Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

ВИВЧЕННЯ МЕТОДІВ ТЕСТУВАННЯ ДЕСКТОПНИХ ЗАСТОСУНКІВ
(тема)

Виконав:
студент 2 курсу, групи ІНФм-21-1
Полубєхін А.А.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Творошенко І.С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Полубєхіну Антону Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Вивчення методів тестування десктопних застосунків

затверджена наказом по університету від 9 листопада 2022 року №1469Ст

2. Термін подання студентом роботи до екзаменаційної комісії 21 листопада 2022 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, опис мови програмування Python і технічні знання з реалізації фреймворка тестування, опис функціонального, димового, інтеграційного, автоматизованого методів тестування.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз методів тестування десктопних застосунків.
2. Дослідження фреймворків автоматизованого тестування.
3. Проведення порівняльного аналізу результатів дослідження.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п. 5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми вивчення і аналіз методів тестування десктопних застосунків для визначення найбільш продуктивного метода.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Консультант з дотримання діючих стандартів та норм	доц. Творошенко І. С.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	09.11.2022	
2	Аналіз завдання, підбір літератури	09.11.22-11.11.22	
3	Аналіз літератури з досліджуваної проблеми	11.11.22-13.11.22	
4	Аналіз проблеми	13.11.22-14.11.22	
5	Дослідження методів тестування	14.11.22-19.11.22	
6	Аналіз методів	19.11.22-23.11.22	
7	Оформлення пояснювальної записки	23.11.22-26.11.22	
8	Перевірка на плагіат	26.11.22	
9	Рецензування	27.11.21	
10	Підготовка презентації та доповіді	28.11.22-29.11.22	
11	Занесення роботи в електронний архів	30.11.22	
12	Попередній захист кваліфікаційної роботи	30.11.22	

Дата видачі завдання __9 листопада 2022 р.__

Студент _____
(підпис)

Керівник роботи _____ _____доц. Творошенко І.С._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 66 с., 7 табл., 38 рис., 1 дод., 53 джерела.

МЕТОДИ ТЕСТУВАННЯ, АВТОМАТИЗАЦІЯ, ДЕСКТОПНІ ЗАСТОСУНКИ, PYTHON, PYWINAUTO, PYTEST.

Об’єктом дослідження є процес тестування десктопних застосунків.

Метою дослідження є вивчення та порівняння вибраних методів тестування десктопних застосунків, які дозволять проводити тестування з найбільшою продуктивністю.

Проведено дослідження методів тестування десктопних застосунків. Досліджено метод тепловізійного контролю. Використано фреймворки для автоматизованого тестування десктопних застосунків.

Визначено вимоги щодо реалізації автоматизованого фреймворку.

У результаті роботи здійснена програмна реалізація автоматизованого фреймворка і автотестів, за базу взято функціональне, димове, інтеграційне та автоматизоване тестування.

TESTING METHODS, AUTOMATION, DESKTOP APPLICATIONS, PYTHON, PYWINAUTO, PYTEST.

The object of research is the process of testing desktop applications.

The purpose of the research is to study and compare selected methods of testing desktop applications that will allow testing with the highest productivity.

The research of methods of testing desktop devices was carried out, and the method of thermal imaging control was studied. Frameworks are used for automation testing of desktop applications.

The requirements for the implementation of the automated framework are defined.

As a result of the work, the software implementation of the automated framework and auto-tests were carried out, where functional, smoke, integration and automated types of testing were taken as a basis.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

NASA – National Aeronautics and Space Administration (національне управління з аеронавтики та дослідження космічного простору)

ПЗ – програмне забезпечення

API – Application Program Interface (інтерфейс прикладної програми)

UX – User Experience (досвід користувача)

UI – User Interface (інтерфейс користувача)

ROI – Return on Investment (прибуток на інвестиції)

IDE – Integrated Development Environment (інтегроване середовище розробки)

GUI – Graphical User Interface (графічний інтерфейс користувача)

MS – Microsoft

QA – Quality Assurance (гарантія якості)

TMS – Testing Management System (система тест-менеджменту)

ВСТУП

Розробка програмного забезпечення наразі перебуває на етапі, коли швидкість розробки, доступні функції і якість разом відіграють провідну роль під час випуску і просування продукту.

Якість є одним з трьох стовпів на яких, тримається успіх ПЗ. У циклі розробки за неї відповідає процес тестування. Під час цього процесу проходить багато різноманітних підпроцесів, створення тестової документації, робота з критеріями, створення процесу тестування і звісно проведення тестування. Під час проведення перевірок використовують різні методи тестування, які поділяються за часом виконання, типом, є методи які є специфічними тільки для конкретних типів застосунків.

Тестування інколи займає багато часу і виконання всіх типів тестування неможливе, тому тестувальники намагаються зменшити час за допомогою автоматизації тестів, використовують конкретні методи тестування і проводять перевірки, які мають найвищий пріоритет.

1 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ТЕСТУВАННЯ

1.1 Роль тестування під час розроблення десктопних застосунків

У сучасному світі розробляється все більше різних інформаційних систем, веб та десктопних застосунків. Дані вказують, що у 2020 році було завантажено більше ніж 204 млрд. Мобільних програмних засобів, що на 45% більше, ніж було у 2016 році [1]. Дивлячись на ці статистичні показники, можна зрозуміти, що люди почали частіше використовувати різні застосунки у повсякденному житті. Через таку популярність, виросла і конкуренція між компаніями, які маючи одну ідею, намагаються перевершити одна одну, у втіленні цієї ідеї у життя. Щоб виграти у цій гонці розробок і забрати собі клієнта, компанії намагаються додавати нові функції, знижувати ціну за користування, дарувати подарунки і, звісно ж, змагаються у якості ПЗ. Маректингові дослідження, які проводяться такими гігантами ринку, як Apple, Netflix, Amazon, із року в рік вказують, що якість напряму впливає на фінальне рішення клієнта купувати і користуватися продуктом [1]. За якість під час розробки відповідає – тестування.

Тестування програмного забезпечення – це метод перевірки того, чи фактичний програмний продукт відповідає очікуваним вимогам, і переконатися, що програмний продукт без дефектів [2]. Даний метод передбачає виконання компонентів програмного забезпечення або системи за допомогою ручних чи автоматизованих інструментів для оцінки однієї чи кількох цікавих властивостей. Метою тестування програмного забезпечення є виявлення помилок, прогалин або відсутніх вимог на відміну від фактичних вимог.

Чому десктопне тестування є невід’ємною частиной розробки програмних продуктів? Завжди десктопні застосунки використовуються на різних підприємствах та організаціях, у військовій та медичній сферах.

У цих сферах застосунок може напряму впливати на людину. Отже, тестування десктопних застосунків є важливою частиною розробки програм.

Людські помилки можуть призвести до дефекту або збою на будь-якому етапі життєвого циклу розробки програмного забезпечення. Наслідки помилки можут бути непомітними для користувача або можуть стати катострофічними для компанії.

Вимога ретельного тестування та відповідної документації протягом життєвого циклу розробки програмного забезпечення виникає з наведених нижче причин (табл. 1.1).

Таблиця 1.1 – Причини ретельного тестування

Причина	**Опис причини**
Причина №1	Виявлення дефектів
Причина №2	Зменшення недоліків в компоненті або системі
Причина №3	Підвищення загальної якості системи

Важливість тестування для розробки якісного програмного забезпечення можна обґрунтувати за допомогою таких пунктів:

– тестування є важливим, оскільки воно виявляє дефекти або помилки до доставки клієнту, що гарантує якість програмного забезпечення;

– тестування робить програмне забезпечення більш надійним і простим у використанні;

– ретельно протестоване програмне забезпечення забезпечує надійну та високопродуктивну роботу програмного забезпечення.

Відповідні методи тестування, застосовані до кожного рівня тестування, разом із належним рівнем досвіду тестування гарантують абсолютне зниження частоти таких збоїв програмного забезпечення.

Навіть коли помилки присутні у застосунку їх виявлення на ранніх стадіях допоможе скоротити витрати на їх виправлення (рис. 1.1) [2].

Рисунок 1.1 – Вартість занайденої помилки на різних етапах розроблення

Є багато прикладів, коли помилки призводили до катастрофічних наслідків [3, 4].

Інженери NASA у 1998 році втратили Mars Climate Orbiter (рис. 1.2), який згорів наблизившись до поверхні Марса. Помилка була знайдена через декілька місяців, у перетворенні британських одиниць у метрику. Загальна вартість помилки склала понад 320 мільйонів доларів.

У квітні 2015 року термінал Bloomberg у Лондоні вийшов з ладу через збій програмного забезпечення, який торкнувся понад 300000 трейдерів на фінансових ринках. Ця помилка змусила уряд відкласти продаж боргу на 3 мільярди фунтів.

Рисунок 1.2 – Mars Climate Orbiter

Помилка у ПЗ стала фатальною біржі Mt. Gox, яка була найбільшою біржею біткойнів у світі.

Збій привів до того, що на біржі створювалися транзакції, які ніколи не могли бути повністю викуплені, вартістю до 1,5 мільйона доларів у

втрачених біткойнах. Через помилку у 2014 році біржа оголосила про банкрутство.

Це призвело до того, що постраждали і люди, які тримали біткоїн на цій біржі, вони все втратили.

Нажаль таких помилок, які були фатальними дуже багато, це доводить що вчасно проведене тестування, використовуючи потрібні види тестування і правильний підхід, може забезпечити якість застосунку і надійність системи.

1.2 Класифікація існуючих методів тестування

Існує багато різних видів тестування, які мають різні функції і для виконання яких потрібно мати спеціальні знання. Через широкий вибір принято класифікувати всі види тестування. Зазвичай, види тестування розподіляють на функціональні і нефункціональні.

Нефункціональне тестування (рис. 1.3) перевіряє, як працює програмне забезпечення і наскільки добре воно працює. Воно поділяється на тестування:

– локалізації;

– на відмову та відновлення;

– сумісності;

– конфігураційне;

– інсталяційне;

– тестування на зручність використання;

– тестування інтерфейсу користувача;

– усі види тестування продуктивності.

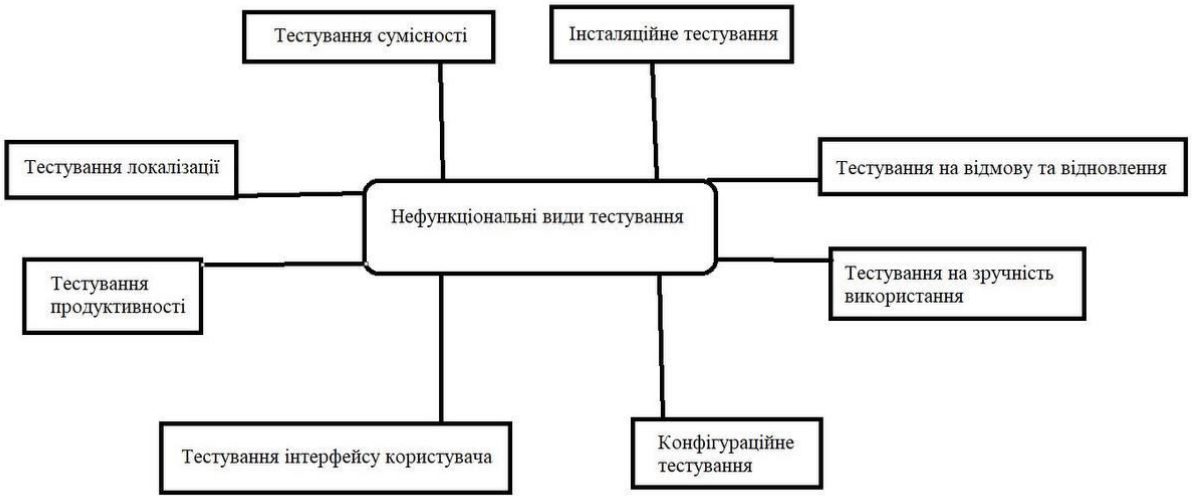


Рисунок 1.3 – Нефункціональні види тестування

Функціональне тестування (рис. 1.4) перевіряє, що робить програмне забезпечення, воно поділяється на:

– функціональне;

– тестування безпеки;

– тестування взаємодії.

