.06.2023-16.06.2023	Виконано
10.	Захист кваліфікаційної роботи.	19.06.2023-23.06.2023	Виконано

7. **Дата видачі завдання**: «15» травня 2023 р.

Керівник дипломного проекту _____ Микола ВІНОГРАДОВ
(підпис керівника) (П.І.Б.)

Завдання прийняла до виконання _____ Катерина ВЛАСОВА
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Програмний застосунок тестування виробничого програмного забезпечення крупного транспортного підприємства» викладена на 68 сторінках, містить 42 рисунки, 1 таблицю та 5 літературно-наукових джерел.

Мета кваліфікаційної роботи: розробити програмний застосунок для тестування програмного забезпечення автомобільного підприємства.

Об'єкт дослідження: автоматизоване та мануальне тестування ПЗ, автомобільне ПЗ від Тесла.

Предмет дослідження: тестування програмного забезпечення.

Метод дослідження: - методи тестування програмного забезпечення;

- методи визначення якості продукту;

- методи верифікації програмного забезпечення;

- методи автоматизації створення тест-кейсів.

Результат проекту: створена формальна система тестування для автомобільного програмного забезпечення, написані функціональні тести.

ТЕСТУВАННЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ТЕСТ, ТЕСТ-КЕЙС, ІНДИФІКАТОР.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ПЗ	Програмне забезпечення.
Ad-hoc testing	Вид тестування, який виконується без підготовки до тестування продукту, без визначення очікуваних результатів, проєктування тестових сценаріїв.

ВСТУП

На сьогоднішній день тестування програмних продуктів є дуже важливою та актуальною темою. Програмне забезпечення є складним та надзвичайно важливим елементом в сучасному світі, яке використовується в багатьох галузях, таких як фінанси, медицина, транспорт, телекомунікації, виробництво, розробка програмного забезпечення та інші.

Тестування ПЗ (програмного забезпечення) допомагає забезпечити високу якість та надійність програмних продуктів, зменшує ризик виникнення помилок, що можуть призвести до серйозних наслідків, таких як втрата даних, фінансові втрати або порушення безпеки.

При відсутності або низькому рівні тестування ПЗ , користувачі можуть зіткнутися з недоліками та помилками у програмному забезпеченні, що може призвести до відмови від використання продукту або негативного впливу на репутацію компанії-розробника.

Також, з огляду на швидкий розвиток технологій та змінні вимоги користувачів, тестування ПЗ є постійним та неперервним процесом, який дозволяє забезпечити високу якість та надійність програмного забезпечення в умовах постійної зміни вимог та технологій.

Отже, актуальність тестування ПЗ полягає в тому, що воно допомагає забезпечити високу якість та надійність програмних продуктів, зменшує ризик виникнення помилок та підвищує задоволення користувачів продуктом.

Мета і завдання виконання кваліфікаційної роботи - розробити програмний застосунок для тестування програмного забезпечення автомобільного підприємства.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- провести аналіз тестування програмного забезпечення;
- створити свою систему тестування для автомобільного підприємства;
- підвищити швидкість тестового процесу за допомогою автоматизації тестування;

- знизити вартість тестового процесу.

Об’єкт дослідження - процес забезпечення якості та надійності програмного забезпечення.

Предмет дослідження - середовище автоматизованого тестування, створене для забезпечення зручності при написанні автоматизованих тестів у сфері автомобільного бізнесу.

Методи дослідження - графоаналітичний метод, порівняльний аналіз, статистичні методи, аналіз ризиків, системний підхід.

Наукова новизна дослідження - розробка нових, специфічних методів, інструментів та підходів, що дозволяють підвищити якість тестування, скоротити його тривалість та забезпечити більш ефективну взаємодію між тестувальним середовищем та об’єктом тестування.

РОЗДІЛ 1
ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Системи з програмним забезпеченням

Системи з програмним забезпеченням - невід’ємна частина нашого життя від щоденних (застосунки на телефоні) до споживчих (автомобілі, комп'ютери). Програмне забезпечення, яке не працює, зазвичай призводить до маленьких незручностей, а іноді і до катастроф, та ретельне тестування системи та її компонентів, які супроводжуються документацією, може допомогти знизити ризик відмови системи під час роботи. Тому тестування ПО дуже важливе у життєвому циклі проекту. Існує сім принципів тестування, які допоможуть зрозуміти у чому постає суть тестування:

1. Тестування демонструє наявність дефектів, а не їх відсутність. Якщо дефектів не було знайдено - це не означає, що їх немає, а от у зворотному порядку це не працює.
2. Вичерпне тестування недосяжне. Протестувати абсолютно всі комбінації користування та передумов - неможливо, за винятком тривіальних випадків. Тому тестування проводиться за допомогою аналізу ризиків, методів тестування, розстановка пріоритетів.
3. Раннє тестування зберігає гроші та час. Аби побачити дефекти раніше, потрібно почати тестування з самого початку створення програмного продукту.
4. Кластеризація дефектів;
5. Парадокс пестициду. Постійне оновлення тестів, створення нових, тому що постійний запуск одних і тих самих тестів скоро не буде ефективним.

6. Тестування залежить від контексту. Тестування, наприклад, машинного ПО буде відрізнятися від тестування мобільного ПО, через критичну потребі у безпеці.
7. Омана про відсутність помилок. Деякі організації очікують від тестувальників повного тестування системі, але принципи 1 і 2 говорять про неможливість цього.

1.2. Процес тестування

Немає універсального процесу тестування, але є деякі набори тестових активностей, які і визначають сам процес, а саме:

- Планування тестування - складається з активностей, які виділяють цілі тестування, підхід до цілей тестування у контексті (наприклад виділення методів тестування та формування графіку тестування для дотримання крайнього строку).
- Моніторинг та контроль тестування - моніторинг, фактично, - порівняння нинішнього результату роботи з тим, що у плані тестуванн, а контроль - міри, що приймаються аби досягти плану.
- Аналіз тестування - аналіз вирішує “що тестувати?” з точки зору вимірюваних критеріїв покриття. Основується на: базисі - специфікації потреб, інформації про проектування і реалізації, звітів аналізу ризиків; оцінка базису тестування та елементів тестування для виявлення дефектів різних типів (неоднозначність, пропуски, невідповідність, неточність, суперечливість); виділення властивостей та їх сукупностей для тестування тощо.
- Проектування тестів - складається з: проектування та пріоритизації тестових сценаріїв та наборів тестових сценаріїв; виділення необхідних тестових даних для підтримки текстових умов; проектування тестового середовища, виділення необхідної інфраструктури та інструментів.
- Реалізація тестів - в цілому - пріоритизація процедур тестування, створення автоматизованих тестів, створення наборів тестів із процедур тестування,

організація наборів тестів з розкладом виконання тестів таким чином, аби воно було ефективним, будування тестового середовища, перевірка правильності налаштувань, підготовка тестових даних ті правильне їх завантаження у середовище тестування.

- Виконання тестів - запис інформації про версію продута, що тестується; порівняння фактичних та очікуваних результатів; аналіз результатів; створення звіту про дефект, якщо такий є; запис результату тестувань;
- Завершення тестування - активності завершення тестування збирають дані із виконаних активностей тестування для досвіду та інших інформацій: закриття усіх звітів про дефектів, юзер сторей, тасок та іншого; звіт про тестування для зацікавлених; завершення та архівування тестового середовища, даних тощо.

1.3. Моделі життєвого циклу розробки ПЗ

Тестування існує для всіх етапів циклу розробки ПЗ, тому тестувальникам необхідно розуміти та знати розповсюджені моделі циклу розробки програмного забезпечення, на основі яких базується вибір тестових активностей. Незалежно від моделі життєвого циклу, тестові активності повинні проводитись на початку, дотримуючись принципу раннього тестування. Також існує декілька показників якісного тестування для будь-якої модели ЖЦ ПЗ: для кожної активності розробк - відповідна активність тестування; для кожного рівня тестування є щось характерне для цього рівня (моделі ЖЦ ПЗ); аналіз та проектування тестів для певного тестового рівня, починаються на відповідній стадії розробки; тестувальник беруть участь у всіх обговореннях, аби мати зможу уточнювати щось про продукт, бути у курсі змін та мати всю документацію, як тільки вона стає доступною.

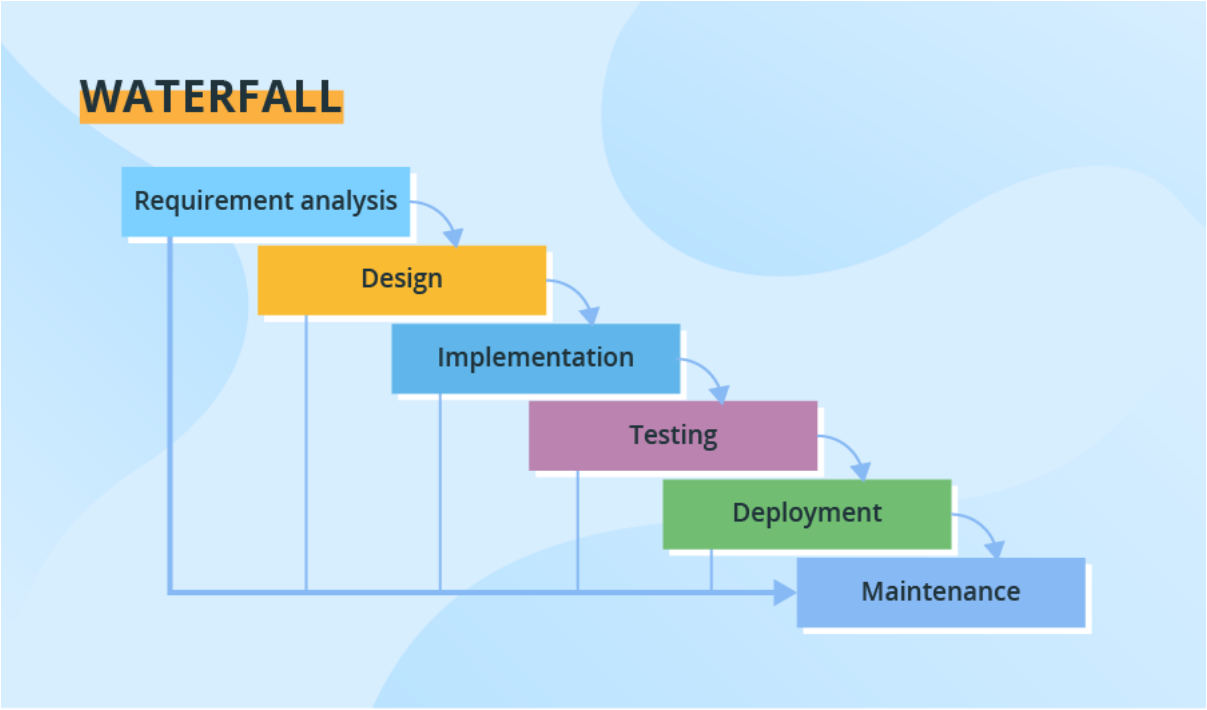

Рис. 1.1. Водоспадна модель життєвого циклу розробки ПЗ

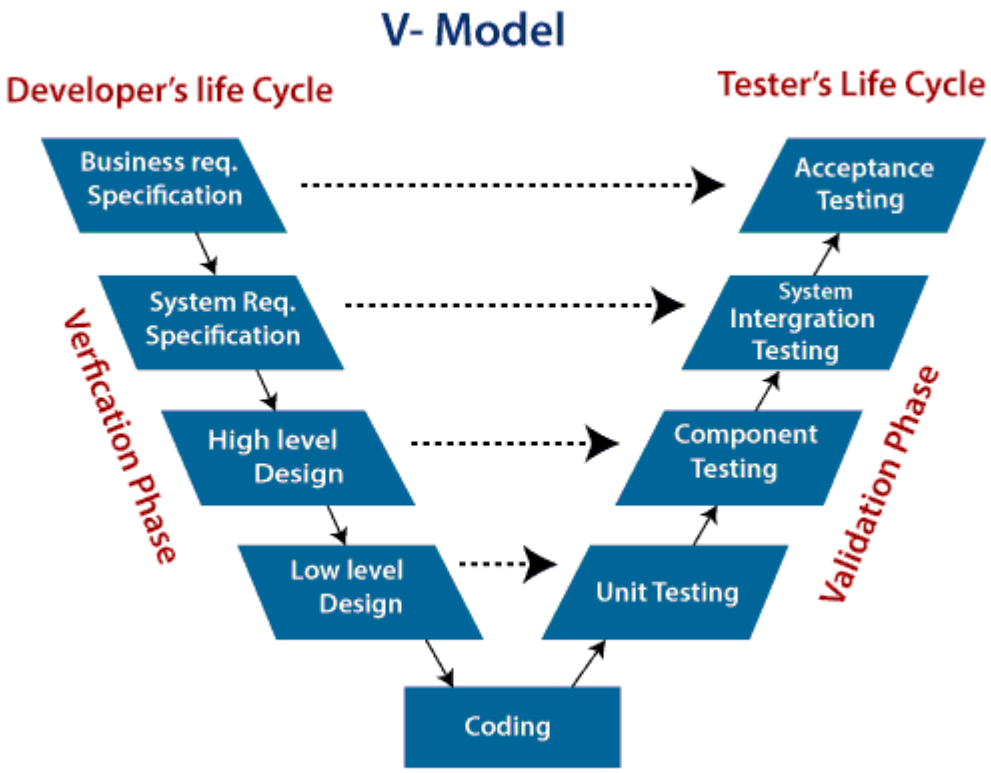

Рис.1.2. V-модель життєвого циклу розробки ПЗ

V-модель інтегрує тестовий процес протягом усієї розробки,та має у собі рівні тестування, які відповідають кожному рівню розробки, у той час, як у “водоспадній” моделі всі активності тестування виконуються тільки після закінчення активностей розробки.

1.4. Класифікація видів тестування.

Рівні тестування - тестові активності, які виконуються відповідно до кожного рівня розробки, починаючи з окремих модулів та компонентів, закінчуючі цілими системи та групами систем.

Кожна група буде характеризуватись такими ознаками: певні цілі, базис тестування, об’єкт тестування, типові дефекти та відмови, специфічні підходи та зони відповідальності існують такі види тестування:

- **Компонентне тестування**. Найчастіше виконується ізольовано від інших частин проекту, фокусуючись на окремих компонентах. . Особливості цього виду тестування:
    - Метою є:
        - зменшення ризику;
        - перевірка функціональних та нефункціональних частин на відповідність вимогам;
        - пересвідчення у якості компонентів;
        - виявлення дефектів ;
        - пересвідчення, що дефекти цього рівня не будуть пропущені на вищі рівні.
    - Базис тестування:
        - Код;
        - модель даних;
        - специфікації компонента.
    - Об’єкти тестування:
        - Код та структури даних;

        - моделі баз даних;
        - компоненти, модулі.
    - Типові дефекти та відмови:
        - Помилки у логіці коду;
        - неправильна робота функціональності, які одразу виправляються та фіксуються розробниками, що допомагає надалі зрозуміти першопричини дефектів.

Компонентне тестування дуже часто проводять розробники, тому зо потрібен доступ до коду.

- **Інтеграційне тестування.** Концентрується на роботі компонентів або систем між собою. Існують два рівня інтеграційного тестування: компонентне інтеграційне тестування та системне інтеграційне тестування.
  На цьому рівні, як і на компонентному, авто тести та регресійні тести допомагають пересвідчитись у тому, що нові зміни не зачепили вже існуючі та корректно працюючі. Особливості цього виду тестування:
    - Базис тестування:
        - Дизайн продукту;
        - діаграми послідовності;
        - специфікації інтерфейсів;
        - сценарії використання системи;
        - архітектури на рівні компонентів ;
        - робочі процеси;
        - специфікації, які описують зовнішні інтерфейси.
    - Об’єкти тестування:
        - підсистеми;
        - бази даних;
        - інфраструктура;
        - інтерфейси;
        - апі;
        - мікросервіси.