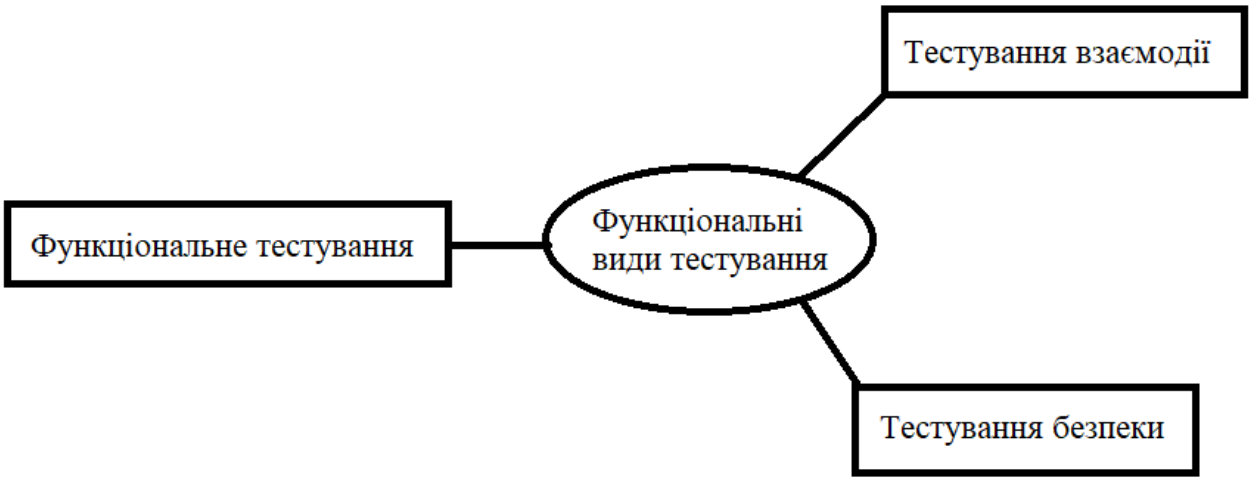


Рисунок 1.4 – Функціональні види тестування

Нефункціональне тестування так само важливе, як і функціональне. Обидва гарантують, що продукт працює належним чином.

Однак, нефункціональне тестування перевіряє речі, які не розглядаються у функціональних тестах.

Насправді класифікацій видів тестування більше, їх можна класифікувати таким чином (рис. 1.5).

– за часом виконання;

– за формальністю;

– за виконанням коду;

– за рівнями тестування;

– за виконавцем тестування;

– за ступенем автоматизації;

– позитивне і негативне тестування;

– за знанням системи;

– за розробкою тестової документації;

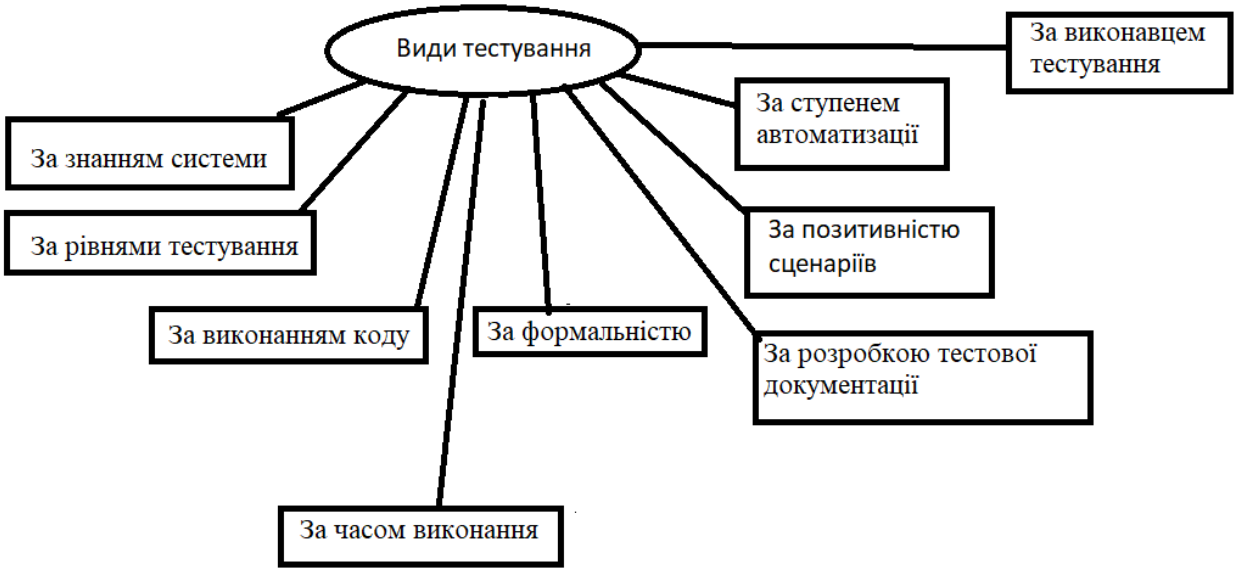


Рисунок 1.5 – Класифікація видів тестування

Тестування за часом виконання:

– комплексне;

– вхідний тест;

– основне;

– повторне;

– регресійне;

– приймальне.

Тестування за формальністю можна поділити на:

– виконання тестування за допомогою тестів;

– спезалізоване;

– дослідницьке.

Сценарії можуть бути позитивними і приймати лише коректні дані або негативними і оперувати, як позитивними так і як мінімум одним негативним значенням – так класифікують позитивні і негативні тести.

Розрізняють види тестування за ступенем автоматизації:

– автоматизоване тестування;

– ручне;

– напівавтоматизоване.

Розрізняють тестування білого ящика, коли у людини є знання про систему, та яке схоже на автоматизоване, а також чорного ящика, коли знань про систему немає, схожість з ручним тестуванням. Тестування сірого ящика, коли білий і чорний ящики переплітаються.

Тестова документація теж підпадає під класифікацію, вона може бути розробленою:

– на основі специфікації;

– на основі моделі;

– на основі варіантів використання.

Документацію також тестують, використовуючи статичне тестування, що має на меті тестування без запуску коду. Протилежним статичному тестуванню виступає динамічне, яке відбувається разом із запуском застосунку. Ці методи тестування підпадають під види тестування, що розрізняють за використанням коду.

1.3 Аналіз сучасних методів тестування десктопних застосунків

Найчастіше при тестуванні десктопних застосунків використовуються такі функціональні види тестування:

– функціональне тестування;

– тестування безпеки.

Функціональне тестування – це тип тестування програмного забезпечення, який перевіряє систему програмного забезпечення на відповідність функціональним вимогам або специфікаціям [4].

Метою функціональних тестів є тестування кожної функції програмного застосунку шляхом надання відповідних вхідних даних і перевірки вихідних даних на відповідність функціональним вимогам.

Функціональне тестування в основному включає тестування чорного ящика, і воно не стосується вихідного коду програми. Це тестування перевіряє інтерфейс користувача, API, базу даних, безпеку, зв’язок між клієнтом і сервером та інші функції програми, що тестується. Тестування можна проводити як вручну, так і за допомогою автоматизації.

Тестування безпеки – це тип тестування програмного забезпечення, який виявляє вразливі місця, загрози, ризики в програмному застосунку та запобігає зловмисним атакам зловмисників [4].

Метою тестів безпеки є виявлення всіх можливих лазівок і слабких місць програмної системи, які можуть призвести до втрати інформації, доходу, репутації з боку співробітників або сторонніх осіб організації.

З нефункціональних видів тестів використовують [5, 6]:

– усі види тестування продуктивності;

– локалізації;

– інсталяційне;

– конфігураційне;

– сумісності;

– тестування на зручність використання;

– тестування інтерфейсу користувача;

– тестування на відмову та відновлення.

Особливу увагу звертають на продуктивність і тестування інсталяції. Також є димове, регресійне, санітарне і тестування збірки, ці методи будуть розглянуті у другому розділі.

Тестування продуктивності – це техніка нефункціонального тестування програмного забезпечення, яка визначає, наскільки стабільність, швидкість, масштабованість і швидкість реагування програми зберігаються при заданому робочому навантаженні [6].

Це ключовий крок у забезпеченні якості програмного забезпечення, але, на жаль, часто розглядається, як пізня думка, ізольовано, розпочинається після завершення функціонального тестування, а в більшості випадків – після того, як код буде готовий до випуску.

Цілі тестування продуктивності включають оцінку [6, 7]:

– вихідних даних програми;

– швидкості обробки;

– швидкості передачі даних;

– використання пропускної здатності мережі;

– максимальної кількості одночасних користувачів;

– використання пам’яті;

– ефективності робочого навантаження;

– часу відповіді на команди.

Тестування продуктивності поділяється на [6, 7]:

– тестування навантаження – перевіряє здатність програми працювати в умовах очікуваного навантаження користувача;

– тестування на довговічність – проводиться, щоб переконатися, що програмне забезпечення може витримати очікуване навантаження протягом тривалого періоду часу;

– стрес-тестування – передбачає тестування програми за екстремальних робочих навантажень, щоб побачити, як вона справляється з великим трафіком або обробкою даних;

– тестування стрибків – перевіряє реакцію програмного забезпечення на раптові великі стрибки навантаження, створювані користувачами;

– об’ємне тестування – полягає в тому, щоб перевірити продуктивність програми в різних обсягах бази даних;

– тестування масштабованості – полягає в тому, щоб визначити ефективність програмного застосунку в «розширенні» для підтримки збільшення навантаження на користувачів.

Тестування продуктивності допомагає спланувати збільшення потужності програмної системи (рис. 1.6).

Тестування локалізації – це техніка тестування програмного забезпечення, яка перевіряє, чи пропонує повну функціональність і зручність використання в певному регіоні програма або вебсайт [7].

Якщо продукт налаштовано для цільової мови чи регіону, перевірка локалізації перевіряє точність і придатність вмісту [7].

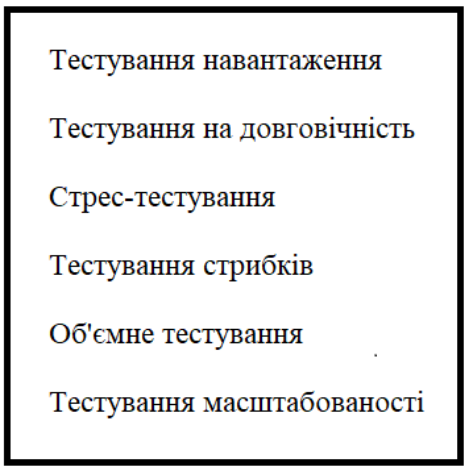


Рисунок 1.6 – Види тестування продуктивності

Тестування інсталяції виконується, щоб перевірити, чи програмне забезпечення було правильно встановлено з усіма властивими функціями та

чи продукт працює відповідно до очікувань [7]. Також відоме як тестування реалізації, воно виконується на останній фазі тестування перед першою взаємодією кінцевого користувача з продуктом.

Тестування конфігурації – це тестування програмного забезпечення, яке перевіряє продуктивність системи, що розробляється, порівняно з різними комбінаціями програмного та апаратного забезпечення, щоб знайти найкращу конфігурацію, коли система може працювати без будь-яких недоліків або проблем, відповідаючи функціональним вимогам [7] (рис. 1.7).

Рисунок 1.7 – Процес конфігураційного тестування

Тестування GUI – це тестування програмного забезпечення, яке перевіряє графічний інтерфейс користувача програмного забезпечення [7].

Мета тестування графічного інтерфейсу користувача (GUI) полягає в тому, щоб переконатися, що функціональні можливості програмного забезпечення працюють відповідно до специфікацій, перевіряючи екрани та елементи керування, наприклад, меню, кнопки, піктограми тощо.

Тестування відновлення – це техніка тестування програмного забезпечення, яка перевіряє здатність програмного забезпечення відновлюватися після збоїв таких, як збої програмного або апаратного забезпечення, збої мережі тощо [7].

Мета тестування відновлення – визначити, чи можна продовжувати роботу програмного забезпечення після аварії чи втрати цілісності. Тестування відновлення передбачає повернення програмного забезпечення

до точки, коли була відома цілісність, а також повторну обробку транзакцій до точки збою.

1.4 Аналіз літературних джерел щодо апробації результатів застосування методів тестування задачі дослідження

Тестування є важливим кроком під час розробки різних програмних засобів. Процес тестування починається, якомога раніше, коли з’являються перші критерії, того, як повинен виглядати продукт і які функції виконувати. Одним із принципів тестування є раннє тестування. Щоб знайти дефекти раніше, активності по тестуванню повинні бути розпочаті як можна раніше у життєвому циклі ПЗ, тестувальники повинні бути сфокусовані на конкретних цілях, зазначається у джерелі [1].

У статтях [2–6] розглядаються принципи тестування. Застосування різних методів тестування залежить від ПЗ, для якого ці методи застосовуються.

Зазначено, що тестування залежить від контексту, тобто різні види тестування використовуються для різних ПЗ, як приклад, для програмного засобу, у якому безпека стоїть на першому місці, тестування буде відрізнятися від тестування сайту електронної комерції.

Як правило, життєвий цикл розробки впливає на методи тестування, а також на черговість їх використання. Існує багато різних циклів таких, як каскадна модель, ітераційна, V-модель, інкрементна. V-модель є найкращою для тестувальника. У цій моделі тестування йде паралельно з розробкою. Але найбільш популярній фреймворк – це Scrum-фреймворк, його часто плутають і називають життєвим циклом ПЗ, яким він не є.

У [7] науковій праці визначається термін фреймворку Scrum. Scrum напряму впливає на процес розробки і тестування. Scrum допомагає команді розробників зосередитися на всіх аспектах розробки програмного продукту

таких, як якість, продуктивність, зручність використання тощо. Він забезпечує прозорість, перевірку та адаптацію під час розробки програмного забезпечення, щоб уникнути ускладнень.

Як приклад Scrum-тестування – це тип тестування програмного забезпечення, який виконується для перевірки здатності програмного забезпечення або програми виконувати складні процеси [8]. Це тестування також перевіряє різні інші параметри програмного забезпечення такі, як якість, зручність використання та продуктивність. Виконання складного процесу потребує складного програмного забезпечення. Отже, щоб створити складне програмне забезпечення, потрібне тестування Scrum.

У джерелі [9] надається повна інформація про мову програмування Python, яка є однією з найбільш популярних мов для автоматизованого тестування.

У статтях [10–12] повністю описані фреймворки тестування десктопних застосунків Pywinauto і PyAutoGui. Описане використання фреймворків, а також надані приклад використання контролів різних функцій.

У джерелах [13, 14] описаний принцип налаштування середовища PyCharm IDE, для розробки фреймворка тестування.

Різні види тестування використовують різні рівні заглиблення у тестування. Через це точно підібраний обсяг тестування може істотно знизити затрати, і зменшити час на проведення деяких видів тестувань, зазначено у статті [15].

1.5 Постановка задачі дослідження

У сучасному світі розробка виходить на новий рівень. Потреба у застосунках для різних доменів зростає. Медичні системи або системи, якими користуються військові, частіше за все подані у вигляді десктопних

застосунків. Через це, на сьогодні, як ніколи актуальна проблема щодо підтримки якості у десктопних застосунках, які використовуються у найбільш відповідальних сферах життя.

Обрані методи тестування (димове, інтеграційне, функціональне і автоматизоване) використовуються не тільки для десктопних застосунків, але і для веб і мобільних застосунків. Швидке тестування з найбільшою продуктивністю є основним завданням, коли ПЗ пов’язане з життям людини.

Об’єктом дослідження є процес тестування десктопних застосунків.

Метою дослідження є вивчення та порівняння вибраних методів тестування десктопних застосунків, які дозволять проводити тестування з найбільшою продуктивністю.

Виходячи із мети дослідження необхідно вирішити такі задачі:

– класифікувати та проаналізувати існуючі методи тестування;

– вибрати та вивчити методи тестування десктопних застосунків;

– визначити критерії для оцінювання методів тестування десктопних застосунків;

– провести порівняльний аналіз методів тестування десктопних застосунків;

– зробити висновок щодо вивчених та проаналізованих методів тестування десктопних застосунків.

2 ДОСЛІДЖЕННЯ ВИБРАНИХ МЕТОДІВ ТЕСТУВАННЯ ДЕСКТОПНИХ ЗАСТОСУНКІВ

Існує багато різних видів тестування, які використовуються у десктопному, мобільному і вебтестуванні. Вони поділяються за рівнями, типами і видами, а також за часом, коли вони використовуються.

Багато часу витрачається на проведення різних видів тестів, на різних проєктах – різні підходи, але дуже часто використовуючи різні види тестів навіть тоді, коли їх використання не принесе бажаного ефекту, тестувальники витрачають час, який можно було би використати більш раціонально. Ця проблема існує на великих проєктах і на малих, у різних компаніях і на проєктах з різними методологіями.

Щоб вирішити цю проблему було визначено набір різних видів тестування, які будуть досліджуватись: функціональне, інтеграціне, димове, а також автоматизоване. Ці види тестування можно реалізувати у вигляді авто тестів, які можут значно скоротити тестування, покращити якість застосунків, а також зменшити витрати на проєкті.

2.1 Метод димового тестування

Димове тестування – це попередня перевірка програмного забезпечення після збірки та перед випуском. Цей тип тестування виявляє базові та критичні проблеми в ПЗ до того, як розпочнеться критичне тестування.

Термін «тестування на дим» походить від аналогічного базового типу тестування апаратного забезпечення, під час якого пристрій проходить тест, якщо він не загорівся під час першого ввімкнення.

Тестувальники із забезпечення якості проводять димове тестування після того, як розробники створюють нову збірку (рис. 2.1).

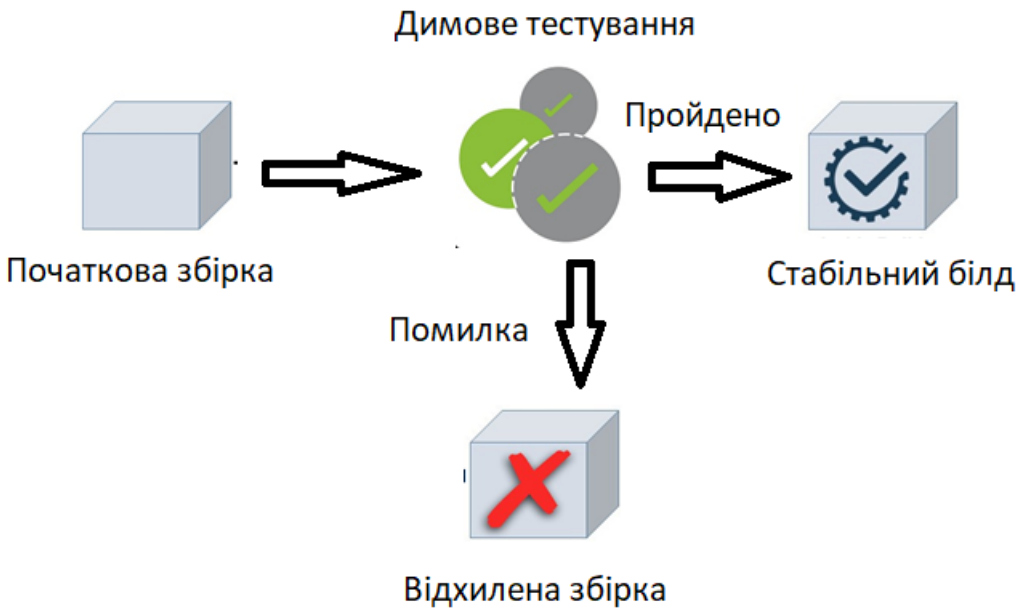


Рисунок 2.1 – Процес димового тестування

Якщо димове тестування проходить вдало, збірка програмного забезпечення переходить до більш суворих тестів, таких як модульні та інтеграційні тести. Якщо були виявлені помилки, це означає, що тестування виявили серйозний недолік, який зупиняє всі подальші тести. Потім тестувальник просить розробників надіслати іншу збірку. Цей широкий початковий тест є більш ефективною стратегією вдосконалення програмного коду, ніж якби команда проводила спеціальні та ретельні тести на цьому ранньому етапі процесу розробки.

Димове тестування також виконується з точки зору досвіду користувача (UX). Цей підхід включає тестування ключових функціональних можливостей, наприклад, чи доступна збірка, чи функціонує інтерфейс користувача (UI) і механізм входу.

Інші ключові функції включають, якщо вибір дії корелює з запланованою дією. Наприклад, якщо користувач додає товар до кошика для покупок у вебпрограмі електронної комерції, чи з’являється цей товар у кошику?

У індустрії програмного забезпечення існує плутанина між термінами димове тестування та перевірка розумності. Деякі професіонали використовують термін «тестування на осудність» як еквівалент «димове тестування», тоді як інші використовують «тестування на осудність» як термін, який відноситься до підмножини регресійного тестування.

2.2 Метод інтеграційного тестування

Інтеграційне тестування визначається, як тип тестування, у якому програмні модулі логічно інтегруються та тестуються як група. Типовий програмний проєкт складається з кількох програмних модулів, закодованих різними програмістами. Метою цього рівня тестування є виявлення дефектів у взаємодії між програмними модулями під час їх інтеграції.

Інтеграційне тестування зосереджується на перевірці передачі даних між цими модулями. Тому його також називають «I & T» (інтеграція та тестування), «тестування рядків» і іноді «тестування потоків» (рис. 2.2).

Модуль, як правило, розробляється окремим розробником програмного забезпечення, чиє розуміння та логіка програмування можуть відрізнятися від інших програмістів. Тестування інтеграції стає необхідним для перевірки роботи модулів програмного забезпечення в єдності.