- Типові дефекти та відмови:
  - для компонентного інтеграційного тестування:
    - помилки у коді, неправильна послідовність звернення до інтерфейсів;
    - несумісність інтерфейсів;
    - збій зв’язку між компонентами;
    - неправильні позначення одиниць або кордонів даних, які передаються між компонентами.
  - для системного інтеграційного тестування:
    - помилки у коді, неправильна послідовність звернення до інтерфейсів;
    - несумісність інтерфейсів;
    - збій зв’язку між компонентами;
    - неправильні позначення одиниць або кордонів даних, які передаються між компонентами;
    - недотримання правил безпеки.

Ці два типи інтеграційного тестування повинні бути зосередженні на інтеграції. При компонентному інтеграційному тестуванні, яке зазвичай - робота розробників, при інтеграції компонентів, перевіряється їх сумісність, а не окрема робота, так само при системно інтеграційному, яке належить до роботи тестувальників, перевіряється робота систем між собою, а не окремо. Коли непротестованих, але вже “склеєних” між собою, дуже багато, виявити дефекти дуже важко, тому програмне забезпечення збирається та тестується “компонент за компонентом”.

- Системне тестування. Фокусується на поведінці та можливостях системи в цілому, переважно у тому ж самому або найбільш наближеним до реально експлуатаційного середовища середовищі.метою цього рівня є зменшення ризику, перевірка функціональних та нефункціональних частин на відповідність вимогам, пересвідчення у якості системи, виявлення дефектів, пересвідчення, що дефекти цього рівня не будуть пропущені на вищі рівні. Має такі особливості:

- Базисис тестування:
  - системні вимоги;
  - звіти про аналіз ризиків;
  - сценарії використання;
  - бізнес потреби та юзер сторі;
  - моделі поведінки системи;
  - діаграми станів;
  - системні та користувальницькі посібники.
- Об’єкти тестування:
  - застосунки;
  - програмні чи апаратні системи;
  - система, яка тестується;
  - операційна система;
  - конфігурації системи, даних.
- Типові дефекти та відмови:
  - некоректні вичислення ;
  - некоректна функціональна та нефункціональна робота системи;
  - некоректна передача даних всередині системи;
  - неможливість користуватися системою коректно у середовищі експлуатації.

Системне тестування зачасту проводиться незалежними тестувальниками, але бажано, щоб вони були залучені якомога раніше, аби не відбулися непорозуміння через відсутність документації, або її неправильність.

- Приймальне тестування:
  - Цілі приймального тестування:
    - показати впевненість у якості системи в цілому;
    - перевірити, що система закінчена та працюватиме так, як цього очікують;
    - перевірити функціональну та нефункціональну поведінку системи.

- Цей рівень тестування демонструє те, що система готова до експлуатації та працює коректно. Є декілька типів приймального тестування:
  - Користувацьке приймальне тестування зосереджено на перевірці придатності використання системи користувачами в реальному середовищі;
  - Експлуатаційне приймальне тестування проводиться зазвичай у імітований середі користування та приділяє увагу до експлуатаційних аспектів: резервне копіювання та відновлення; завантаження, видалення та оновлення; управління користувачами; перевірки на вразливість;
  - Контрактне та нормативне приймальне тестування базується на контрактах, указаних там критеріях приймання спеціалізованого ПО. А нормативне - на будь-яких нормативах, котрі повинні бути дотримані;
  - Альфа-тестування та бета-тестування. Альфа-тестування зазвичай проводиться на пізній стадії розробки продукту і включає імітацію реального використання продукту штатними розробниками або командою тестувальників, а бета - інтенсивне використання майже готової версії продукту з метою виявлення максимального числа помилок в його роботі для їх подальшого усунення перед остаточним виходом (релізом) продукту на ринок, до масового споживача. Бета-тестування є реально працюючою версію програми з повним функціоналом. І завдання бета-тестів – оцінити можливості і стабільність роботи програми з точки зору її майбутніх користувачів.
- Базис тестування:
  - бізнес процеси;
  - користувацькі та бізнес-потреби;
  - нормативи, юридичні контракти та стандарти;
  - системні потреби;

        - системна та користувацька документація;
        - звіти аналізу ризиків.
    - Об’єкти тестування:
        - система, яка тестується;
        - конфігурація системи;
        - бізнес процеси;
        - операційні та експлуатаційні процеси.
    - Типові дефекти та відмови:
        - системні робочі процеси не відповідають бізнес-потребам чи потребам користувачів;
        - бізнес потреби некоректно реалізовані.

Часто - відповідальність приймальної сторони, клієнту, бізнес-користувачів та інших зацікавлених сторін.

1.5. Типи тестування

Типи тестування - це сукупність активностей тестування, які направлені на тестування якихось характеристик системи, базуючись таких цілях, як оцінка функціональних та нефункціональних характеристик системи; оцінка правильності, повноти структури та архітектури; оцінка впливу змін, підтвердження того, що дефекти були виправлені.

Функціональне тестування - “що” система повинна робити - завершеність, коректність, доречність; базис може бути описаний у специфікації, юзер сторі, потребах користувача, у функціональній специфікації або взагалі не існувати. Виконуються на кожному рівні тестування, тому фокус тестування на кожному рівні - різний. Проектування тестів функціонального тестування потребує знань у специфічний предметній області продукту (наприклад у гемблінгу, слотах, казино іграх). Оскільки тестується поведінка системи , для отримання тест сценаріїв або умов використовуються техніка тестування чорної скриньки.

Метода чорної скриньки, як такого не існує, наразі ми маємо *техніка тестування* чорної скриньки. Ці техніки базуються на специфікації та документації (ніякого коду).

Еквівалентне розбивання ділить дані значень на групи (класи еквівалентності), які можна обробити, протестувати за однаковим алгоритмом. класи можуть бути визначені для будь яких даних, при необхідності може бути поділений на підкласи, а всі кожна одиниця даних може належати лише одному класу (позитивних значень - значення, які повинні бути прийняті або негативні значення - ті, які повинні бути відкинуто)

Аналіз граничних значень - цей метод - продовження методу еквівалентних розбиттів, але може бути використаним лише при визначених числових значеннях. Макс. та мін. значення класу - його кордони. Некоректна поведінка біль вірогідна на кордонах класу ніж у середині.

Тестування за допомогою таблиць альтернатив - цей метод дуже корисний при тестуванні потреб, які мають умови, які дають різні результати в залежності від комбінацій. У процесі розробки цієї таблиці, тестувальник виділяє умови (входи) та результати (виходи). пари умов та дій створюють строки таблиці (умови - зверху, дії - знизу), а кожний стовпчик має собою бізнес правило з унікальною комбінацією умов та дій, пов’язаних із цим правилом.

Нефункціональне тестування - “Наскільки добре працює система”. Тестування системи для оцінки зручності користування, продуктивності та безпеки. Якщо не виявити нефункціональність системи невчасно - це може бути загрозою для всього проекту, тому цей тип тестування повинен виконуватись на всіх рівнях/етапах проекту. Так само використовуються методи тестування чорного ящика, наприклад аналіз покордонних значень допоможе виділити пікови навантаження системи при тестуванні виробництва. Покриття та виконання нефункціональних тестів може вимагати спеціальних навичок та знань про слабкі сторони структури. яка тестується.

Тестування методом білої скриньки - базується на внутрішній структурі системи (код, архітектура, принципи роботи) або її реалізації. Повноту тестування виділяють за допомогою структурного покриття.

Тестування пов’язане зі змінами - коли система зазнає змін (виправлення дефекту, нова функціональність тощо), вона потребує тестування, аби бути впевненим, що система працює коректно, або дефекти були виправлені, не зачіпаючи весь інший функіонал.

Підтверджуюче тестування - після виправлення дефекту, ПО може бути протестовано тими самими тестовими сценаріями, які були завершені з помилкою через знайдений дефект, аби впевнитись, що дефект був виправлений.

Регресійне тестування - після змін, пов’язаних із внесенням нового функціоналу до програми, може бути так, що вони зачепили інший функціонал, такі ненавмисні побічні ефекти вимагають регресійного тестування, яке включає виконання тестів для їх виявлення. Проводятся на всіх рівнях тестування.

1.6. Ручне та автоматизоване тестування

Ручне тестування

Ручне тестування — це тип тестування програмного забезпечення, у якому тестові приклади виконуються тестувальником вручну без використання будь-яких автоматизованих інструментів. Метою тестування вручну є виявлення помилок, проблем і дефектів програмного забезпечення. Тестування програмного забезпечення вручну є найпримітивнішою технікою з усіх типів тестування, яка допомагає знайти критичні помилки в програмному додатку.

Будь-яка нова програма повинна бути протестована вручну, перш ніж її тестування можна буде автоматизувати. Ручне тестування програмного забезпечення вимагає більше зусиль, але необхідно для перевірки можливості автоматизації. Концепції ручного тестування не вимагають знання будь-якого інструменту тестування. Однією з основ тестування програмного забезпечення є «100% автоматизація неможлива». Це робить обов’язковим ручне тестування.

Ключова концепція ручного тестування полягає в тому, щоб гарантувати відсутність помилок у програмі та її роботу відповідно до визначених функціональних вимог.

Набори тестів або кейси створюються на етапі тестування та мають охоплювати 100% тестування

Він також гарантує, що повідомлені дефекти виправлено розробниками, а тестувальники провели повторне тестування виправлених дефектів

В основному це тестування перевіряє якість системи та надає замовнику продукт без помилок.

Автоматизоване тестування

Автоматичне тестування — це техніка тестування програмного забезпечення, яка виконується за допомогою спеціальних програмних засобів автоматизованого тестування для виконання набору тестових випадків. Навпаки, ручне тестування виконується людиною, яка сидить перед комп’ютером і ретельно виконує тестові кроки.

Програмне забезпечення для автоматизованого тестування також може вводити тестові дані в тестовану систему, порівнювати очікувані та фактичні результати та створювати детальні звіти про тестування. Автоматизація тестування програмного забезпечення вимагає значних вкладень грошей і ресурсів.

Послідовні цикли розробки вимагатимуть багаторазового виконання того самого набору тестів. Використовуючи інструмент автоматизації тестування, можна записати цей набір тестів і відтворити його за потреби. Після автоматизації набору тестів втручання людини не потрібне. Це покращило ROI автоматизації тестування. Метою автоматизації є зменшення кількості тестів, які потрібно запускати вручну, а не повна ліквідація ручного тестування.

1.7. Психологія тестування

Мислення інженера з тестування повинно відрізнятися від мислення розробника. У цілому, програмісти цілком здатні тестувати як код, який вони пишуть, так і функціональність системи, над якою вони працюють. Тому тестувальники, які мають певний ступінь незалежності, майже завжди здатні виявити недоліки і дефекти в системі більш ефективно, ніж програмісти.

Існує чотири рівні незалежності, від низького до високого:

1. Тестування програми розробляється і виконується людиною, яка її створила;
2. Тести розробляються та виконуються іншими людьми (наприклад, іншим розробником);
3. Тести розробляються іншими організаційними групами (наприклад, відділами тестування) або спеціалізованими тестувальниками (наприклад, тестувальниками продуктивності або безпеки);
4. Тести розробляються і виконуються фахівцями з інших організацій (наприклад, аутсорсинг або аудит).

Тестувальник повинен вміти знаходити всі недоліки в продукті, на відміну від програміста, головна мета якого - створити працюючий продукт. А для цього вони повинні спочатку зосередитися на тому, що може піти не так. Дослідження показують, що якщо людина, яка тестує програму, бачить, що вона працює правильно, вона знайде менше помилок, ніж той, хто переконаний, що дуже багато недоліків. Тому тестувальники повинні завжди пам'ятати, що "у програмного забезпечення є помилки".

1.8. Тестування ПЗ у автомобільних підприємствах

Тестування програмного забезпечення для автомобільного бізнесу є важливою складовою в процесі розробки програмного забезпечення для автомобілів. У зв'язку зі зростаючою складністю автомобільних систем, які включають програмне

забезпечення, тестування стає все більш важливим для забезпечення безпеки та якості автомобілів.

Основна мета тестування ПЗ для автомобільного бізнесу - забезпечення безпеки та стабільності автомобілів, а також визначення дефектів та помилок в програмному забезпеченні. Для цього проводяться різні види тестування, такі як функціональне, регресійне, тестування на навантаження, тестування на стійкість до помилок та інші.

Тестування ПЗ для автомобільного бізнесу може бути складним і вимагати спеціалізованих інструментів та технологій. Особлива увага приділяється тестуванню безпеки, так як безпека пасажирів та водіїв є головним пріоритетом. Важливо також забезпечити правильну взаємодію між програмним та апаратним забезпеченням автомобіля, щоб уникнути проблем при роботі системи.

Усі виявлені дефекти та помилки повинні бути відремонтовані та перевірені ще раз перед випуском автомобіля на ринок. Це може забезпечити високу якість та надійність продукту та підвищити рівень довіри споживачів до компанії. В цілому, тестування ПЗ для автомобільного бізнесу є невід'ємною складовою у процесі розробки автомобілів та впливає на безпеку та задоволення користувачів.

Використання автоматизованих тестів дозволяє прискорити процес тестування та знизити витрати на тестування. Крім того, автоматизовані тести забезпечують більш високу точність та надійність результатів, що допомагає забезпечити високу якість продукту.

У автомобільному бізнесі використовуються різні види автоматизованих тестів, такі як модульні, інтеграційні, системні, тести на навантаження та інші. Застосування автоматизованих тестів дозволяє здійснювати тестування автомобільної електроніки та програмного забезпечення на різних рівнях, включаючи симуляцію реальних умов експлуатації автомобілів.

Застосування автоматизованих тестів також дозволяє швидко виявляти та локалізувати помилки в програмному забезпеченні та виконувати регресійні тести при внесенні змін до програмного коду. Це забезпечує швидкий відгук на зміни та дозволяє швидко виправляти помилки, знижуючи час внесення змін та ризик випуску продукту з помилками.

Проте, автоматизоване тестування має свої обмеження та недоліки. Наприклад, автоматизовані тести можуть не виявити певні види помилок та дефектів, які можуть виявитися лише при ручному тестуванні. Крім того, автоматизоване тестування вимагає додаткових витрат на розробку тестів та на підтримку автоматизованих тестів після їх розробки.

Усе ж, автоматизоване тестування є важливою складовою тестування програмного забезпечення в автомобільному бізнесі і дозволяє підвищити якість та надійність продукту, знизити витрати на тестування та прискорити процес розробки продукту. Для успішного впровадження автоматизованого тестування в автомобільному бізнесі, необхідно враховувати особливості цієї галузі, такі як високі вимоги до якості, складність та різноманітність програмного забезпечення, технічні особливості автомобілів тощо.

Для ефективного використання автоматизованого тестування в автомобільному бізнесі, слід використовувати сучасні інструменти та технології, такі як фреймворки для автоматизованого тестування, контроль версій, інструменти для автоматизованої генерації тестів, віртуалізація, інструменти для навчання машинного навчання та інші.

Також важливо враховувати, що автоматизоване тестування має бути доповнене ручним тестуванням, оскільки ручне тестування дозволяє виявляти певні види помилок та дефектів, які можуть бути пропущені автоматизованими тестами. У загальному, автоматизоване тестування є важливою складовою процесу тестування в автомобільному бізнесі, яка дозволяє забезпечити високу якість та надійність програмного забезпечення та електроніки в автомобілях, зменшити час внесення змін та витрати на тестування.

1.9. Висновок до 1 розділу

Тестування програмного забезпечення є важливим етапом в процесі розробки програмного забезпечення.

Основною метою тестування є забезпечення того, що програмне забезпечення працює вірно та відповідає заданим вимогам. Це може зменшити кількість дефектів та помилок, що можуть виникнути в процесі роботи програми та, отже, забезпечити високу якість та надійність продукту.

Автоматизоване тестування дозволяє покращити ефективність тестування. Воно дозволяє проводити тести швидше та з меншими зусиллями, зменшує кількість помилок, пов'язаних з ручним введенням даних та забезпечує більш високу точність результатів. Крім того, автоматизоване тестування дозволяє покривати більшу кількість тестових сценаріїв, що забезпечує вищу якість продукту.

Автоматизоване тестування є важливим етапом в процесі розробки програмного забезпечення, оскільки дозволяє зменшити витрати на тестування, забезпечити високу якість та надійність продукту та забезпечити більш швидкий та ефективний процес розробки. Водночас, слід пам'ятати, що автоматизоване тестування не може повністю замінити ручне тестування та вимагає належної уваги та вміння використовувати інструменти автоматизованого тестування.