Під час розробки модуля існує велика ймовірність зміни вимог клієнтів. Ці нові вимоги можуть не проходити модульне тестування, тому тестування системної інтеграції стає необхідним.

Приклади помилок, інтерфейси програмних модулів з базою даних можуть бути помилковими, зовнішні апаратні інтерфейси, якщо такі є, можуть бути помилковими, неадекватна обробка винятків може спричинити проблеми.

Рисунок 2.2 – Системні види тестів

Розробка програмного забезпечення визначає різноманітні стратегії для виконання інтеграційного тестування, а саме, підхід великого вибуху, підхід зверху вниз, підхід знизу вгору і сендвічовий підхід – поєднання «зверху вниз» і «знизу вгору».

Підхід великого вибуху – це підхід до інтеграційного тестування, за якого всі компоненти або модулі об’єднуються разом, а потім тестуються як єдине ціле. Цей комбінований набір компонентів під час тестування розглядається як єдине ціле. Якщо всі компоненти пристрою не завершено, процес інтеграції не буде виконано.

Переваги: зручно для невеликих систем.

Недоліки: локалізація несправності складна. Враховуючи величезну кількість інтерфейсів, які необхідно протестувати в цьому підході, деякі посилання на інтерфейси, які потрібно перевірити, можна легко пропустити.

Оскільки інтеграційне тестування може розпочатися лише після того, як спроєктовано «всі» модулі, команда тестувальників матиме менше часу для виконання на етапі тестування. Через те, що всі модулі перевіряються одночасно, критичні модулі високого ризику не ізольовані та перевіряються в

пріоритеті. Периферійні модулі, які працюють з інтерфейсами користувача, також не ізольовані та тестуються в пріоритеті.

Інтеграційне тестування «знизу вгору» – це стратегія, за якої спочатку тестуються модулі нижчого рівня. Ці протестовані модулі потім використовуються для полегшення тестування модулів вищого рівня. Процес триває, доки не будуть перевірені всі модулі верхнього рівня. Після того, як модулі нижчого рівня перевірені та інтегровані, формується наступний рівень модулів (рис. 2.3).

Переваги: локалізація несправності легша. На відміну від підходу великого вибуху, час не витрачається на очікування розробки всіх модулів.

Недоліки: критичні модулі (на верхньому рівні архітектури програмного забезпечення), які контролюють потік програми, перевіряються в останню чергу та можуть бути схильні до дефектів. Ранній прототип неможливий.

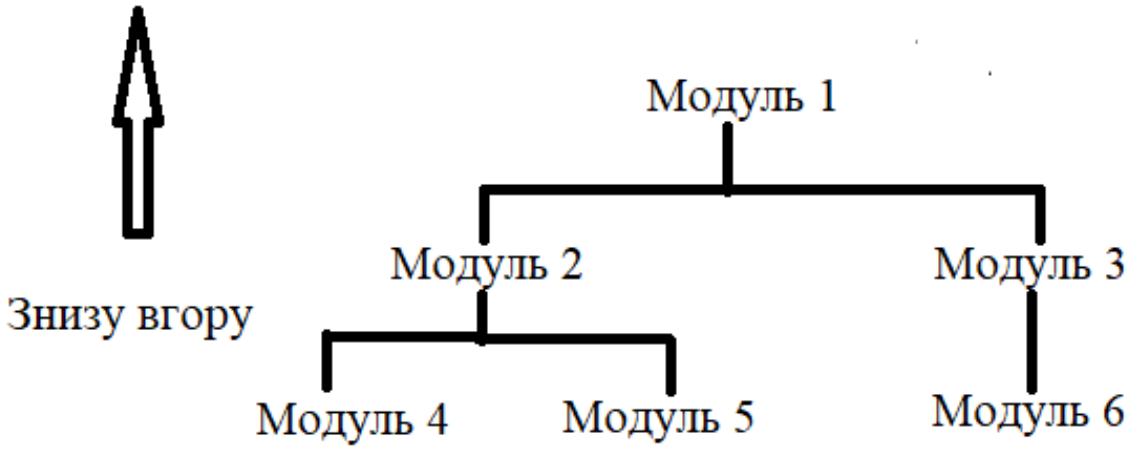


Рисунок 2.3 – Інтеграційне тестування «знизу вгору»

Інтеграційне тестування «зверху вниз» – це метод, де інтеграційне тестування відбувається зверху вниз за керуванням програмною системою.

Спочатку тестуються модулі вищого рівня, а потім тестуються та інтегруються модулі нижчого рівня, щоб перевірити функціональність

програмного забезпечення. Заглушки використовуються для тестування, якщо деякі модулі не готові (рис. 2.4).

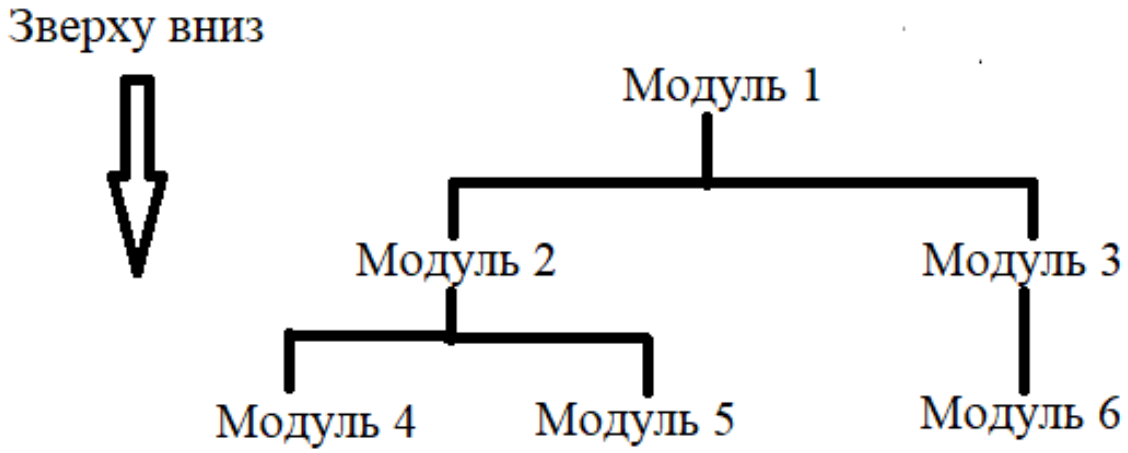


Рисунок 2.4 – Інтеграційне тестування «зверху вниз»

Переваги: локалізація несправності легша. Можливість отримати ранній прототип. Критичні модулі перевіряються в пріоритеті; основні недоліки дизайну можна знайти та виправити спочатку.

Недоліки: потрібно багато заглушок. Модулі нижчого рівня перевірені неадекватно [16].

Сендвіч-тестування – це стратегія, за якої модулі верхнього рівня тестуються з модулями нижчого рівня, у той же час модулі нижчого рівня інтегруються з модулями верхнього рівня та тестуються як система (рис. 2.5). Це поєднання підходів «зверху вниз» і «знизу вгору», тому воно називається гібридним інтеграційним тестуванням.

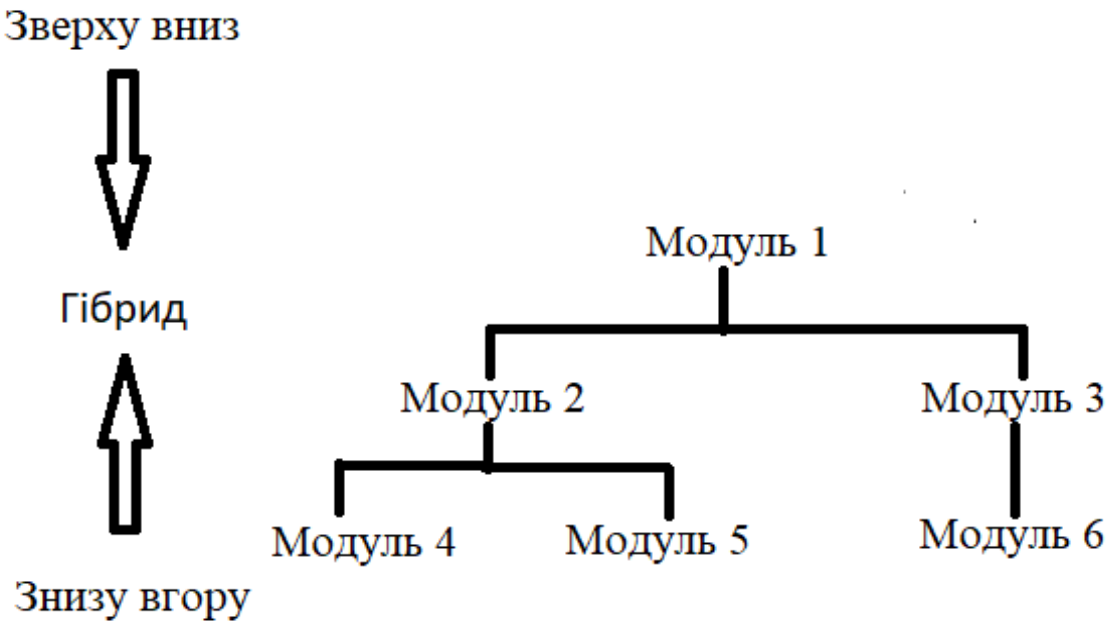


Рисунок 2.5 – Сендвіч-тестування

Процедура тестування інтеграції незалежна від стратегій тестування програмного забезпечення (табл. 2.1).

Таблиця 2.1 – Етапи інтеграційного тестування

Етапи	**Опис етапу**
Етап 1	Підготовка плану інтеграційних тестів
Етап 2	Розробка тестовихх сценаріїв, кейсів
Етап 3	Виконання тестових випадків із подальшим звітуванням про дефекти
Етап 4	Відстеження та повторне тестування дефектів

Кроки 3 і 4 повторюються, доки інтеграція не завершиться успішно.

2.3 Метод функціонального тестування

Функціональне тестування – це тип тестування програмного забезпечення, який перевіряє програмну систему на відповідність функціональним вимогам/специфікаціям. Процес функціонального тестування полягає у тестуванні кожної функції програмного застосунку шляхом надання відповідних вхідних даних і перевірки вихідних даних на відповідність функціональним вимогам (рис. 2.6).

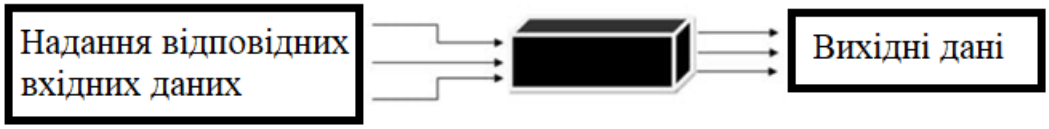


Рисунок 2.6 – Процес функціонального тестування

Функціональне тестування включає тестування чорного ящика, і воно не стосується вихідного коду програми. Це тестування перевіряє інтерфейс користувача, API, базу даних, безпеку, зв’язок між клієнтом і сервером та інші функції програми, що тестується.

Детальне функціональне тестування складається з таких кроків. Спочатку потрібно визначити, які функції продукту потрібно перевірити. Це може відрізнятися від тестування основних функцій, повідомлень, умов помилки та/або зручності використання продукту. Створити вході дані для функціональних можливостей, які потрібно перевірити відповідно до визначених вимог. Далі визначити прийнятні вихідні параметри відповідно до заданих вимог. Виконати тест-кейси. Порівняйте фактичні результати тесту з попередньо визначеними вихідними значеннями. Це показує, чи система працює належним чином.