РОЗДІЛ 2
АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ АВТОМОБІЛЬНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Мета автоматизованого тестування

Автоматизоване тестування, як і будь яке, має за мету - отримання максимуму інформації про якість продукту, звичайно ж у рамках ресурсів проєкту, які є обмеженими. Але автоматизоване тестування насправді є найкращою оптимізацією тестової стратегії на проєкті. Спочатку - дорого, але потім, коли тестове середовище створено та налаштовано, тести написані тараняться кожного дня - це окупається.

Ось деякі цілі автоматизованого тестування.

Економія часу

Мануальне тестування, тобто ручне, забирає багато часу. При використанні автоматизованого тестування, усі задачі, такі як: налаштування передумов, запуск ПО, проходження кожного кроку з тест кейсу, звіряння з очікуваним результатом - передаються аппаратному комплексу, а це значить - швидше, якісніше та без зайвих зупинок на каву. Результати автоматично завантажуються до кожного тесту на будь-яку платфотму, яка налаштована для проєкту.

Звільнення від рутини

Програми та роботи роблять лише те, для чого вони створені, слідуючи точним інструкціям та алгоритмам, які в них закладені. Після повного налаштування програми, на яку всеодно треба витратити час, але у перспективі це нічого!, вона виконує свою роботу 24/7/365, таким чином тестувальники мають час на генерацію нових ідей, на тестування частин, які мають більше проблем, та звичайно на каву.

Надійність

Важливим фактором автоматизації є підвищення надійності. Як я вказала у пункті вище, програма, тобто тест, буде робити, а в нашому випадку - перевіряти, лише те, для чого вона була створена. Робитиме вона це завжди однаково, виставляючи пердумови, налаштовуючи середу, “мокуючи” різні стани продукту, з одним і тим самим результатом, якщо ніякі зміни не відбулись у прогрманому забезпеченні, що тестується.

2.2. Випадки для використання автоматизованого тестування

Якщо дивитися широко, то будь-яка тестувальна робота може бути автоматизована, як мінімум - більша частина. Питання лише в грошах да актуальності. Наприклад, тестувати графічні інтерфейси, іноді, набагато зручніше вручну: важлива не лише наявність елементів, а і їх логічна розташованість, кольори ітд. Але завжди можна налаштувати середовище, описати елементи на екрані, їх властивості, таким чином, щоб ми мали змогу тестувати їх наявність, логіку, розміщення автоматизовано, але чи буде це доречно?

Аби уникнути такої проблеми, як: час витрачений на створення, перевищив час використання та вартість перевищила вартість ручного тестування - на етапі підготовки автоматизації тестування, компанія підраховує економічну доречність автоматизації.

Однак для багатьох видів тестування автоматизація є економічно обґрунтованою.

Регресійне та димове тестування

Їх особливість — в багаторазовому повторі великої кількості тестів в однакових умовах.

Регресійне тестування полягає у перевірці того, що зміни в коді програмного забезпечення не зіпсують функціональність уже протестованих функцій.

Димове тестування ж має на меті перевірити, чи працює програмне забезпечення після внесення змін в код або налаштування серверів. Використання автоматизованих тестів дозволяє автоматично перевірити ці функції програмного забезпечення і отримати швидкі результати без необхідності витрачати багато часу на ручне тестування.

Unit-тестування

У модульному тестуванні без автоматизації не обійтись. Перевірити мільйони операторів, переходів та їх ланцюжки, комбінації на всіх можливих наборах даних мануально - дуже важко, а імітувати одночасне відвідування сайту тисячами або навіть десятками тисяч користувачів просто неможливо.

Accessibility-тестування

Тестування доступності є одним із специфічних видів тестування, які можуть дати корисну інформацію про якість програмного забезпечення, яка є необхідною в залежності від предметної галузі та умов застосування. Саме тестування доступності, визначає можливість використання додатку користувачами з особливими потребами. Ці перевірки є обов'язковими для більшості комерційних та державних сервісів у розвинених країнах, і можуть бути автоматизовані за допомогою безкоштовних інструментів, що дозволяє значно скоротити час тестування та підвищити його ефективність.

2.3. Розробка тестових сценаріїв та тест-кейсів

Розробка тестових сценаріїв та тест-кейсів є важливою частиною процесу тестування програмного забезпечення.

Тестовий сценарій - це опис послідовності кроків, які виконуються під час тестування для перевірки певної функціональності або сценарію використання програмного забезпечення. Тест-кейс - це конкретний тестовий сценарій, де вказані вхідні дані, кроки, очікувані результати та інші деталі, необхідні для його виконання.

Вони допомагають виявити дефекти або помилки в програмному забезпеченні. Шляхом систематичного виконання тестів можна перевірити, чи працює програма так, як очікується, та виявити можливі проблеми, Також дозволяють забезпечити покриття всіх функцій і можливих сценаріїв використання програмного забезпечення; сприяють встановленню стандартів для тестування; описують, яким чином програмне забезпечення повинно працювати і як перевірити його роботу. Якщо мається чіткий тестовий сценарій, його можна використовувати для написання автоматизованих тестів, що забезпечує швидке та ефективне тестування програмного забезпечення.

Усі ці фактори роблять розробку тестових сценаріїв та тест-кейсів необхідною складовою процесу тестування, оскільки вони допомагають забезпечити якість програмного забезпечення та вчасно виявляти та виправляти проблеми.

Розробки тестових сценаріїв та тест-кейсів

При написанні тестових сценаріїв слід дотримуватися певної структури та методології, щоб забезпечити їх чіткість та ефективність. Вони повинні мати чітку мету, що саме повинно бути протестоване, перевірене; чіткі передумови які необхідні для виконання тестового сценарію; кроки, які повинні бути виконані для проведення тестового сценарію, однозначні та конкретні; вхідні дані; очікувані результати, стан системи, аби розуміти, що буде “виводитись” при правильній роботі системи; будь-яка додаткова інформація (якщо потрібно), яка може бути корисною для виконання тесту або його розуміння.

Тест-кейси є детальними інструкціями для проведення окремих тестових сценаріїв. При їх написанні слід дотримуватися певної структури та методології. Кожен тест-кейс повинен мати: унікальний номер або ідентифікатор, який допомагає легко відстежувати його стан та результати; зрозуміла та коротка назва, що відображає його суть; передумови або початкові умови, необхідні для виконання тест-кейсу; послідовність кроків, які потрібно виконати для проведення тесту; вхідні дані; очікувані результати або стан системи після кожного кроку; перевірки, які потрібно здійснити після кожного кроку або в кінці тест-кейсу для підтвердження

правильності результатів; коментарі або додаткові відомості, які можуть бути корисними для зрозуміння тест-кейсу; залежності або обмеження, які пов'язані з цим тест-кейсом.

Не дивлячись на такі схожі описання, тестовий сценарій та тест кейс - різні речі.

Тестовий сценарій:

- Тестовий сценарій - це високорівневий опис послідовності подій або дій, які повинні бути виконані для перевірки певного аспекту або функціоналу програмного забезпечення. Описує загальний порядок дій, який виконує користувач або тестувальник для перевірки певного сценарію використання програми. Може бути більш загальним і орієнтованим на великі функціональні блоки, а не на докладні дії або перевірки на рівні окремих елементів інтерфейсу або функцій.

Тест-кейс - це детальна специфікація або інструкція, що визначає послідовність кроків, умови тестування, вхідні дані та очікувані результати для проведення окремого тесту. Він описує конкретні кроки, які повинні бути виконані, а також вказує, які дані використовувати та які результати очікувати. Тест-кейси зазвичай є більш детальними, конкретними і фокусуються на перевірці окремих елементів програмного забезпечення, таких як функції, інтерфейси, алгоритми тощо.

У процесі розробки тестових сценаріїв та тест-кейсів важливо також забезпечити документацію, яка би містила опис процедур тестування та рекомендації з їх виконання. Це дозволить забезпечити якість тестування та легше проводити майбутні оновлення додатку.

2.4. Розробка автоматизованих тестів

Розробка автоматизованих тестів є важливою складовою процесу тестування програмного забезпечення. Вона включає створення тестових сценаріїв і їх автоматизацію з використанням спеціальних інструментів і фреймворків. Автоматизація тестування допомагає покращити якість і ефективність тестування

шляхом автоматичного виконання тестів, що забезпечує швидкість, точність і повторюваність результатів.

Основні етапи розробки автоматизованих тестів включають:

Аналіз вимог, що є розумінням вимог до програмного забезпечення і визначення, які аспекти потребують автоматизованого тестування. Важливо виявити ключові функціональність, інтеграцію, продуктивність або інші аспекти, які вимагають автоматизованих тестів; вибір інструментарію, а саме інструментів роботи та фреймворків для розробки автоматизованих тестів. На ринку існує багато інструментів, таких як Selenium, Appium, JUnit, TestNG і інші, які надають потужні можливості для автоматизації тестування; планування тестів - визначення тестових сценаріїв, які ви хочете автоматизувати, а це включає визначення послідовності кроків, введення даних, очікуваних результатів та умов, що впливають на тестування; написання тестових скриптів - написання коду тестових скриптів, який виконує автоматизоване тестування, а це може включати в себе використання мов програмування, які підтримуються вибраним інструментом, для створення тестових сценаріїв.

Далі буде виконання тестів - запуск автоматизованих тестів для перевірки програмного забезпечення. Тестові скрипти виконуються автоматично з використанням інструментів автоматизації, які взаємодіють з програмним забезпеченням та перевіряють його роботу згідно з заданими тестовими сценаріями, а після виконання, ми отримуємо результатів, які ми повинні проаналізувати, тобто виявити помилки і проблеми у програмному забезпеченні, автоматизовані тестові звіти та журнали можуть надати важливу інформацію про стан програмного забезпечення і допомогти у виправленні помилок.

Так як тести вже написані, та запускаються постійно, ми їх постійно перевіряємо, та підтримуємо, тобто проводимо, оновлення при змінах в програмному забезпеченні або вимогах до тестування.

Результатом процесу розробки автоматизованих тестів є набір автоматизованих тестових сценаріїв або тестових скриптів, які можуть бути використані для

автоматичного виконання тестів програмного забезпечення. Ці автоматизовані тестові сценарії можуть включати різні типи тестів, такі як функціональні тести, інтеграційні тести, тести продуктивності, тести безпеки тощо. Кожен тестовий сценарій містить послідовність дій, введення даних та очікуваних результатів. Результатом успішної розробки автоматизованих тестів є збільшення продуктивності тестування, зниження ризиків помилок, прискорення випуску програмного забезпечення та поліпшення його якості. Автоматизовані тести також сприяють збільшенню ефективності розробників, забезпечуючи швидку зворотну інформацію про стан програми.

2.5. Запуск та виконання автоматизованих тестів

Після того, як тести вже написані, залишається їх запускати, отримувати результати та насолоджуватись життям.

Процес запуску автоматизованих тестів включає кілька етапів, що забезпечують виконання тестових сценаріїв та отримання результатів тестування. Для початку нам потрібно підготувати тестове середовище, тобто налаштувати тестові залежності, перевірити налаштування, пов’язані зі зв’язком тестового середовища та програмного забезпечення, тощо. Це є дуже важливим кроком для запуска тестів та отримання валідних результатів.

Після цього важливо нараштувати тестові параметри, конфігурацію, це може бути зв’язок з базою даних, вибір специфічного середовища, або тестових сценаріїв, обмеження деяких ресурсів або будь-які параметри, які пов’язані та впливають на хід тестування.

Запуск тестів. Буде проводитись запуск тих тетів, які підходять за налаштування, або тести, які були спеціально обрані для цього “рану”. Ну і наступне, але далеко не останнє - збір результатів. Під час та після виконання тестів важливо відслідковувати їх прогрес та збирати дані про результати. Це може включати запис пройдених кроків, виявлення помилок або проблем, збір метрик про продуктивність

або покриття тестами, а також збір загальних результатів тестування. Всі результати повинні бути проаналізовані, це може включати перевірку наявних помилок або несправностей, порівняння очікуваних результатів з фактичними, оцінку покриття тестами та визначення наступних кроків на основі зібраних даних.

Всі кроки, описані вище повинні бути задокументовані для проєкту, аби відслідковувати прогрес, знайдені проблеми, виправлені помилки, описані результати тестування та рекомендації щодо подальших кроків.

Навіть процес запуску тестів та збору результатів може бути автоматизованим сам по собі, де інструменти автоматизації тестування беруть на себе керування, це дозволяє забезпечити швидкість, точність та повторюваність виконання тестів. Але для нас тестувальників все ще залишається важливі кроки, такі як аналіз результатів та покращення/підтримка тестів.

Аналіз результатів тестування та виявлення проблем

Аналіз результатів автоматизованого тестування є важливою частиною процесу тестування програмного забезпечення. Його метою є виявлення проблем, які можуть бути присутніми в ПЗ після його тестування. Аналіз може включати перегляд інформації про виконані тестові сценарії, виявлення помилок та проблем з функціональністю ПЗ. Важливо детально описувати виявлені проблеми та помилки. Крім того, слід визначити причину виникнення цих проблем та знайти спосіб виправлення їх. Для кращої організації цього процесу можна використовувати спеціальні інструменти для відстеження проблем, такі як системи управління помилками.

Аналіз результатів тестування дозволяє виявити потенційні проблеми, які можуть виникнути у користувачів під час використання ПЗ. Це дозволяє розробникам виправити проблеми та поліпшити якість програмного забезпечення перед його випуском на ринок.

2.6. Підтримка та розвиток автоматизованих тестів

Підтримка та розвиток автоматизованих тестів - це процес, що включає у себе підтримку і постійне вдосконалення набору автоматизованих тестів, які використовуються для тестування програмного забезпечення.

Підтримка автоматизованих тестів передбачає їх актуалізацію та підтримання у справному стані протягом усього життєвого циклу програмного забезпечення. Це означає вирішення проблем, що виникають у тестових скриптах, вносити зміни при зміні вимог до програмного забезпечення або модифікації самого програмного продукту. При необхідності, також може виникати потреба в оновленні тестових скриптів, їх рефакторингу або розширенні функціоналу.

Розвиток автоматизованих тестів полягає у постійному вдосконаленні та розширенні набору тестів з метою покриття більшої частини функціональності програмного забезпечення. Це може включати створення нових тестових сценаріїв, додавання додаткових перевірок або розширення тестового покриття для виявлення більш широкого спектру помилок або проблем.

Підтримка та розвиток автоматизованих тестів вимагає систематичного підходу та дисципліни. Команда тестувальників повинна відстежувати проблеми, аналізувати результати, пропонувати та впроваджувати зміни в тестові скрипти та інструменти для поліпшення ефективності тестування. Це дозволяє підтримувати високу якість автоматизованих тестів, що в свою чергу позитивно впливає на якість і надійність програмного забезпечення, яке тестується.

Підтримка автоматизованих тестів

Підтримка автоматизованих тестів включає в себе набір дій, спрямованих на забезпечення ефективності та надійності автоматизованих тестів протягом усього життєвого циклу програмного забезпечення. Основна мета підтримки автоматизованих тестів - це забезпечити, що тестові скрипти залишаються актуальними, працездатними та продуктивними, а також виявляють якнайбільше проблем та помилок в програмному забезпеченні.

Основні аспекти підтримки автоматизованих тестів включають:

- Виявлення та виправлення проблем: Команда займається виявленням та усуненням проблем, що виникають у тестових скриптах. Це можуть бути помилки в коді, неправильне взаємодія з програмним забезпеченням, некоректні дані чи неправильне налаштування тестового середовища. Важливо швидко реагувати на такі проблеми і виправляти їх для збереження працездатності тестів.

- Оновлення тестових скриптів: З часом можуть змінюватися вимоги до програмного забезпечення, або можуть бути внесені зміни у саме програмне забезпечення. В такому випадку тестові скрипти потрібно оновити, щоб вони відповідали новим вимогам або враховували зміни у програмному забезпеченні. Це може включати редагування або переписування тестових скриптів, а також перевірку їх працездатності після оновлення.