Основною метою функціонального тестування є перевірка функціональності програмної системи (табл. 2.2).

Таблиця 2.2 – Основні напрямки перевірок функціонального тестування

Перевірка	**Опис перевірки**
Основні функції	Тестування основних функцій програми
Базове тестування зручності використання системи	Перевіряється, чи може користувач вільно переміщатися по екранах без будь-яких труднощів
Доступність	Перевіряється доступність системи для користувача
Умови помилок	Використання методів тестування для перевірки умов помилок. Перевіряється, чи відображаються відповідні повідомлення про помилки

Переваги функціонального тестування: це тестування відтворює або є копією фактичної системи, тобто це копія того, яким є продукт у реальному середовищі.

Тестування зосереджено на специфікаціях відповідно до використання клієнтом, тобто системними специфікаціями, операційною системою, браузерами тощо.

Це тестування гарантує надання високоякісного продукту, який відповідає вимогам клієнта, продукту без помилок, який має всі функції, що працюють відповідно до вимог клієнта.

2.4 Метод автоматизованого тестування

Автоматизоване тестування або тестування білого ящика – 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 систему, яка тестується, порівнювати очікувані та фактичні результати та створювати докладні 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.

Автоматизація тестування – найкращий спосіб підвищити ефективність і швидкість виконання тестування програмного забезпечення. Автоматизоване тестування програмного забезпечення є важливим з таких причин: ручне тестування всіх робочих процесів, усіх полів, усіх негативних сценаріїв потребує часу та грошей. Автоматизація тестування навпаки збільшує швидкість виконання. Автоматизація допомагає покрити більшу частину функціоналу тестами, у той час, як ручне тестування може стати нудним і, отже, схильним до помилок.

Тестові випадки, які потрібно автоматизувати, можна вибрати за таким критерієм для підвищення рентабельності інвестицій автоматизації, як показано у таблиці 2.3.

Таблиця 2.3 – Критерії вибору тестів для автоматизації

Тестові випадки	**Критерії**
Підходять для автоматизації	Критичні тестові випадки для бізнесу Тестові випадки, які багаторазово виконуються Тестові випадки, які забирають багато часу
Не підходять для автоматизації	Тестові випадки, створені заново і не виконані вручну принаймні один раз Тестові випадки, які часто змінюються Тестові випадки, які виконуються на нерегулярній основі

Процес автоматизованого тестування відображено на рисунку 2.7.

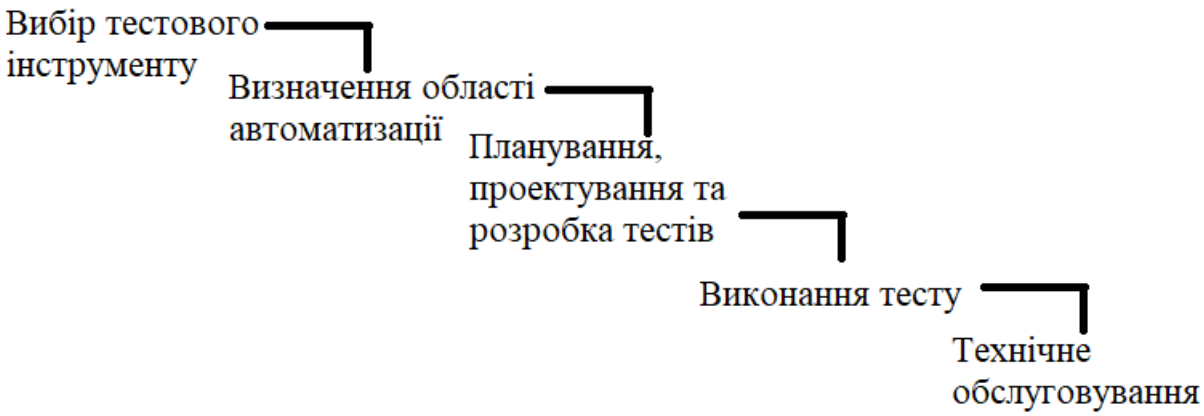


Рисунок 2.7 – Етапи процесу автоматизованого тестування

Процес автоматизованого тестування складається з наступних кроків: вибір тестового інструменту, зараз існує багато різних фреймворків за допомогою, яких можно автоматизувати мобільні, веб і десктопні застосунки.

Вибір інструменту тестування значною мірою залежить від технології, на якій побудовано застосунок.

Фреймворки є на основі мов програмувань або у вигляді вже створених елементів, які можно використовувати з конкретним застосунком. Визначення області автоматизації. Потім йде планування, проєктування та розробка тестів, що є основним етапом процесу автоматизації тестування. Фінальними етапами є виконання тесту та технічне обслуговування [17–24].

Технічне обслуговування, або підхід підтримки автоматизації тестування є фазою автоматизованого тестування, яка виконується для того, щоб перевірити, чи добре працюють нові функції, додані до програмного забезпечення і має на меті підтримку існуючих тестів, оновлюючи, виправляючи помилки і розширюючи покриття функціоналу тестами [25–34].

3 ВИВЧЕННЯ МЕТОДІВ ТЕСТУВАННЯ ДЕСКТОПНИХ ЗАСТОСУНКІВ СТОСОВНО ВИБРАНОЇ ПРЕДМЕТНОЇ ОБЛАСТІ

3.1 Вибір інструментальних засобів для реалізації вибраних методів

Для реалізації тестових сценаріїв спочатку потрібно вибрати середовище розробки, для різних мов програмування використовуються різні IDE.

Інтегроване середовище розробки (IDE) – це програмне забезпечення для створення програм, яке об’єднує загальні інструменти розробника в єдиний графічний інтерфейс користувача (GUI) [35–46].

IDE, зазвичай, складається з редактору вихідного коду, це текстовий редактор, який може допомогти у написанні програмного коду за допомогою таких функцій, як підсвічування синтаксису за допомогою візуальних підказок, забезпечення автозавершення для певної мови та перевірка на наявність помилок під час написання коду [47–52].

Автоматизація локальної збірки, утиліти, які автоматизують прості, повторювані завдання, як частину створення локальної збірки програмного забезпечення для використання розробником, як-от компіляція вихідного коду комп’ютера у двійковий код, пакування двійкового коду та виконання автоматизованих тестів.

І останній елемент, це налагоджувач: програма для тестування інших програм, яка може графічно відображати місце помилки в оригінальному коді.

IDE дозволяє розробникам швидко розпочати програмування нових застосунків, оскільки утиліти не потрібно налаштовувати та інтегрувати вручну під час процесу підключення.

Розробникам також не потрібно витрачати години окремо на вивчення того, як використовувати різні інструменти, коли кожна утиліта представлена в одному робочому місці.

Це може бути особливо корисним для залучення нових розробників, які можуть покластися на IDE, щоб пришвидшити роботу зі стандартними інструментами та робочими процесами команди. Насправді більшість функцій IDE спрямовані на економію часу, як-от інтелектуальне завершення коду та автоматизоване створення коду, що усуває потребу вводити повну послідовність символів.

Інші поширені функції IDE покликані допомогти розробникам організувати свій робочий процес і вирішити проблеми. IDE аналізує код, коли він написаний, тому помилки, спричинені людьми, визначаються в режимі реального часу.

Оскільки утиліти представлені одним графічним інтерфейсом користувача, розробники можуть виконувати дії, не перемикаючись між програмами. Підсвічування синтаксису також поширене в більшості IDE, де використовуються візуальні підказки для розрізнення граматики в текстовому редакторі.

Деякі IDE додатково містять браузери класів і об’єктів, а також діаграми ієрархії класів для певних мов.

Можна розробляти програми без IDE, кожен розробник по суті створює власну IDE, вручну інтегруючи різні утиліти з легким текстовим редактором, таким як Vim або Emacs.

Для деяких розробників перевага цього підходу полягає в ультра налаштуванні та контролі, які він пропонує. Однак, у корпоративному контексті економія часу, стандартизація середовища та функції автоматизації сучасних IDE, зазвичай, переважають інші міркування.

Через використання мови програмування Python, під час розробки тестового фреймворка, була вибрана PyCharm IDE (рис. 3.1).

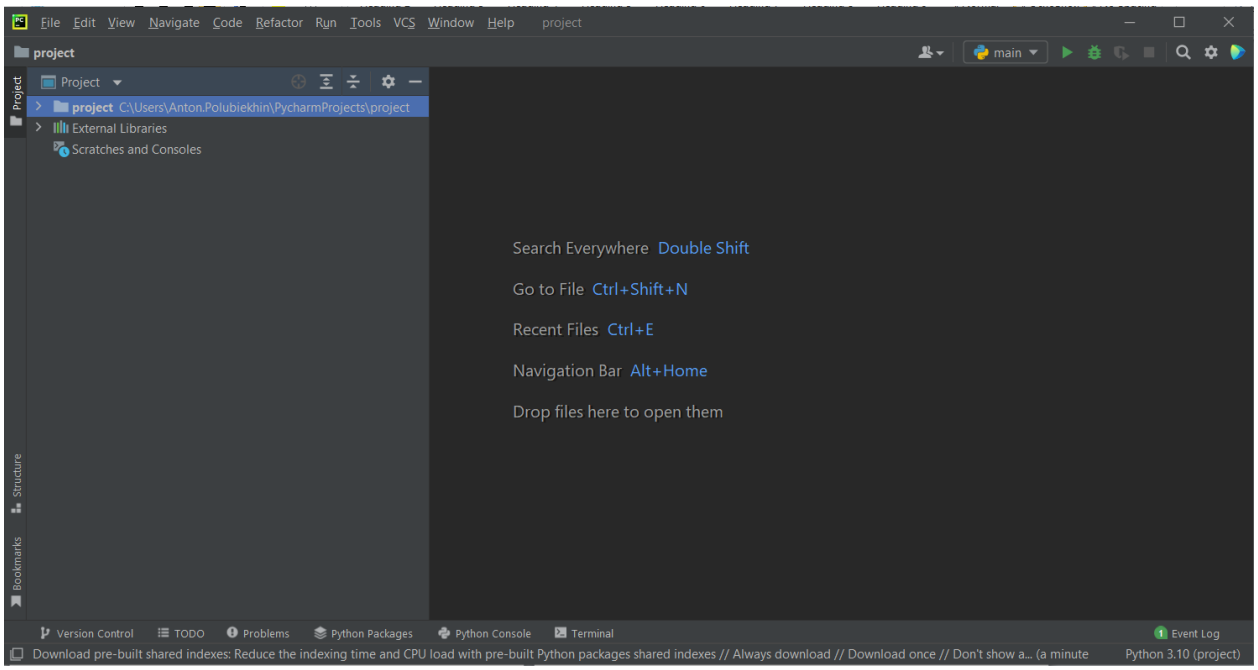


Рисунок 3.1 – Вигляд головної сторінки PyCharm IDE

PyCharm – це гібридна платформа, розроблена JetBrains, як IDE для Python. Зазвичай, використовується для розробки застосунків Python. Деякі з організацій єдинорога, такі, як Twitter, Facebook, Amazon Pinterest, використовують PyCharm, як своє середовище розробки Python.