- Розширення функціональності: Підтримка автоматизованих тестів також включає розширення функціональності тестових скриптів. Команда може додавати нові тестові сценарії, перевірки або параметризацію для покриття більш широкого спектру функціональності програмного забезпечення. Це дозволяє виявляти більше проблем та забезпечує більш повне тестування.

- Оптимізація та покращення ефективності: Підтримка автоматизованих тестів включає постійне покращення ефективності тестування. Це може включати оптимізацію тестових скриптів, використання більш швидких алгоритмів, покращення шаблонів тестування або використання нових технологій та інструментів. Метою є зменшення часу виконання тестів та збільшення їхньої точності та надійності.

- Моніторинг та звітність: Підтримка автоматизованих тестів включає постійний моніторинг результатів тестування та створення звітів про стан тестів. Команда аналізує результати тестування, виявляє тренди та проблеми, вносить зміни до тестових скриптів та процесу тестування для покращення якості тестування.

Підтримка автоматизованих тестів є важливою для забезпечення ефективності та надійності тестування програмного забезпечення. Вона допомагає забезпечити,

що автоматизовані тести залишаються актуальними, працездатними та продуктивними протягом усього життєвого циклу програмного забезпечення.

Розвиток автоматизованих тестів

Розвиток автоматизованих тестів - це процес постійного вдосконалення та розширення набору автоматизованих тестів для покриття різних аспектів тестування програмного забезпечення. Основна мета розвитку автоматизованих тестів полягає в покращенні якості тестування, забезпеченні виявлення більшої кількості помилок та проблем і забезпеченні більш широкого покриття функціональності програмного забезпечення.

- Аналіз і планування: На цьому етапі команда аналізує існуючі автоматизовані тести, визначає їх поточні обмеження та проблеми і планує подальші кроки для розвитку. Враховуються потреби бізнесу, нові функціональність та зміни в програмному забезпеченні.

- Розширення покриття: Команда розробляє нові тестові сценарії, що включають додаткові функціональність або випадки використання. Це дозволяє виявити нові проблеми та помилки, які можуть виникнути під час взаємодії з програмним забезпеченням.

- Покращення якості тестів: Команда оцінює ефективність і надійність існуючих автоматизованих тестів і вносить вдосконалення для поліпшення їх точності та надійності. Це може включати оптимізацію тестових сценаріїв, усунення помилок або оптимізацію використання ресурсів.

- Інтеграція з процесом розробки: Автоматизовані тести повинні бути інтегровані в процес розробки програмного забезпечення. Команда розробників та тестувальників співпрацюють для забезпечення автоматичного виконання тестів при кожній зміні в програмному забезпеченні.

- Технічне удосконалення: Команда розробників тестів постійно оновлює інструменти та фреймворки, які використовуються для автоматизованих тестів. Це дозволяє використовувати нові можливості та покращує ефективність тестування.

- Постійний моніторинг: Розвиток автоматизованих тестів вимагає постійного моніторингу тестових результатів та виявлення проблем. Команда аналізує результати тестування, виявляє пропущені проблеми і вносить зміни в тестові скрипти або методики тестування.

- Автоматизована звітність: Забезпечення автоматизованої звітності про результати тестування є важливою складовою розвитку автоматизованих тестів. Команда розробляє звіти, що надають інформацію про стан тестування, результати, виявлені проблеми та рекомендації для подальшого вдосконалення.

Розвиток автоматизованих тестів - це постійний процес, який допомагає забезпечити ефективне та надійне тестування програмного забезпечення, а також покращує продуктивність та якість розробки.

Розгляд технік покриття коду автоматизованими тестами

Техніка покриття коду автоматизованими тестами використовуються для забезпечення високої якості коду та забезпечення відповідності коду вимогам. Основні техніки покриття коду автоматизованими тестами включають:

1. Unit-тестування: тестування окремих функцій або методів, що дозволяє перевірити правильність їхньої реалізації.

2. Інтеграційне тестування: тестування взаємодії між компонентами системи та їхню взаємодію з іншими системами.

3. Функціональне тестування: тестування функціональності системи, яке включає тестування всіх функцій та операцій, що повинні бути доступні користувачам.

4. Навантажувальне тестування: тестування системи при великому обсязі роботи з метою перевірки її працездатності та стабільності.

5. Граничні тестування: тестування поведінки системи на межі допустимих значень, наприклад, максимальний розмір файлу, максимальна швидкість передачі даних тощо.

6. Тестування безпеки: тестування системи на вразливість до атак та вимоги до захисту даних.

7. Тестування доступності: тестування доступності системи для користувачів з різними особливостями, наприклад, з обмеженими можливостями або з різними типами пристроїв.

Комбінація цих технік дозволяє забезпечити максимальне покриття коду автоматизованими тестами та забезпечити високу якість програмного продукту.

2.7. Висновок до 2 розділу

Автоматизоване тестування є важливою частиною сучасного процесу розробки програмного забезпечення. Завдяки йому, розробники можуть швидко та ефективно виявляти та виправляти помилки в програмному забезпеченні, що забезпечує високу якість продукту та задоволеність користувачів.

Переваги автоматизованого тестування полягають у тому, що воно:

- Забезпечує швидке виявлення помилок в програмному забезпеченні;
- Дозволяє автоматизувати повторювані тести, що зменшує час, витрачений на тестування;
- Знижує ризик людських помилок;
- Дозволяє швидко перевіряти функціональність, продуктивність, безпеку та інші параметри програмного забезпечення;
- Забезпечує високу якість продукту та підвищує задоволеність користувачів.

Недоліки автоматизованого тестування полягають у тому, що воно:

- Вимагає значних зусиль та ресурсів на початковому етапі розробки;
- Не замінює повноцінного ручного тестування;
- Вимагає постійного оновлення тестових скриптів та інструментів.

Однак, автоматизоване тестування має й недоліки. Наприклад, воно може бути менш гнучким, ніж тестування вручну. Якщо тестувальник вручну може змінити тестові сценарії на ходу, то автоматизований тест, який виконується без допомоги людини, не може реагувати на непередбачувані ситуації. Крім того, автоматизоване

тестування може бути витратним з точки зору часу і ресурсів на його створення та підтримку.

Таким чином, автоматизоване тестування є актуальним і корисним інструментом для покращення якості програмного забезпечення та прискорення процесу розробки. І хоча воно не є бездоганним, все більше і більше компаній використовують автоматизоване тестування як складову частину своєї стратегії тестування.

РОЗДІЛ 3
АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У моїй роботі я розглядати тестування користувацького інтерфейсу. Користувацький інтерфейс (англ. User Interface, UI) - це спосіб взаємодії користувача з програмним та/або апаратним забезпеченням. В контексті автомобіля, користувацький інтерфейс охоплює всі елементи, що дозволяють водієві та пасажирам взаємодіяти з автомобільною системою, управляти функціями та отримувати необхідну інформацію.

Тестування користувацького інтерфейсу допомагає переконатися, що всі елементи працюють належним чином і задовольняють потреби користувачів.

3.1. Опис програмного забезпечення/користувацького інтерфейсу, що буде тестуватися

Оскільки я не маю доступа ні до одного автомобільного програмного забезпечення, я буду використовувати те, що ми маємо у відкритому доступі, як приклад Tesla Touchscreen.

На офіційному сайті можна побачити інструкцію до використання, яка вказує що, і як повинно працювати, її я використовуватиму як документацію.

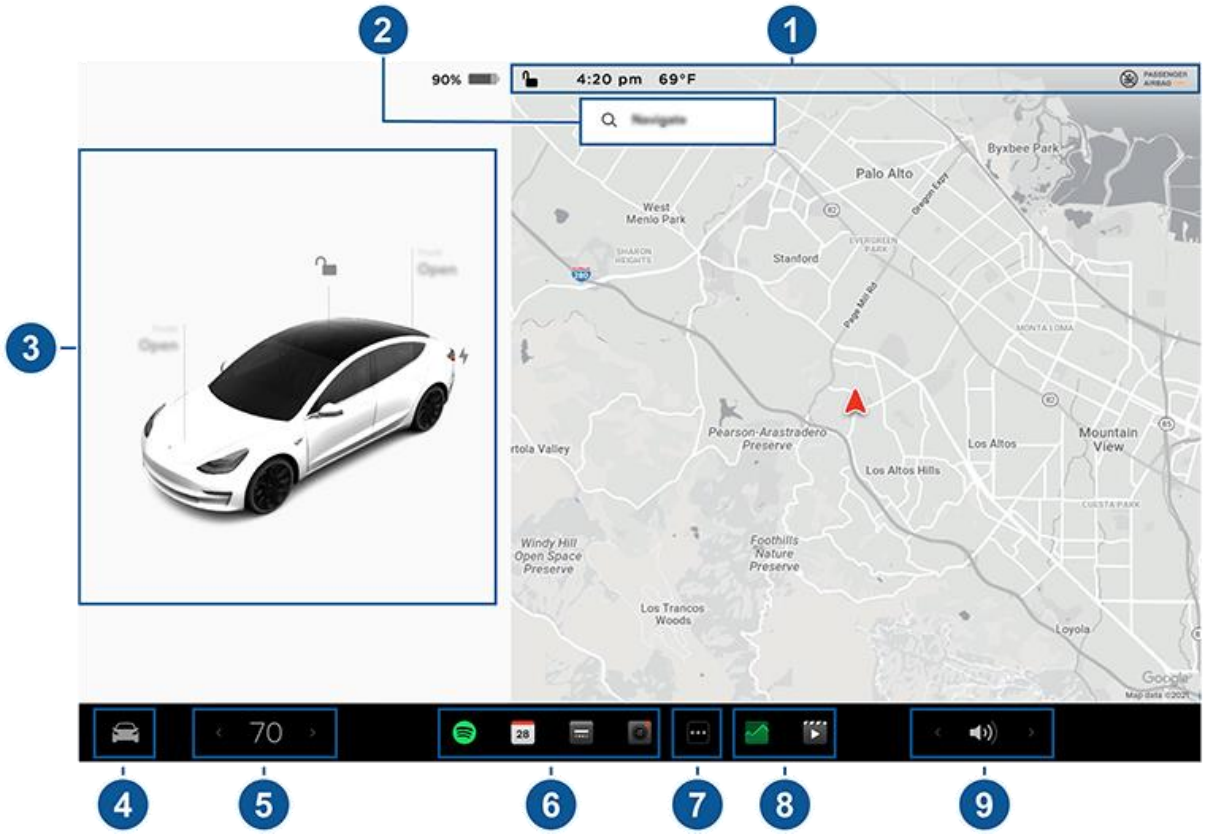

Рис.3.1. Tesla touchscreen – головний екран

Розглянемо розділ стану автомобіля.

Сенсорний екран відображає статус Model 3 у будь-який час. Те, що ви бачите, залежить від того, чи транспортний засіб:

- Припаркований;
- У стані водіння;
- Заряджається.

Коли Model 3 припарковано, на панелі стану відображається режим руху, приблизний запас ходу та вид автомобіля зверху з кнопками, до яких можна торкатися, щоб відкрити багажники та дверцята зарядного порту.

Коли ви натискаєте на гальмо, Model 3 вмикається, а індикатори коротко блимають. Якщо індикатор не відповідає поточній ситуації (наприклад, ремінь безпеки не пристебнутий), він повинен згаснути. Якщо індикатор не вмикається або не вимикається, зверніться до Tesla.

У нижній частині дисплея стану автомобіля також відображаються «картки» швидкого доступу для швидкого доступу до медіа, даних про тиск у шинах та інформації про поїздку. Гортайте картки ліворуч або праворуч, щоб змінити ярлики.

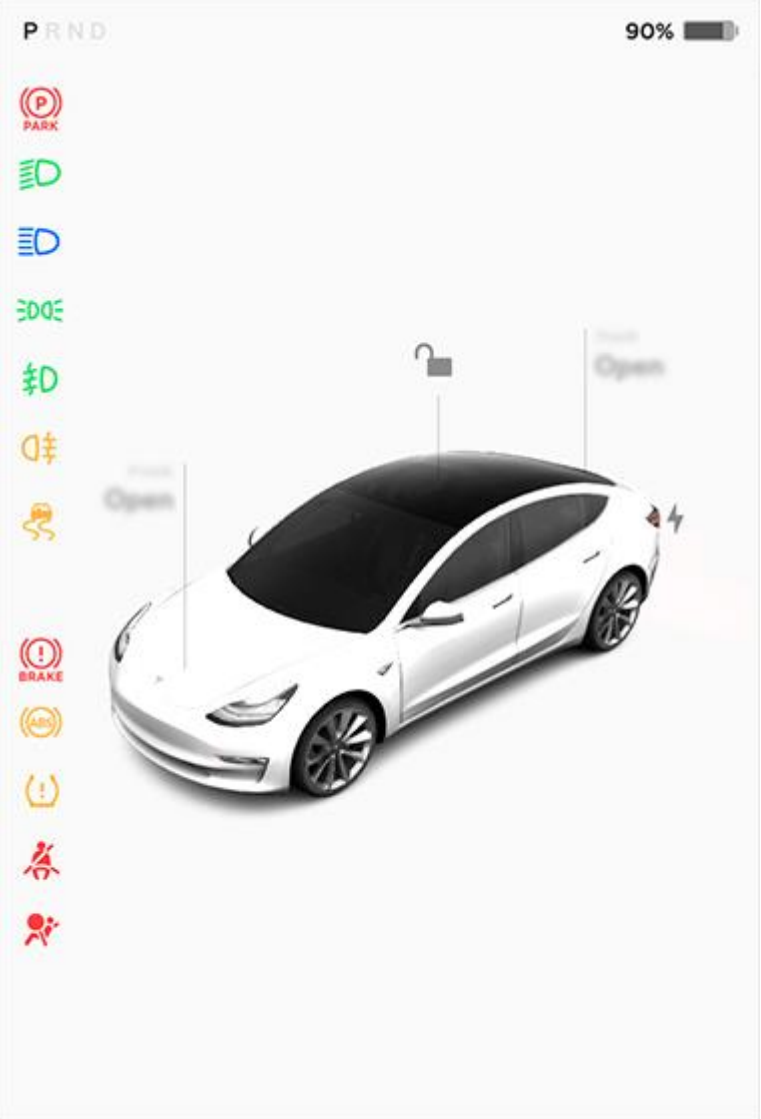

Рис. 3.2 Tesla touchscreen – екран стану автомобіля

Парковий стан автомобіля.

Світлові індикатори

Наступні індикатори світяться, щоб повідомити вам або попередити вас про певний стан або стан.

Рис.3.3. Значок несправність гальмівної системи

Виявлено несправність гальмівної системи або низький рівень гальмівної рідини. Негайно зверніться до Tesla.

Рис.3.4. Значок несправності стоянкового гальма

Виявлено несправність стоянкового гальма. Зв'яжіться з Tesla.

Рис.3.5. Значок несправності ABS

Виявлено несправність ABS (антиблокувальної гальмівної системи). Див. Гальмування та зупинка. Негайно зверніться до Tesla.

Рис.3.6. Значок несправності гальма стоянки

Виявлено несправність гальма стоянки. Зв'яжіться з Tesla. Див. Стоянкове гальмо.

Рис.3.7. Значок стоянкового гальма

Стоянкове гальмо вмикається вручну. Див. Стоянкове гальмо.

Рис.3.8. Значок попередження про тиск у шинах

Попередження про тиск у шинах. Тиск у шині виходить за допустимі межі. Якщо виявлено несправність системи контролю тиску в шинах (TPMS), індикатор блимає. У разі несправності TPMS зверніться до Tesla. Див. Догляд і технічне обслуговування шин.

Рис.3.9. Значок - не пристебнутий ремінь безпеки

Не пристебнутий ремінь безпеки для зайнятого сидіння. Див. Ремені безпеки.

Рис.3.10. Значок - безпека подушок безпеки

Безпека подушок безпеки. Якщо цей індикатор не блимає на короткий час, коли Model 3 готується до руху, або продовжує світитися, негайно зверніться до Tesla. Див. Попереджувальний індикатор подушки безпеки.

Рис.3.11. Значок - увімкнено передні протитуманні фари

Увімкнено передні протитуманні фари, якщо вони є. Дивіться Світло.