Запускати PyCharm можна на Windows, Linux або Mac OS. Крім того, він містить модулі та пакети, які допомагають програмістам розробляти програмне забезпечення за допомогою Python за менший час і з мінімальними зусиллями. Крім того, його також можна налаштувати відповідно до вимог розробників.

Python – це інтерпретована об’єктно-орієнтована мова програмування високого рівня з динамічною семантикою [20].

Його високорівневі вбудовані структури даних у поєднанні з динамічною типізацією та динамічним зв’язуванням роблять його дуже привабливим для швидкої розробки застосунків, а також для використання як мови сценаріїв або з’єднувальної мови для з’єднання існуючих компонентів.

Простий, легкий для вивчення синтаксис Python підкреслює читабельність і, отже, знижує вартість обслуговування програми. Python

підтримує модулі та пакети, що заохочує модульність програми та повторне використання коду. Інтерпретатор Python і обширна стандартна бібліотека доступні у вихідному або двійковому вигляді безкоштовно для всіх основних платформ і можуть вільно поширюватися.

Часто програмісти вибирають Python через підвищену продуктивність, яку він забезпечує. Оскільки етапу компіляції немає, цикл редагування-тестування-налагодження відбувається неймовірно швидко. Налагоджувати програми на Python легко: помилка чи неправильний вхід ніколи не спричинить помилку сегментації.

Натомість, коли інтерпретатор виявляє помилку, він викликає виняток. Якщо програма не вловлює виняток, інтерпретатор друкує трасування стека. Налагоджувач рівня вихідного коду дозволяє перевіряти локальні та глобальні змінні, оцінювати довільні вирази, встановлювати контрольні точки, покроково виконувати код по рядках і так далі.

Сам налагоджувач написаний на Python, що свідчить про інтроспективну силу Python. З іншого боку, часто найшвидшим способом налагодження програми є додавання кількох операторів друку до джерела: швидкий цикл редагування-тестування-налагодження робить цей простий підхід дуже ефективним.

Для написання тестів також були вибрані фреймворки автоматизованого тестування десктопних застосунків Pywinauto і PyAutoGUI.

Pywinauto – це набір модулів Python для автоматизації графічного інтерфейсу Microsoft Windows. У найпростішому вигляді він дозволяє надсилати дії миші та клавіатури до діалогових вікон і елементів керування Windows, але він підтримує більш складні дії, наприклад отримання текстових даних.

Підтримувані технології під капотом: Win32 API (backend=»win32»; використовується за замовчуванням), MS UI Automation (backend=»UI»). Модулі емуляції введення користувача миша та клавіатура працюють як у Windows так і в Linux.

PyAutoGUI дозволяє сценаріям Python керувати мишею та клавіатурою для автоматизації взаємодії з іншими програмами. API розроблений таким чином, щоб бути простим. PyAutoGUI працює у Windows, Mac OS і Linux, а також працює на Python 2 і 3.

PyAutoGUI має кілька функцій: переміщення миші та клацання у вікнах інших програм, надсилання натискань клавіш (наприклад, для заповнення форм), можливість робити скріншоти або заванатжувати і знаходити його на екрані, знаходити вікно програми, переміщувати, змінювати розмір, розгортати, згортати або закрити його. А також відображати вікна сповіщень і повідомлень.

Для запуску тестів був обраний фреймворк PyTest.

PyTest – це платформа тестування, яка дозволяє користувачам писати тестові коди за допомогою мови програмування Python. Допомагає писати прості та масштабовані тести для баз даних, API або інтерфейсу користувача. PyTest в основному використовується для написання тестів для API. Допомагає писати тести від простих модульних тестів до складних функціональних тестів. Перевагою PyTest є те, що дуже легко почати використовувати через його простий і легкий синтаксис, можна запускати тести паралельно.

Можна виконувати певний тест або підмножину тестів, є автоматизоване визначення тестів, можна пропустити тести.

Для отримання результатів був вибраний фреймворк Allure. Allure Framework – це гнучкий легкий багатомовний інструмент звіту про тестування, який не тільки показує дуже стисле представлення того, що було протестовано в акуратній формі вебзвіту, але дозволяє кожному, хто бере участь у процесі розробки, отримувати максимум корисної інформації з виконання тестів.

З точки зору QA звіти Allure скорочують загальний життєвий цикл дефектів: невдачі тестів можна розділити на помилки та несправні тести, також можна налаштувати журнали, кроки, фікстури, вкладення, час, історію

та інтеграцію з TMS і системами відстеження помилок, тому відповідальні розробники та тестувальники матимуть всю інформацію під рукою.

З точки зору менеджерів, Allure надає чітку «загальну картину» того, які функції були охоплені, де згруповані дефекти, як виглядає графік виконання та багато інших зручних речей. Модульність і розширюваність Allure гарантує, що завжди зможете щось налаштувати, щоб Allure більше підходив.

3.2 Етапи програмної реалізації вибраних методів тестування десктопних застосунків

Для програмної реалізації вибраних методів тестування були визначені декілька етапів.

Під час першого етапу була визначена предметна область і її особливості. Для реалізації тестів був вибраний десктопний застосунок Notepad (рис. 3.2).

Notepad має багато різний функцій, для яких можна застосувати вибрані види тестування: функціональне, димове, інтеграційне, автоматизоване.

Наступні перевірки потрібно виконати:

– основна сторінка Notepad;

– функція «Save As»;

– функція «Open»;

– функція «Print»;

– функція «Font».

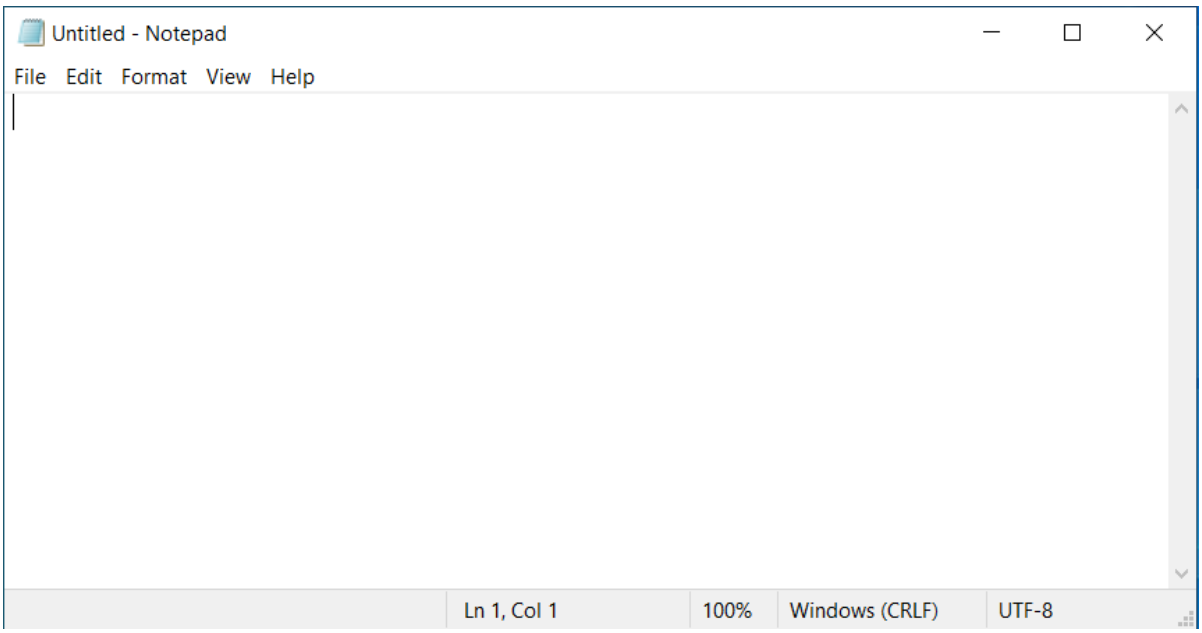


Рисунок 3.2 – Сторінка створення нового файлу у Notepad

Під час другого етапу були оцінені, вхідні дані, які були отримані під час вибору програмного забезпечення для реалізації:

– встановлена та налаштована PyCharm IDE;

– встановлений інтерпретатор Python (версія не нижче, ніж 3.6);

– встановлений і налаштований фреймворк Pywinauto;

– встановлений і налаштований фреймворк PyAutoGUI;

– встановлений і налаштований фреймворк PyTest;

– встановлений і налаштований фреймворк Allure;

– десктопний застосунок Notepad.

Під час третього етапу, визначаються основні задачі:

– перевірка відображення елементів на сторінці;

– перевірка функціоналу.

Програмна реалізація відбувається на четвертому етапі.

Реалізація перевірки автоматизованим тестуванням, що застосунок Notepad відкривається, наведена у лістингу 3.1.

Тест відноситься до комплекту основної сторінки Notepad.

Лістинг 3.1 Реалізація перевірки, що застосунок Notepad відкривається:

```
def test_auto_main_window():
    app = Application(backend=»uia»).start(«notepad.exe»)
    dlg_main = app.UntitledNotepad
    dlg_main.wait(‘visible’)
```

Реалізація перевірки за допомогою димового тестування, що застосунок Notepad відкривається і версії співпадають, наведена у лістингу 3.2.

Тест відносится до комплекту основної сторінки Notepad.

Лістинг 3.2 Реалізація первірки, що застсоунок Notepad відкривається і версії співпадають:

```
def test_smoke_main_window():
    app = Application(backend=”uia”).start(“notepad.exe”)
    dlg_main = app.UntitledNotepad
    dlg_main.wait(‘visible’)
    app.UntitledNotepad.menu_select(“Help->About Notepad”)
    assert app.UntitledNotepad.AboutNotepad.Static2.texts() == [‘Version 21H2 (OS Build 19044.1706)’]
```

Під час тестування, якщо фактична версія не співпадає з очікуваною, тест буде провалений, за допомогою функції assert зможемо побачити, де була помилка (рис. 3.3).

Різниця між першим і другим тестом у тому, що під час автотесту, перевіряємо тільки те, що основна сторінка Notepad відкривається, під час димового тестування також відбувається перевірка на співпадання версій застосунку.

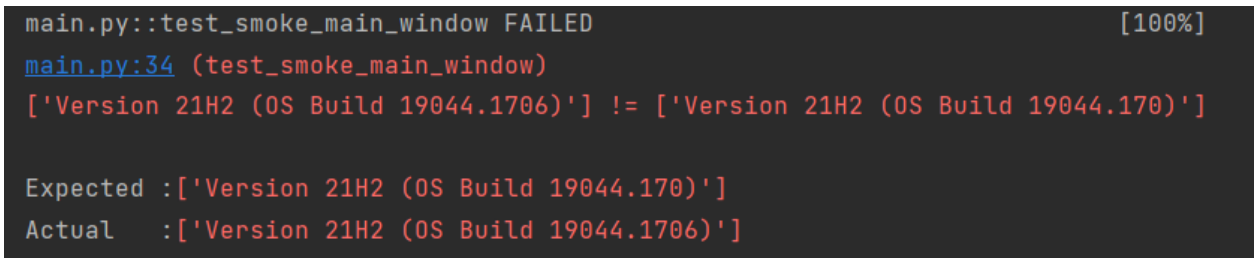


Рисунок 3.3 – Вигляд тесту, який знайшов помилку

Реалізація перевірки функціональним тестуванням, що застосунок Notepad відкривається і основна функція написання тексту працює, наведена у лістингу 3.3.