Рис.3.12. Значок увімкнених габаритних вогнів

Увімкнені габаритні вогні (габаритні вогні, задні ліхтарі та підсвічування номерного знака). Дивіться Світло.

Рис.3.13. Зачок увімкнених ближніх світло фар

Увімкнене ближнє світло фар.

3.2. Вибір мови програмування

Вибір мови програмування для автотестування залежить від кількох факторів і вимог вашого проекту. Вимоги, якими користуємось ми:

1. Легкість використання - найкраще обирати мову програмування, яка має не складний синтаксис та є потужною;
2. Наявність підтримки фреймворків - важливо, аби обрана мова мала підтримку відповідних фреймворків для автотестування;

3. Екосистема - також важливим фактором є достатня екосистема, яка включає бібліотеки та інструменти, необхідні для автотестування - це полегшує розробку та підтримку автотестів;
4. Крос-платформенність - якщо виконання автотестів потрібне на різних платформах, дуже і дуже важливо, аби мова програмування була крос-платформна - мала змогу працювати на різних операційних системах;
5. Інтеграція з іншими інструментами – якщо мова програмування легко інтегрується з іншими інструментами автоматизації та управління - це надає можливість створити реально потужне середовище для автоматизованого тестування.

Тож, спираючись на фактори вище, я обрала Python.

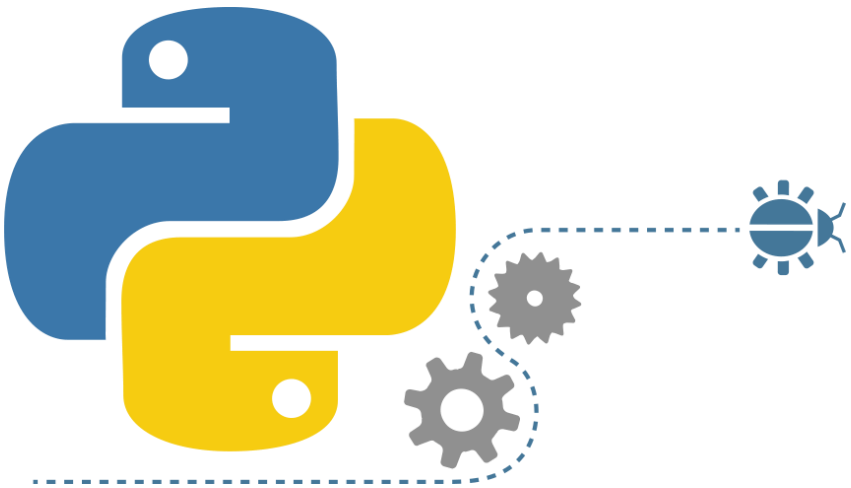

Рис.3.14. Логотип Python

Ця мова програмування відповідає вимогам, описаним вище і навіть більше. Проста та зрозуміла, легко читається та вивчається, тож навіть, якщо наймативуться робітники без знань цієї мови, вони легко її опанують! Багата екосистема, мільйон фреймворків на будь-який смак, python підтримується на різних операційних системах (Windows, macOS, Linux та всі їх версії.) Велика спільнота розробників надає нам багато документації, ресурсів, форумів, плагінів - будь-яка помилка,

задача буде вирішена! Python легко інтегрується з іншими інструментами, які використовуються для автоматизації тестування, такими як Jenkins, Git, JIRA - це дозволяє вам побудувати потужну та ефективну інфраструктуру автотестування.

Встановлення Python.

Завантажуємо Python для Windows з офіційного сайту https://www.python.org/downloads/windows/. Відкриваємо «Latest Python 3 Release - Python x.x.x». Залежно від версії вашого програмного забезпечення йдемо по фінструкції, що вказана там.

Для мене це **64-бітова** версія Windows, тож завантажуємо **Windows x86-64 executable installer**. ПІсля завантаження дистрибутива, запускаємо та йдемо за інструкцією.

Запускаємо додаток Python 3.

На першому екрані потрібно відмітити “Add Python 3.7 to PATH та натиснути “Install Now”.

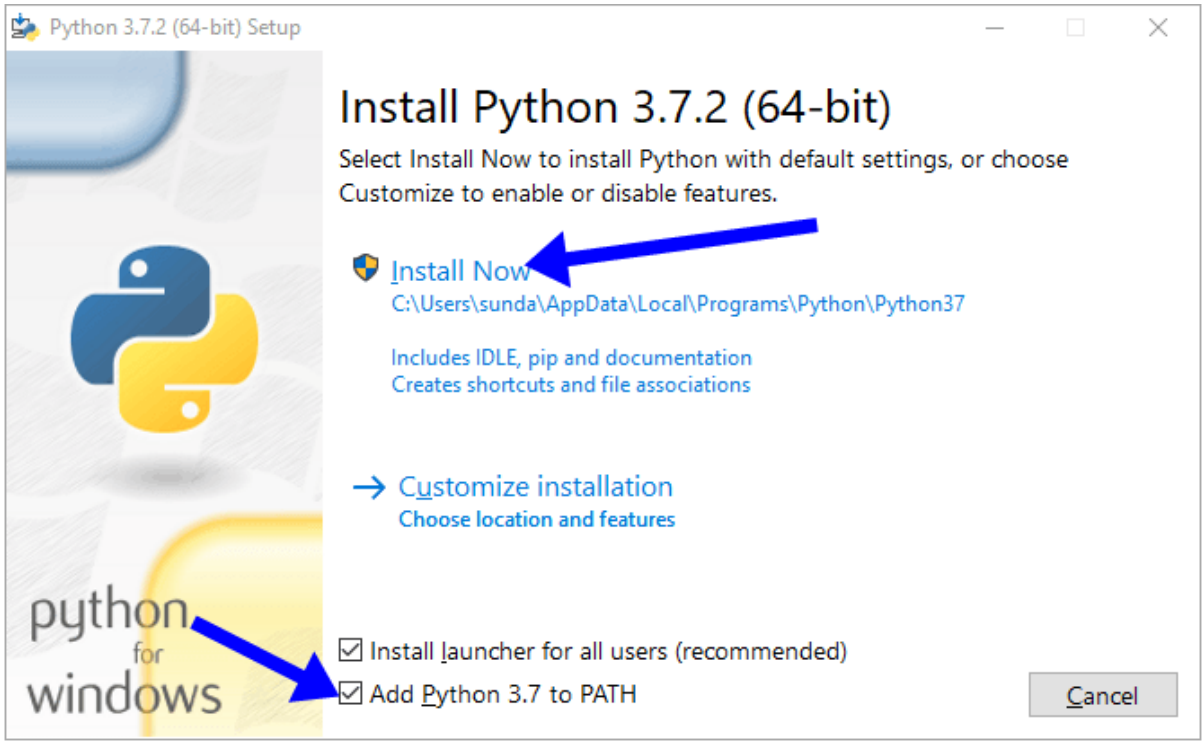

Рис.3.15. Екран встановлення Python

Після встановлення з'явиться можливість вимкнути обмеження довжини MAX_PATH. У системах Linux цих обмежень немає. Проігнорувавши цей пункт, ви можете зіткнутися з проблемою сумісності у майбутньому. Код, створений на Linux, не запуститься на Windows.

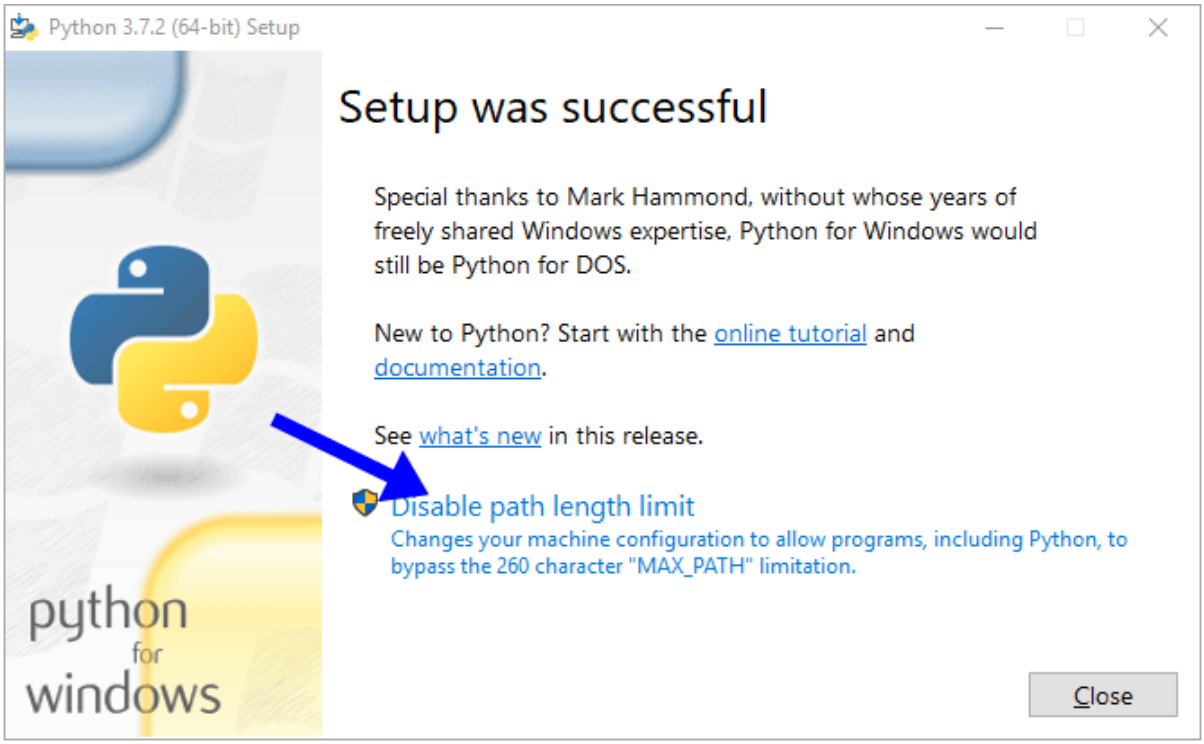

Рис.3.16. Повідомлення про успішне встановлення Python

Краще вимкнути цю опцію. Але якщо крос-платформенність не планується, то проблеми сумісності не буде. У такому випадку натискаємо “Close”.

Найкращий спосіб перевірити наявність python на комп'ютері (рис.3.17):

1. Запустити cmd.exe через диспетчер задач або пошук;
2. Ввести python;
3. В результате командная строка выведет версию python, которая установлена в системе.

```
C:\Users\katia>python
Python 3.10.11 (tags/v3.10.11:7d4cc5a, Apr  5 2023, 00:38:17) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

Рис.3.17. Перевірка наявності Python у терміналі

3.3. Необхідні інструменти для створення автотестів

Серед усіх інструментів для створення автотестів, я обрала PyTest.

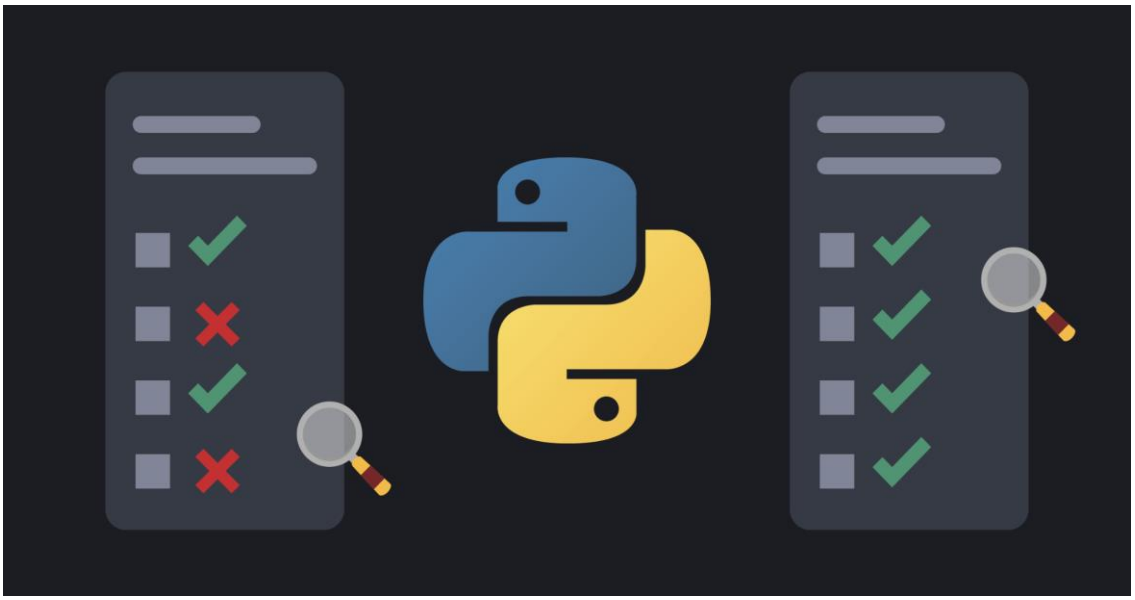

Рис.3.18. Python Pytest

Він є популярним і зручним вибором для тестування в програмній розробці. Pytest має простий та зрозумілий синтаксис, що сприяє швидкому навчанню і використанню фреймворку. Він надає чітку структуру для написання тестів і дозволяє швидко писати тести з мінімальними зусиллями? має широкий спектр розширень та плагінів, що дозволяє розширювати його функціональність та пристосовувати до специфічних потреб проекту. Завдяки цьому, у майбутньому є можливість використовувати різні додаткові інструменти і функції, які полегшують процес тестування. Pytest надає зручні можливості для параметризації тестів, що дозволяє виконувати один і той же тест з різними вхідними значеннями. Це дозволяє ефективно тестувати різні сценарії та варіації поведінки програмного забезпечення,

має потужну систему фікстур, яка дозволяє легко створювати та управляти передумовами для тестів. Фікстури допомагають уникнути дублювання коду і спрощують налаштування початкового стану для тестів. Pytest легко інтегрується з різними інструментами та фреймворками для тестування і програмної розробки, такими як Selenium, Django, Flask, PyCharm, Jenkins.

Для того, аби встановлення Pytest потрібно спочатку пересвідчитись. Що у вас встановлений пайтон.

Ми робитимемо це одразу в PyCharm. Встановлюємо його з офіційного сайту:

1. Перейдіть на сайт JetBrains https://www.jetbrains.com/pycharm/. PyCharm – це проект JetBrains;

Рис.3.19. JetBrains – головна сторінка

2. Натисніть кнопку «Download». Відбудеться перенаправлення на сторінку завантаження PyCharm;

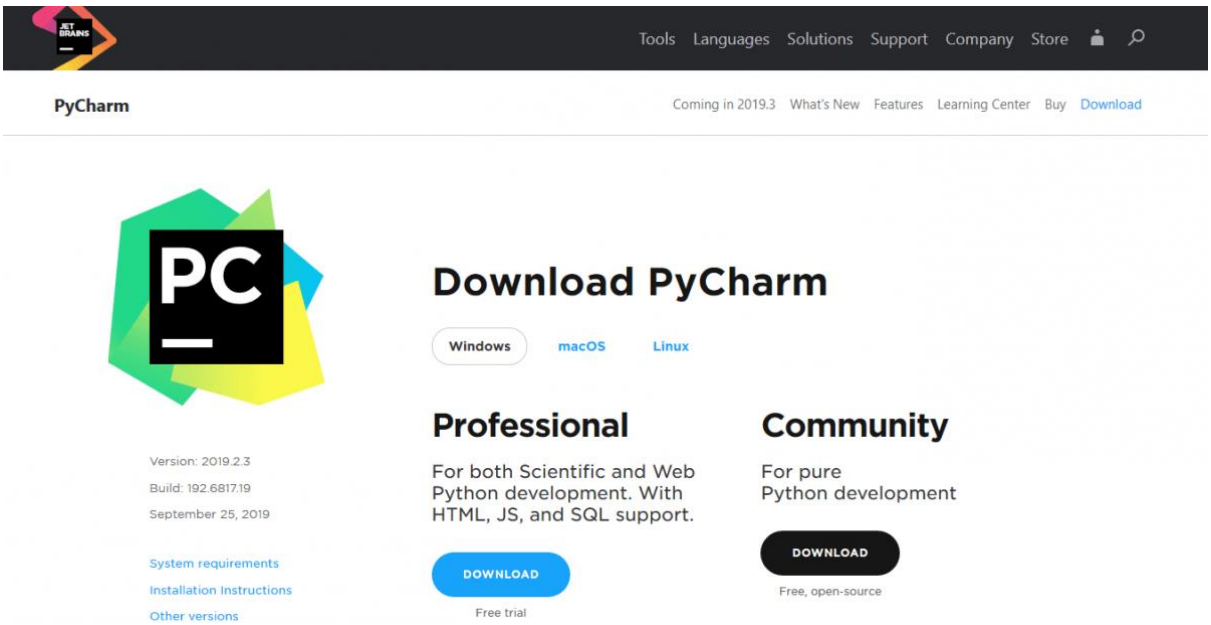

Рис.3.20. JetBrains – сторінка завантаження PyCharm

3. Далі необхідно вибрати версію PyCharm: Community чи Professional. Professional – це платна версія з повним набором функцій. Вона ідеально підходить для професійної розробки. Community безкоштовна. Нею можна скористатися завдяки набору базових можливостей. Після натискання на кнопку «Завантажити» завантаження розпочнеться автоматично.

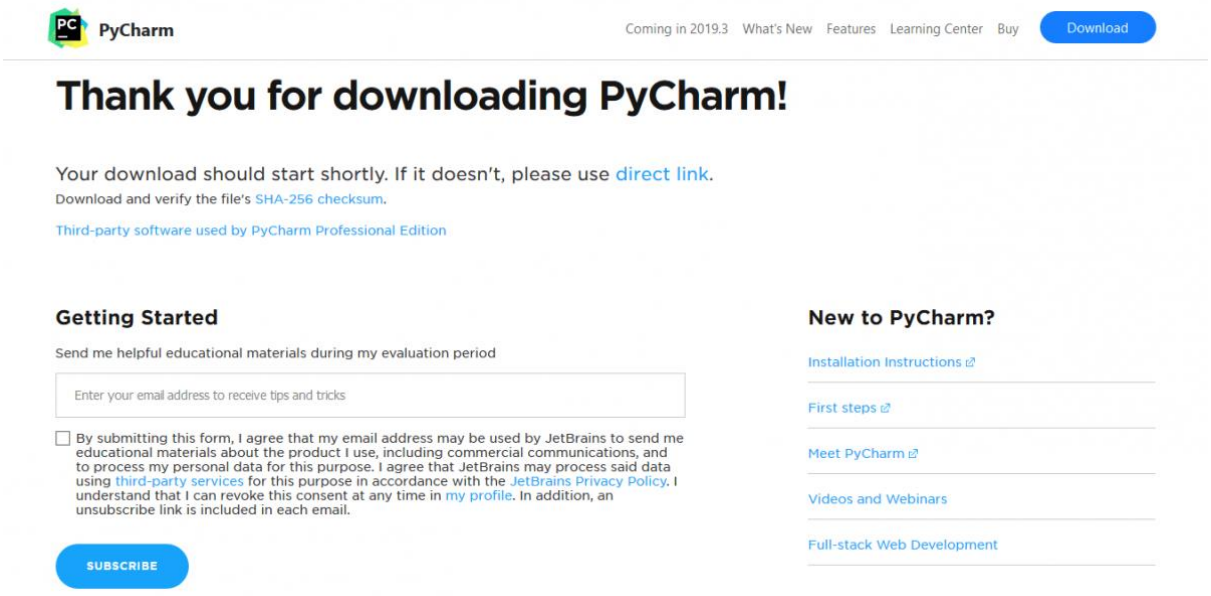

Рис.3.21. JetBrains - сторінка після завантаження

4. Тепер необхідно запустити інсталяцію (~ pycharm-community). Натисніть "Next".

Рис.3.22. Сет ап екран PyCharm

5. Залишіть стандартну папку установки. Це папка C:\Program files(x86)\JetBrains\PyCharm Community Edition 2019.2. Якщо обрано іншу, краще вказати саме цей напрямок. Натисніть "Next".
6. Поставте галочки навпроти обох пунктів, якщо потрібний ярлик для робочого столу. Натисніть "Next".

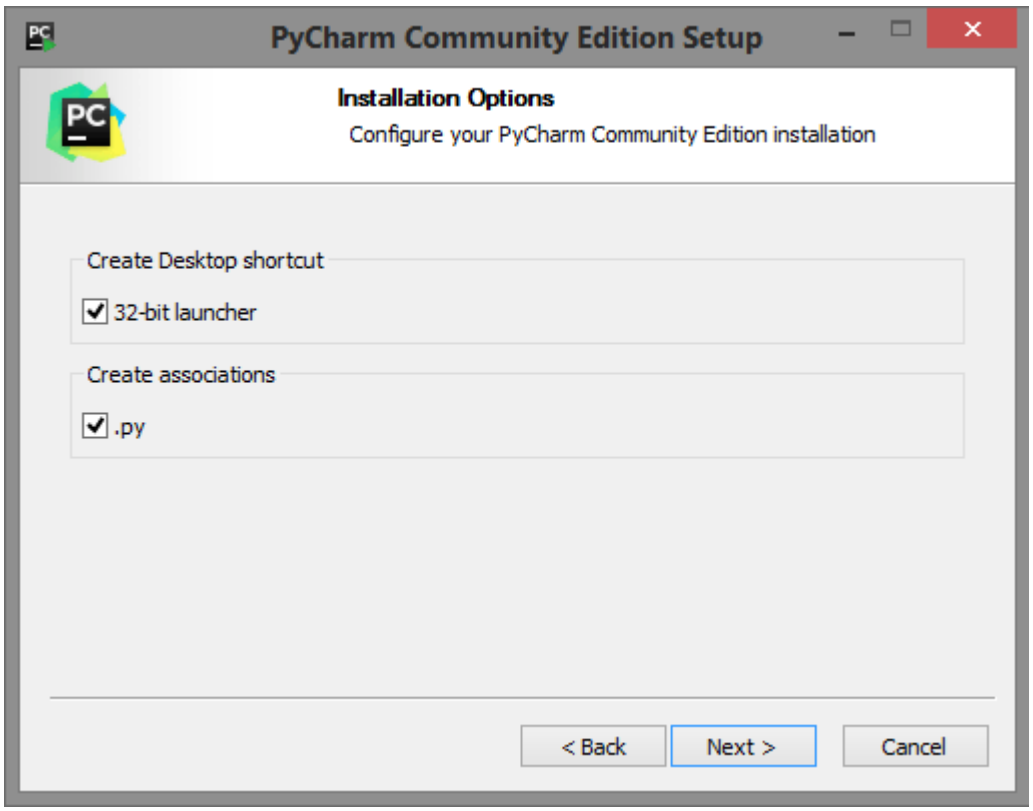

Рис.3.23. Додаткові опції для встановлення PyCharm

7. Далі інсталятор попросить вказати ім'я для відображення у стартовому меню. За замовчуванням це JetBrains. Натисніть "Install".
8. Встановлення розпочнеться автоматично. Дочекайтеся завершення.
9. Після завершення встановлення поставте галочку навпроти Run PyCharm Community Edition для запуску, а потім Finish.

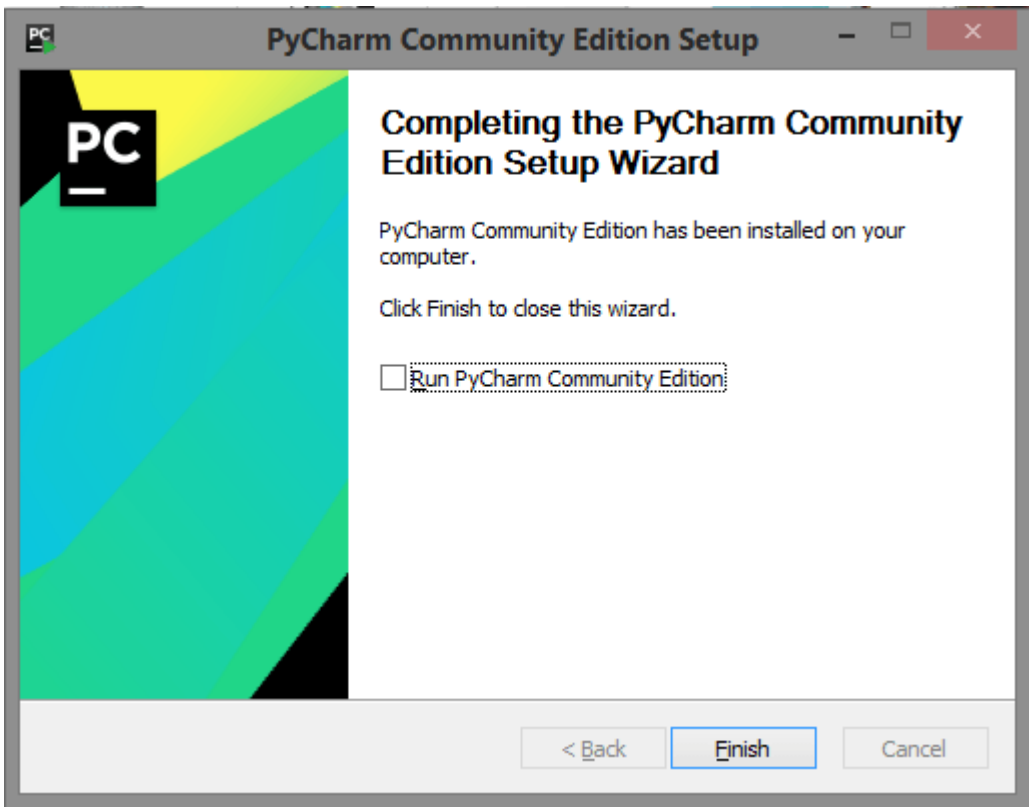

Рис.3.24. Додаткові опції при встановленні PyCharm

10. PyCharm запуститься. Вперше це займе більше часу.
11. Після вітального вікна натисніть Create New Project.
12. Виберіть каталог для збереження проекту. У другому полі потрібно вказати розташування інтерпретатора Python (який вже має бути встановлений). Зазвичай PyCharm знаходить його самостійно. Якщо не вдалося, потрібно вказати шлях до нього та натиснути Create.

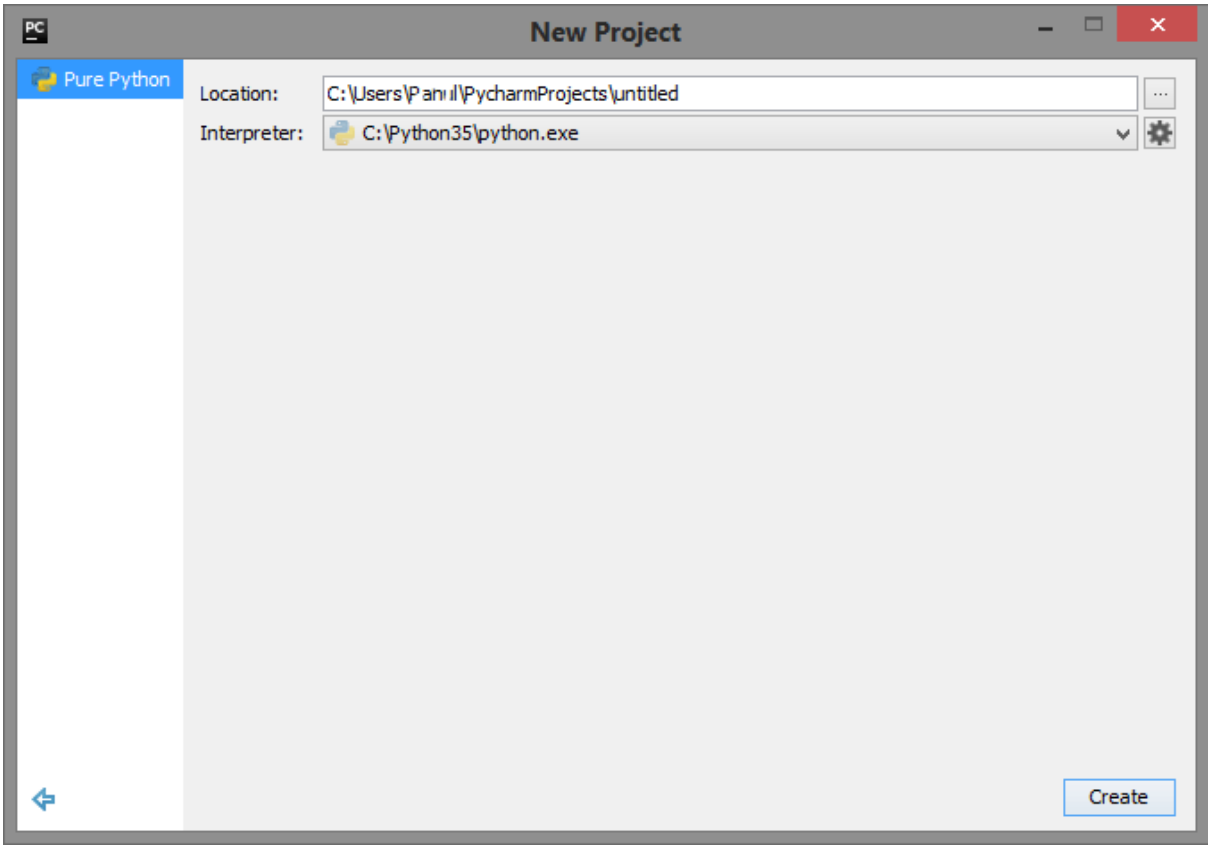

Рис.3.25. Вибір шляху при створенні нового проєкту у PyCharm

Після цього відкриваємо наш проєкт. Ви можете налаштувати інтерпретатор таким чином, аби для вас було комфортно. Відкриваємо термінал нашого проєкту. Та пишемо у терміналі команду “pip install -U pytest”, після цього завантаження почнеться.