Тест відносится до комплекту основної сторінки Notepad.

Лістинг 3.3 Реалізація первірки, що застсоунок Notepad відкривається і основна функція написання тексту працює:

```
def test_functional_main_window():
    app = Application(backend=»uia»).start(«notepad.exe»)
    pyautogui.write(«Test22»)
    pic = pyautogui.screenshot()
    pic.save(‘Screenshot3.png’)
```

Під час тесту робиться скріншот, який дає змогу перевірити відпрацювання тесту (рис. 3.4).

Реалізація перевірки інтеграційним тестуванням, що діалогове вікно «Print» для застосунку Notepad відкривається за допомогою гарячої клавіші і пункту меню, наведена у лістингу 3.4.

Тест відноситься до комплекту функції «Print» застосунку Notepad.

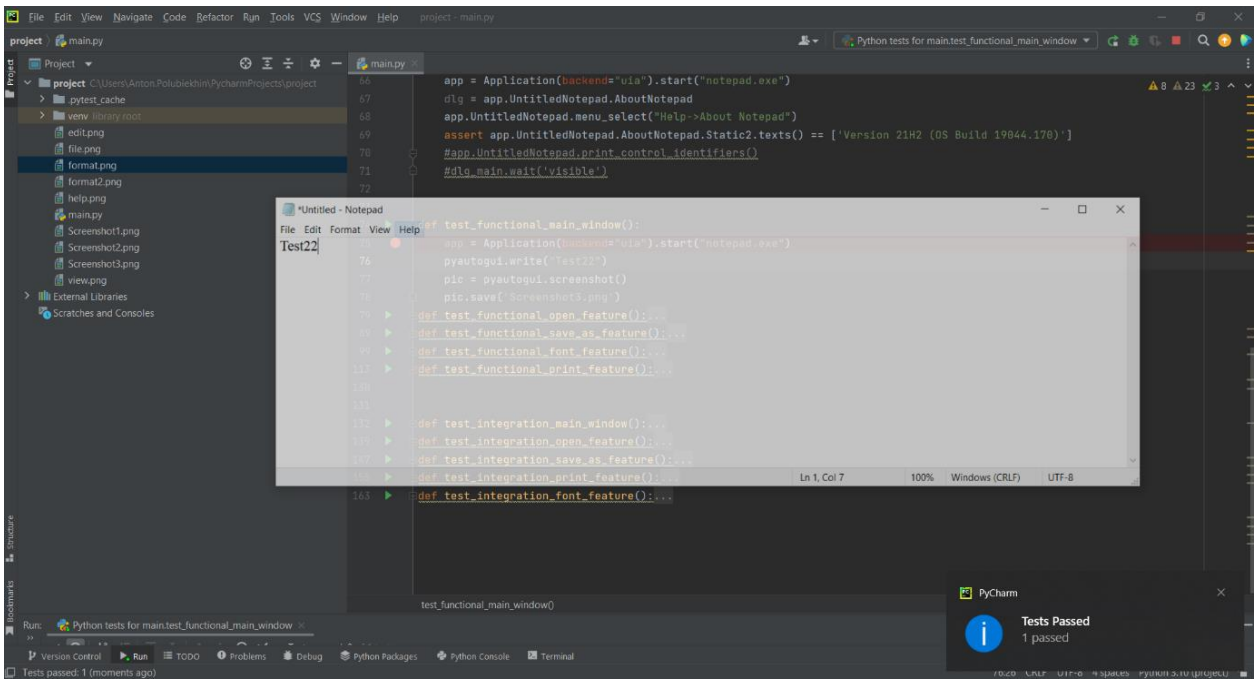


Рисунок 3.4 – Відпрацювання тесту

Лістинг 3.4 Реалізація перевірки, що діалогове вікно «Print» для застосунку Notepad відкривається:

```
def test_integration_print_feature():
    app = Application(backend=»uia»).start(«notepad.exe»)
    app.UntitledNotepad.menu_select(«File->Print»)
    dlg_print = app.UntitledNotepad.Print
    dlg_print.wait(‘visible’)
    app.UntitledNotepad.Print.Cancel.click()
    pyautogui.hotkey(‘ctrl’, ‘p’)
    dlg_print.wait(‘visible’)
```

3.3 Впровадження методів тестування десктопних застосунків стосовно вибраної предметної області

Для того щоб почати запуск тестів, потрібно спочатку налаштувати середовище розробки, у даному випадку це PyCharm IDE.

Перш ніж перейти до середовища розробки потрібно встановити Python interpreter, у даному випадку версія повинна бути не нижча ніж 3.6.

Наступним кроком відкриваємо PyCharm і створюємо новий проєкт. Якщо Python interpreter не буде вибраний автоматично, тоді його можна буде вибрати у полі Base interpreter (рис. 3.5).

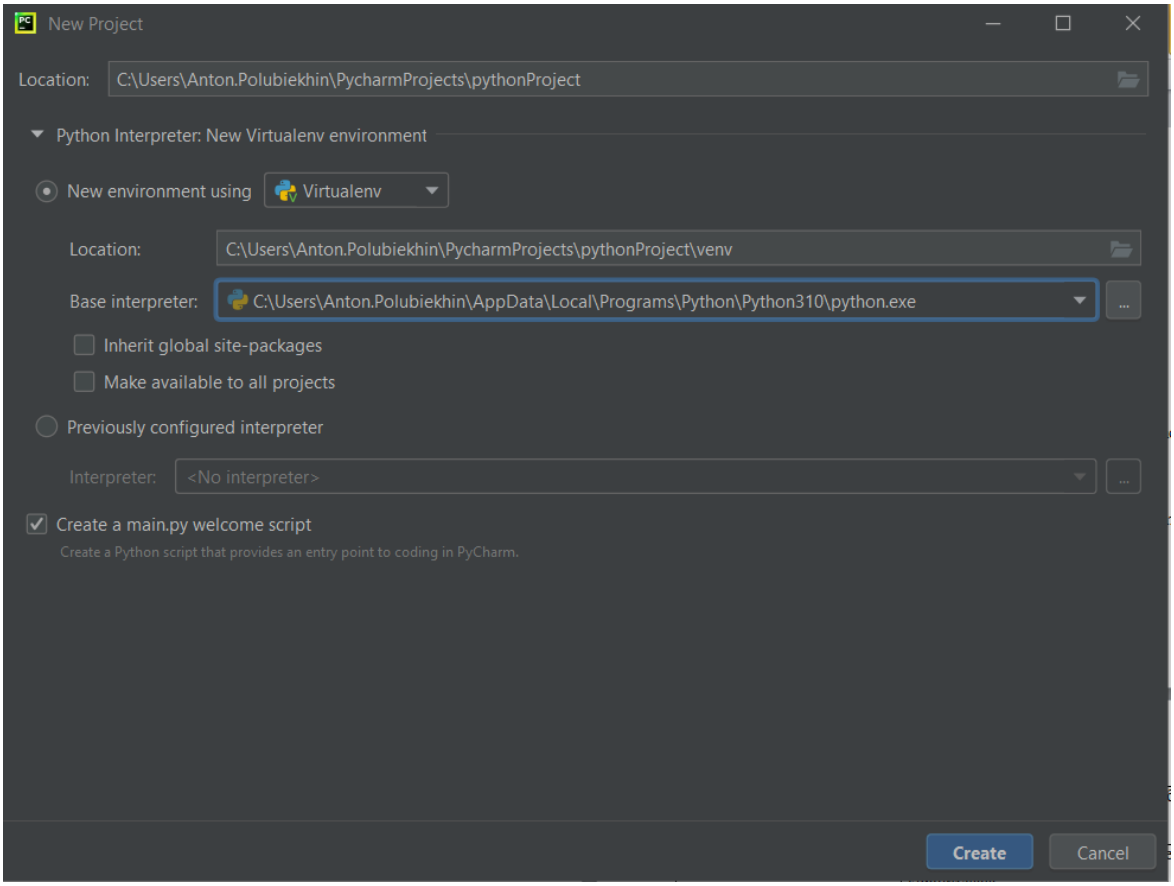


Рисунок 3.5 – Сторінка створення нового проєкту у PyCharm IDE

Після створення нового проєкту, потрібно завантажити усі фреймворки, які будуть використані під час прогону тестів.

Для цього переходимо у Terminal і вводимо наступну команду «pip install pywinauto», за допомогою неї фреймворк Pywinauto буде завантажений до проєкту (рис. 3.6).

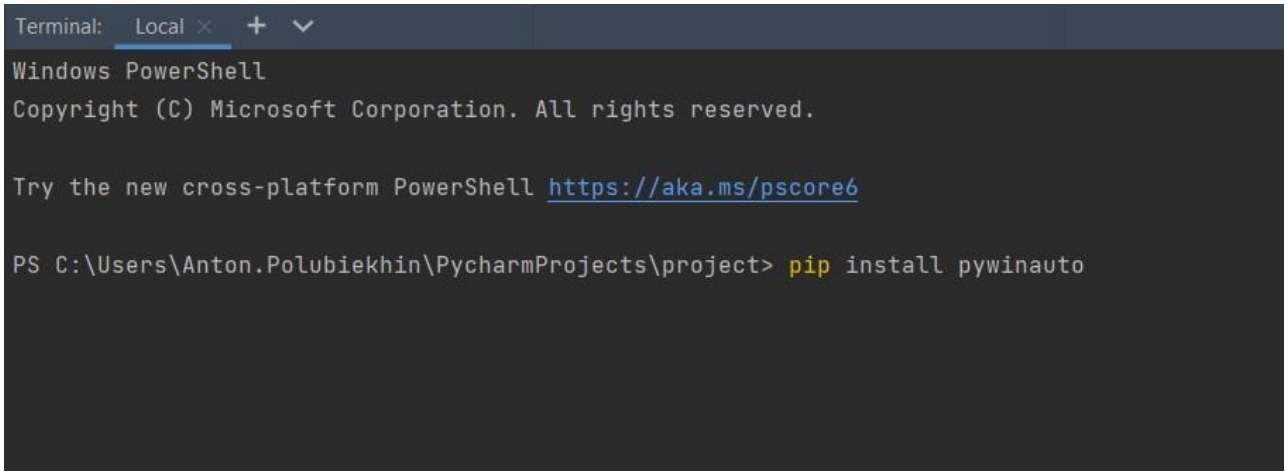


Рисунок 3.6 – Завантаження фреймворку Pywinauto

Завантаження фреймворку PyAutoGui, «pip install pyautogui» (рис. 3.7).