```
(venv) PS C:\Users\katia\PycharmProjects\Diploma> pip install -U pytest
>> nv) PS C:\Users\katia\PycharmProjects\Diploma>
Requirement already satisfied: pytest in c:\users\katia\pycharmprojects\diploma\venv\lib\site-packages (7.3.1)
Requirement already satisfied: packaging in c:\users\katia\pycharmprojects\diploma\venv\lib\site-packages (from pytest) (23.1)
Requirement already satisfied: iniconfig in c:\users\katia\pycharmprojects\diploma\venv\lib\site-packages (from pytest) (2.0.0)
Requirement already satisfied: tomli>=1.0.0 in c:\users\katia\pycharmprojects\diploma\venv\lib\site-packages (from pytest) (2.0.1)
Requirement already satisfied: colorama in c:\users\katia\pycharmprojects\diploma\venv\lib\site-packages (from pytest) (0.4.6)
Requirement already satisfied: exceptiongroup>=1.0.0rc8 in c:\users\katia\pycharmprojects\diploma\venv\lib\site-packages (from pytest) (1.1.1)
Requirement already satisfied: pluggy<2.0,>=0.12 in c:\users\katia\pycharmprojects\diploma\venv\lib\site-packages (from pytest) (1.0.0)
```

Рис.3.26. Встановлення PyTest через термінал

Аби перевірити чи pytest встановився, набираємо терміналі команду “pytest –version”.

Рис.3.27. Перевірка версії Pytest через термінал

Тепер ми маємо встановлений pytest і вільно можемо ним користуватися.

3.4. Відображення процесу проходження документації

Після встановлення усіх необхідних додатків, ми можемо переходити до опису документації.

Перший етап полягає в складанні тест-плану і чек-листа, а також часткового прогону функціональних тестів.

Тест- план для Tesla Touchscreen Model 3:

1. Мета:

Метою даного тест плану є опис процесу тестування користувацького інтерфейсу програмного забезпечення Tesla Touchscreen Model 3. Опис дозволить мати більш чітке уявлення про планові роботи тестування.

2. Вихідні дані:

Документація, інструкція та наявні дизайни програмного забезпечення Tesla Touchscreen, https://www.tesla.com/ownersmanual/model3/en_jo/. Використання цього ПЗ дозволяє користувачу - водію, пасажиру - взаємодіяти з машиною через інтерфейс. Наприклад відчиняти/зачиняти двері, заводити/глушити мотор, дізнаватися данні про стан автомобіля тощо.

3. Мета тестування:

Метою тестування є перевірка якості програмного забезпечення, коректної роботи функціоналу.

4. Умови тестування:

Умови описані на офіційному сайті тесла. Я буду використовувати офіційну інструкцію для створення тест кейсів, та тест скриптів далі.

5. Стратегія процесу тестування:

Наведений нижче план є формальним. Для побудови реального плану, проєкт, усі вимоги від замовника, ресурси, інструменти - потрібні бути у повному доступі. Тестування буде проводитися вручну і за допомогою автотестів, методом «неформального» тестування (adhoc testing) з позиції кінцевого користувача.

6. Типи тестування: функціональне тестування.

Мета: виявлення функціональних помилок, невідповідностей ТЗ і очікуванням користувача шляхом реалізації стандартних, а також нетривіальних тестових сценаріїв.

7. Тестування дизайну:

Мета: виявлення недоліків дизайну, виявлення не відповідних/невалідних дизайнів (наприклад кнопка стоп зеленого кольору) тощо.

8. План робіт.

Таблиця 3.1

Тест план

Задача	Об’єм роботи	Дата початку	Дата закінчення
Складання тестплану	5 годин	04.05.2023	04.05.2023
Виконання тестування	12 годин	06.05.2023	06.05.2023
Аналіз тестування	9 годин	08.05.2023	08.05.2023

3.5. Автоматизоване тестування користувацького інтерфейсу

Оскільки у нас вже є створений проект і встановлений інструмент для автоматизованого тестування, ми можемо перейти до процесу написання тест скриптів, але для цього нам потрібен тест кейс.

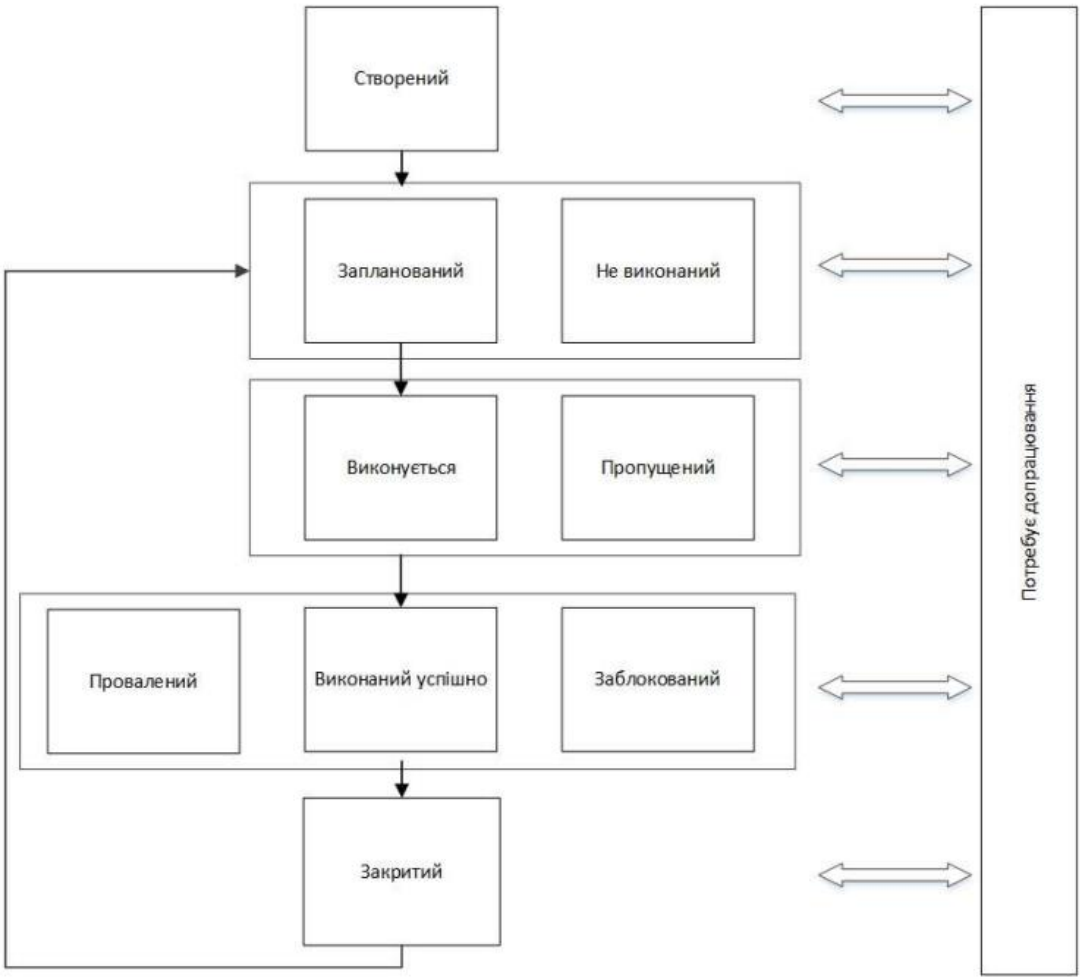

Рис.3.28. Життєвий цикл Тест кейсу

Для створення тест кейсу, я зареєструвалась на TestRail — це веб-інструмент для тестування програмного забезпечення, що розробляється. Використовується тестувальниками, розробниками та керівниками груп для управління, відстеження та організації заходів щодо тестування програмного забезпечення.

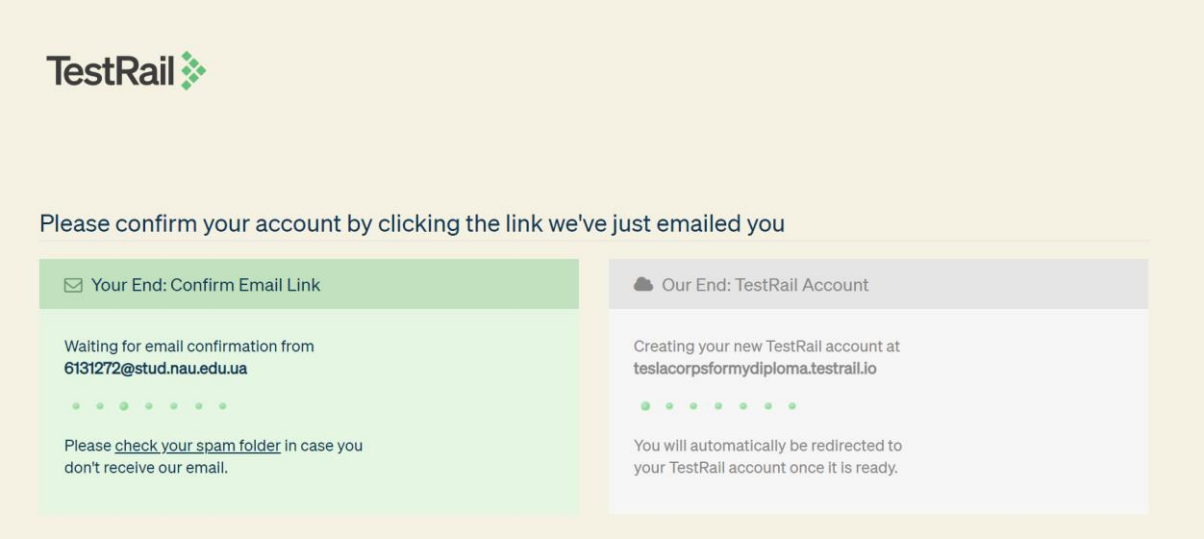

Рис.3.29. Створення профілю у TestRail

Створювати тест кейс ми будемо відповідно до інструкцій роботи Tesla touchscreen, а саме car status.

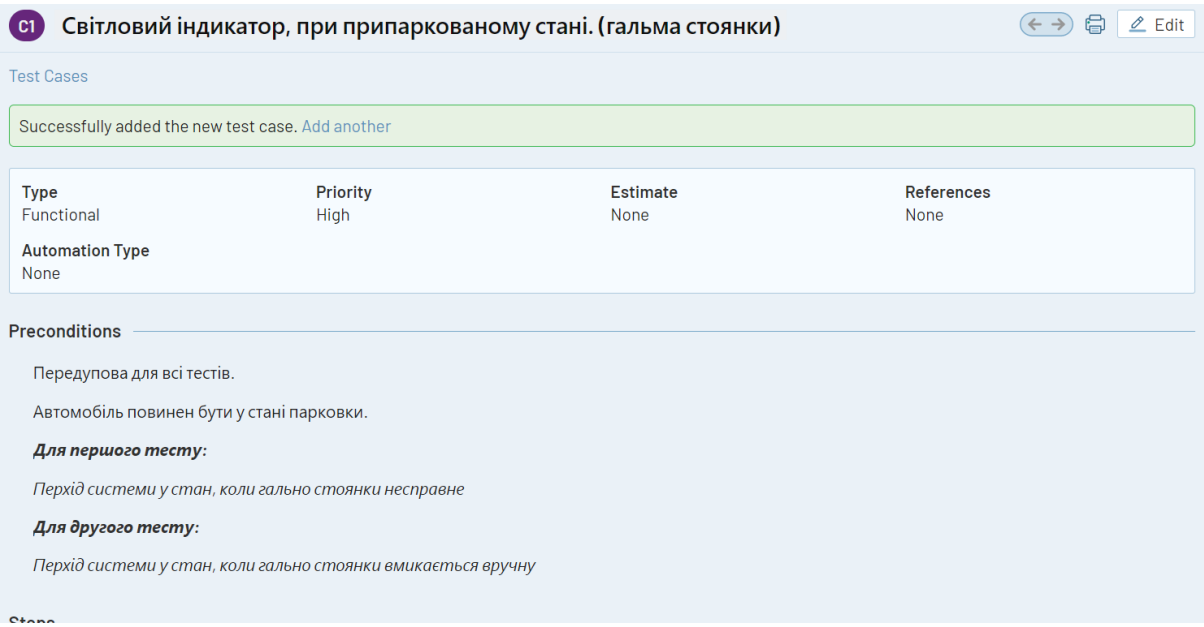

Рис.3.30. Тест кейс у TestRail #1

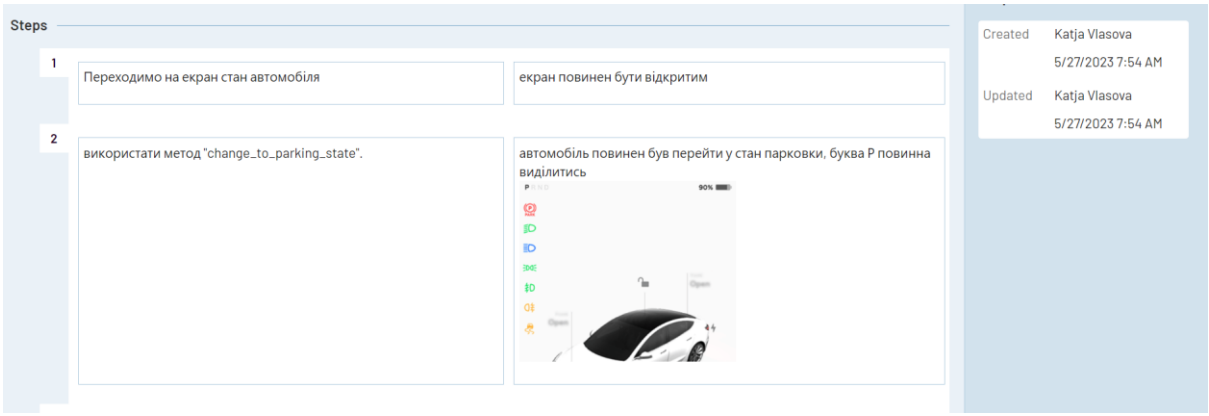

Рис.3.31. Тест кейс у TestRail #2

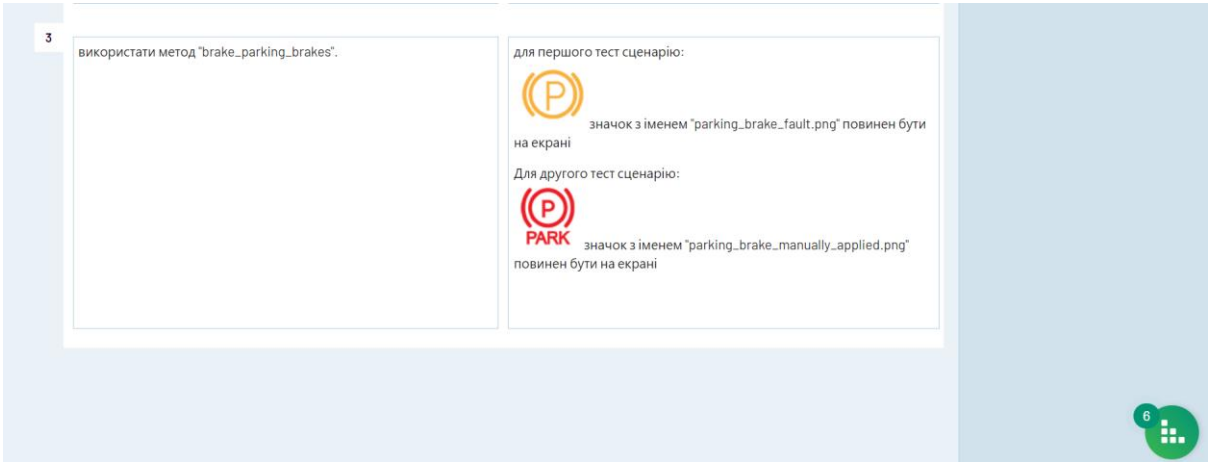

Рис.3.32. Тест кейс у TestRail #3

Тест кейс створено. Тепер можна перейти до його автоматизування.

Через те, що мій автоматизований тест - формальний, я створюю окремий фай, в якому я описую екран статусу автомобіля, вношу свої дані, які я очікую. Якби я мала реальне забезпечення для тестування, всі дані були б взяті з реального користувацького інтерфейсу через інструмент такий як Qt.

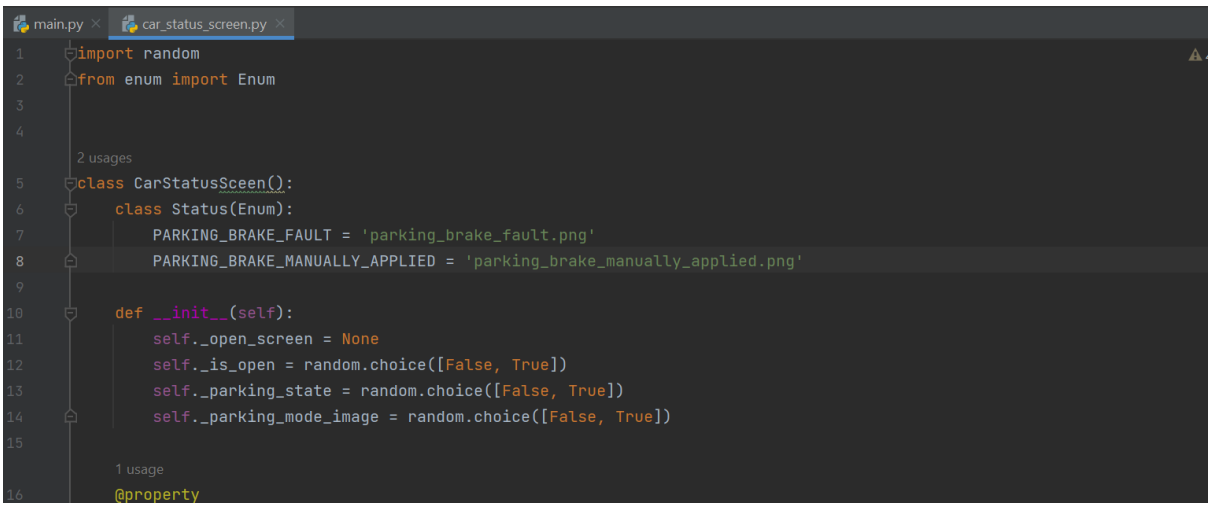

Рис 3.33. Код у PyCharm

Створюю методи, які збирають інформацію про стан об’єктів (чи є об'єкт виділеним, збільшеним, чи змінений колір, чи екран відкритий ітд.).

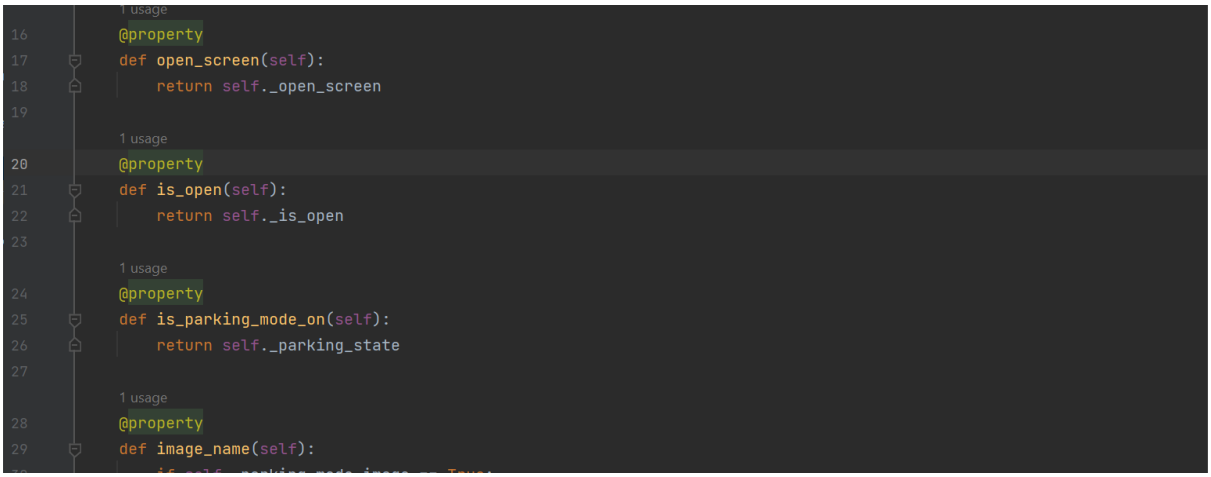

Рис.3.34. Код у PyCharm

Рис.3.35. Код у PyCharm

Також створюю клас, у якому я буду зберігати методи для “мокування” системи. Так як наше середовище знаходиться на комп’ютері, ми можемо лише імітувати стан автомобіля, та приборів, які у ньому.

створюю ще один документ “головний”, у якому я писатиму тести.

Створюю тест. Та прописую кожний крок. Відкриваю екран - перевію, чи він відкритий за допомогою метода, який було створено у класі.

Змінюю стан автомобіля - перевіряю відповідні об’єкти чи змінились вони, чи ні.

“Ламаю” тормоз - перевіряю чи змінився стан.

```
def test_car_status():
    car_status = car_status_screen.CarStatusSceen()

    # Переходимо на екран стан автомобіля
    car_status.open_screen
    assert car_status.is_open, 'Car status screen should be open'

    # використати метод "change_to_parking_state"
    method = car_status_screen.CarStatusSceen().Methods()
    method.change_to_parking_state()

    assert car_status.is_parking_mode_on, 'Parking mode should be turn on'

    # використати метод "brake_parking_brakes".
    method.brake_parking_brakes()
    assert car_status.image_name == "parking_brake_fault.png", 'wrong image or parking brakes'
```

Рис.3.36. Код у PyCharm

Запускаю тест - отримую результат. Якщо результат позитивний, це означає, що у ПЗ немає помилок.

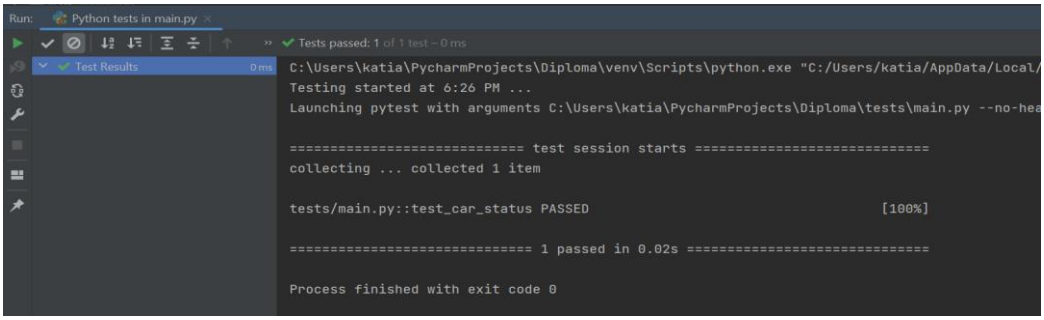

Рис.3.37. Результати тесту у PyCharm – позитивний

Якщо результат негативний, ми маємо червоний тест, який показує в чому причина. В моєму випадку - три негативних теста, можемо бачити чому вони “впали”, на якому місці, та що, мало б бути.

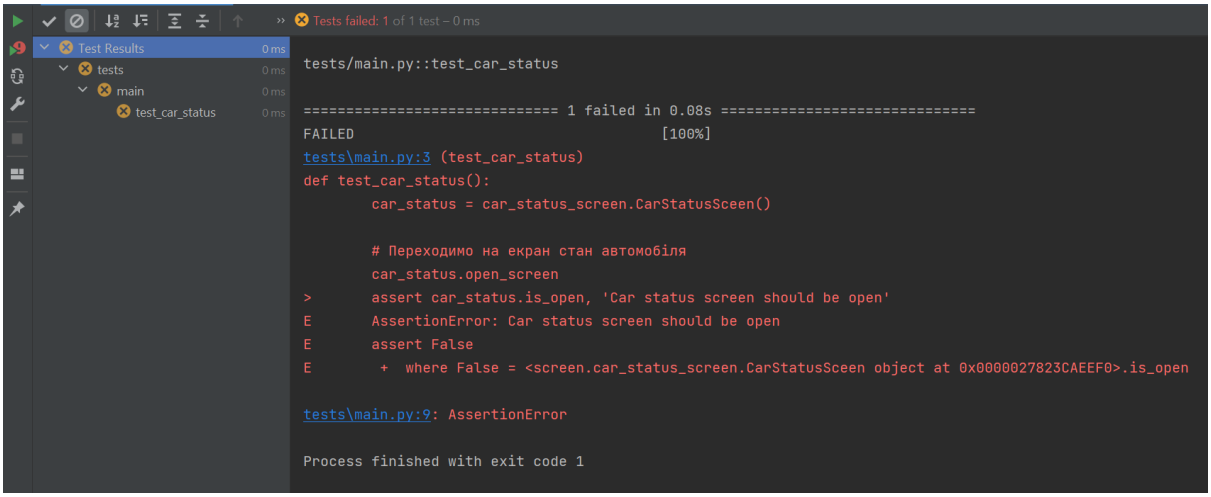

Рис.3.38. Результати тесту у PyCharm – негативний

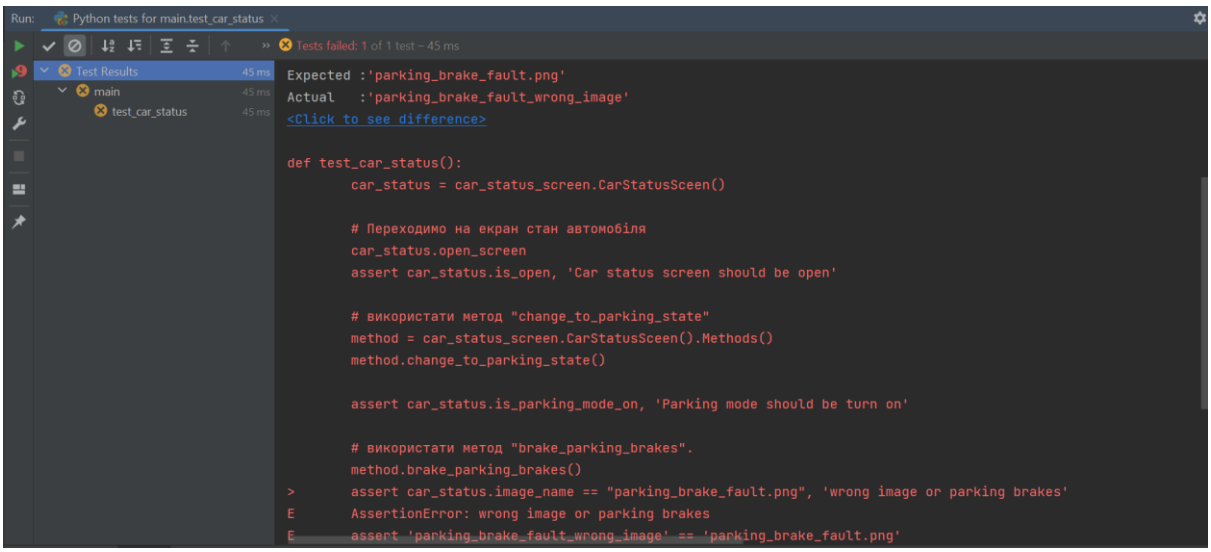

Рис.3.39. Результати тесту у PyCharm – негативний

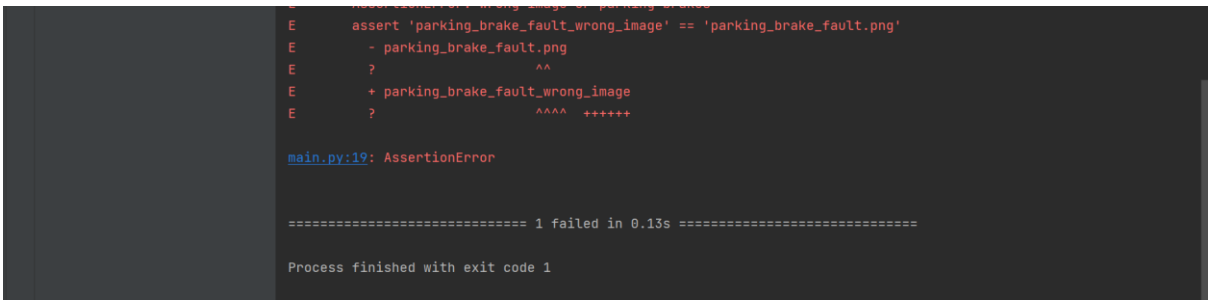

Рис.3.40. Результати тесту у PyCharm – негативний

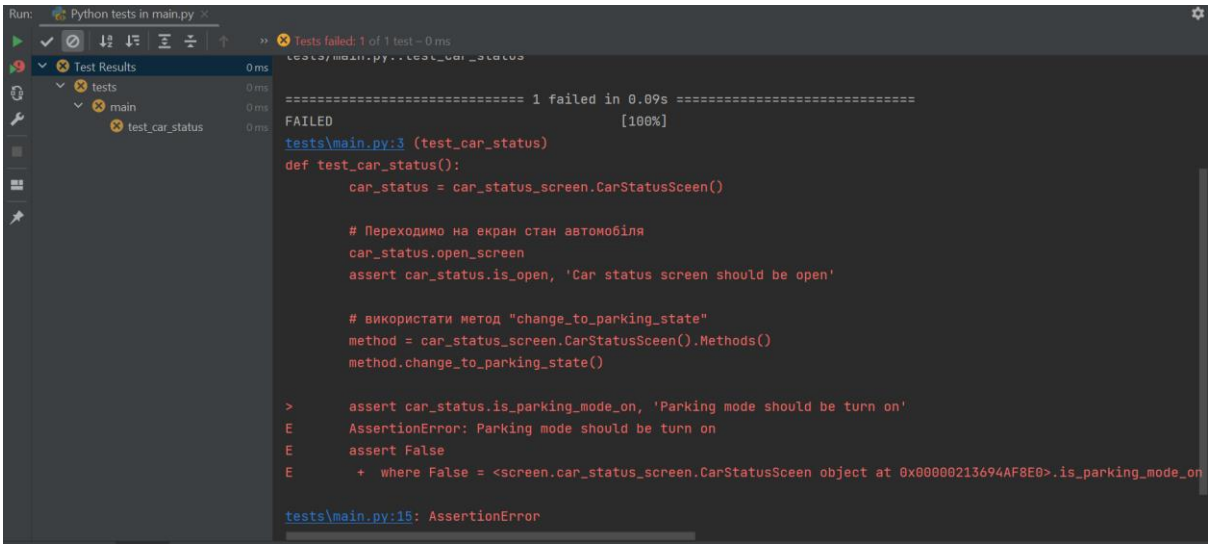

Рис.3.41. Результати тесту у PyCharm – негативний

3.6. Висновок до 3 розділу

У цьому розділі я описала та показала практичну частину використання тестування як мануального так і автоматизованого на прикладі Tesla car. Процес тестування був продемонстрований, що дозволило оцінити ефективність тестування та ідентифікувати потенційні проблеми. Інформація, отримана в процесі тестування, може бути використана для виправлення помилок, поліпшення функціональності та забезпечення якості продукту. Загалом, розробка тест-плану, тест-кейсів, тест-скриптів та продемонстрований процес тестування є важливими кроками у забезпеченні якості програмного продукту. Ці кроки сприяють систематичному підходу до тестування, покращують ефективність та зменшують ризик помилок, що можуть з'явитися під час розробки програмного забезпечення.

ВИСНОВКИ

Тестування будь-якого програмного забезпечення, зокрема автомобільного, є надзвичайно важливою частиною процесу розробки з декількох вагомих причин.

В першу чергу, автомобільне програмне забезпечення відповідає за безпеку, функціональність та надійність автомобілів. Тестування дозволяє перевірити, як програма взаємодіє з різними системами транспортного засобу та реагує на різні умови використання. Це важливо для забезпечення безпеки пасажирів та користувачів дороги, а також для попередження можливих аварій та несправностей. Крім того, автоматизоване тестування відіграє ключову роль у розробці автомобільного програмного забезпечення. Воно дозволяє автоматизувати виконання тест-кейсів, тест-скриптів та інших тестових процедур. Це сприяє зниженню “людських” помилок, покращенню ефективності тестування та скороченню часу, необхідного для виконання тестових завдань. Автоматизоване тестування також дозволяє виявляти помилки, що можуть бути важко виявити вручну, та забезпечує швидкий і повторюваний процес тестування. Також тестування автомобільного програмного забезпечення включає в себе валідацію та верифікацію відповідно до стандартів та нормативів безпеки автомобільної промисловості. Це включає перевірку дотримання вимог функціональності, стійкості до помилок, стандартів безпеки, а також виконання незалежних тестів безпеки.

Отже, тестування будь-якого програмного забезпечення, зокрема автомобільного, відіграє вирішальну роль у забезпеченні безпеки, функціональності та надійності. Автоматизоване тестування вносить вагомий внесок у розробку, допомагаючи виявляти помилки, знижувати ризик людських помилок та забезпечувати ефективність процесу тестування.

Таким чином, в результаті виконання роботи було розроблено програмний застосунок для тестування програмного забезпечення автомобільного підприємства, та вирішені такі задачі, як :

- аналіз тестування програмного забезпечення;
- створення своєї системи тестування для автомобільного підприємства;
- підвищено швидкість тестового процесу за допомогою автоматизації тестування;
- знижено вартість тестового процесу.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ISTQB Foundation level – [Електронний ресурс] – Режим доступу: https://www.istqb.org/certifications/certified-tester-foundation-level (дата звернення 10.04.2023 р.). – Назва з екрана.

2. Complete Guide to Test Automation – [Електронний ресурс] – Режим доступу: Complete Guide to Test Automation (дата звернення 15.04.2023 р.).

3. TestRail tutorial – [Електронний ресурс] – Режим доступу: https://www.tutorialspoint.com/testrail/testrail_tutorial.pdf (дата звернення 05.05.2023 р.). – Назва з екрана.

4. JetBrain tutorials – [Електронний ресурс] – Режим доступу: https://www.jetbrains.com/help/pycharm/getting-started.html (дата звернення 15.05.2023 р.). – Назва з екрана.

5. Tesla Touchscreen – [Електронний ресурс] – Режим доступу: https://www.tesla.com/ownersmanual/model3/en_jo/GUID-518C51C1-E9AC-4A68-AE12-07F4FF8C881E.html (дата звернення 21.05.2023 р.). – Назва з екрана.