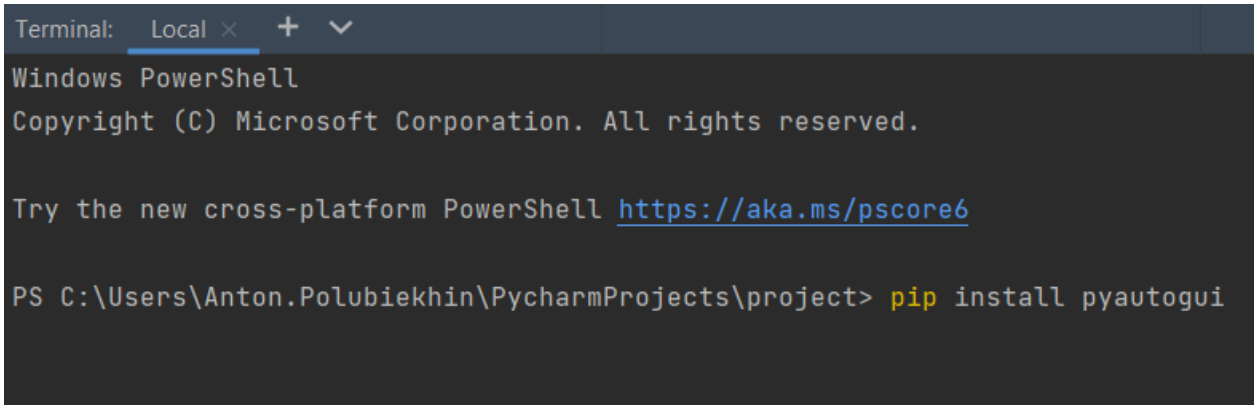


Рисунок 3.7 – Завантаження фреймворку PyAutoGui

Завантаження фреймворку Pytest, «pip install pytest» (рис. 3.8).

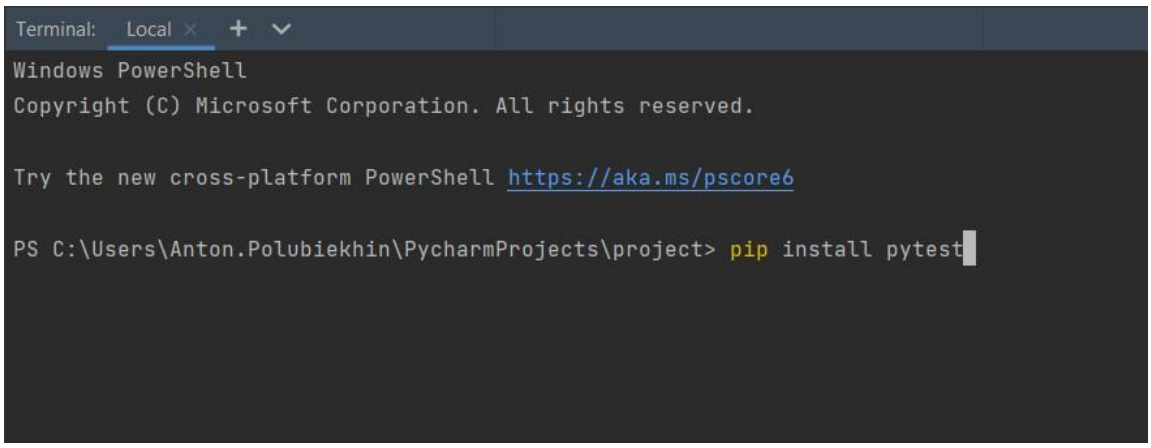


Рисунок 3.8 – Завантаження фреймворку Pytest

Завантаження фреймворку Allure, «pip install allure-pytest» (рис. 3.9).

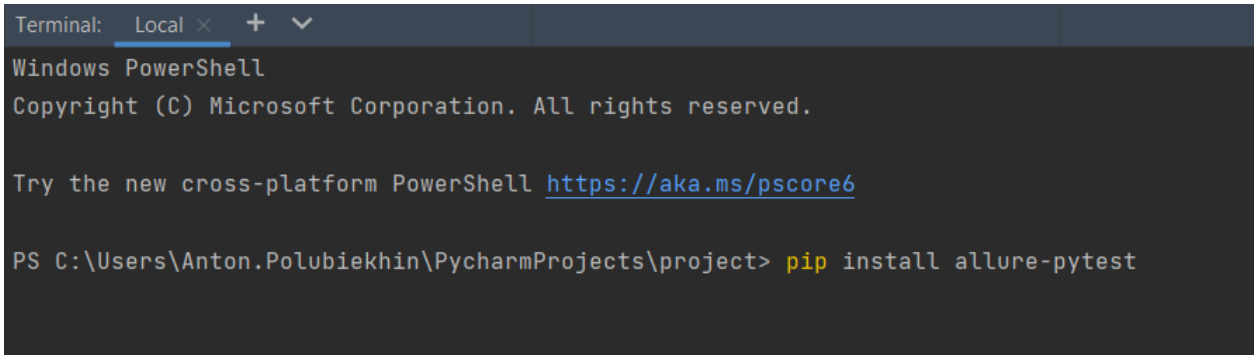
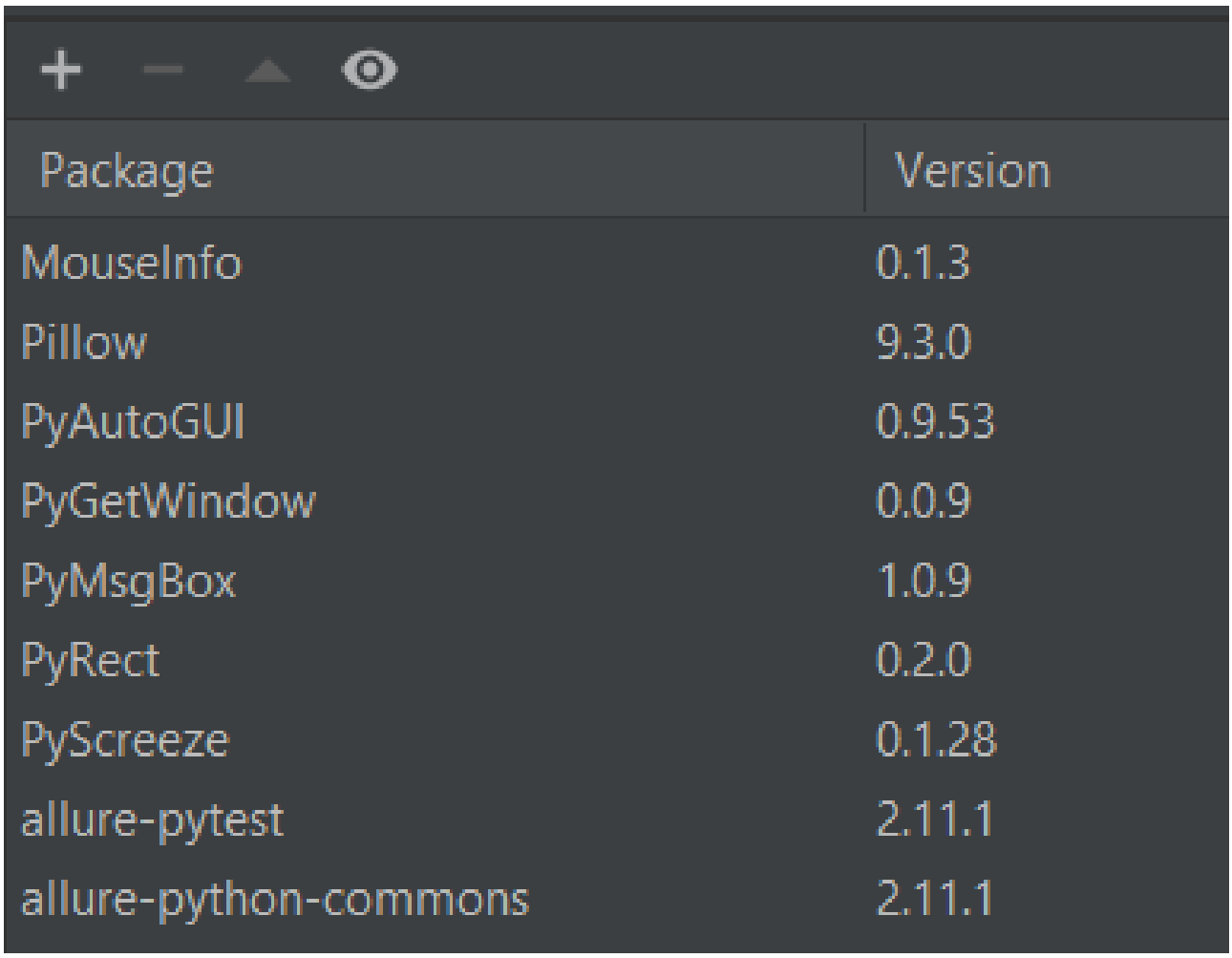


Рисунок 3.9 – Завантаження фреймворку Allure

Після того, як усі потрібні фреймворки були завантажені, їх наявність можна перевірити, перейшовши у меню File, далі вибрати пункт Settings, перейти до пункту Project: назва проєкту і вибрати interpreter, де будуть відображені стандартні пакети, які встановлені, і пакети, які були встановлені за допомогою команди pip і термінала (рис. 3.10).

Запускати тести можна окремо, за допомогою функції pytest, яка знаходиться збоку від кожної функції (рис. 3.11).



Package	Version
MouseInfo	0.1.3
Pillow	9.3.0
PyAutoGUI	0.9.53
PyGetWindow	0.0.9
PyMsgBox	1.0.9
PyRect	0.2.0
PyScreeze	0.1.28
allure-pytest	2.11.1
allure-python-commons	2.11.1

Рисунок 3.10 – Завантажені пакети

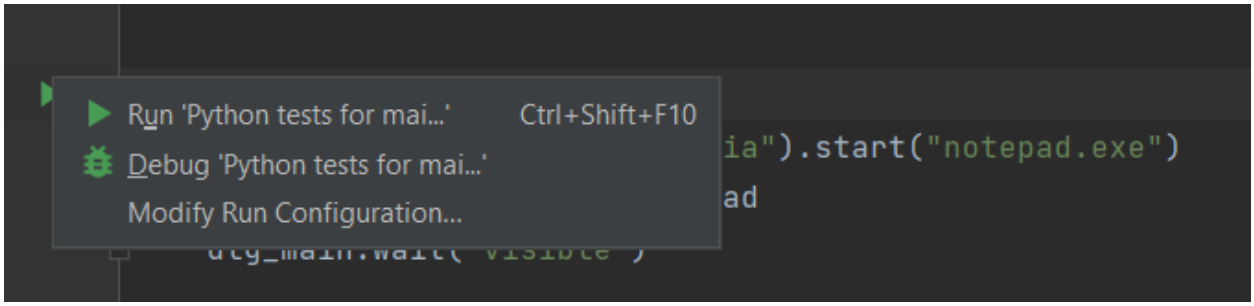


Рисунок 3.11 – Можливість запуску тестів

Функціональний тест для функції «Save As», який перевіряє функцію зберігання файлу (рис. 3.12, рис. 3.13).

```python
def test_functional_save_as_feature():
    app = Application(backend="uia").start("notepad.exe")
    app.UntitledNotepad.type_keys("Anton")
    app.UntitledNotepad.menu_select("File->Save As")
    dlg_save = app.UntitledNotepad.SaveAs
    dlg_save.wait('visible')
    pyautogui.write("Test2")
    pyautogui.press("enter")
    dlg_new_save = app.window(title='Test2.txt - Notepad')
    dlg_new_save.wait('visible')
```

Рисунок 3.12 – Функціональний тест для функції Save As

```
main.py::test_functional_save_as_feature PASSED                    [100%]

============================== 1 passed in 2.54s ==============================
```

Рисунок 3.13 – Час виконання тесту

Функціональний тест для функції «Font», який перевіряє, що функція зміни шрифту працює коректно (рис. 3.14, рис. 3.15).

```
main.py::test_functional_font_feature PASSED                       [100%]

============================== 1 passed in 15.37s ==============================
```

Рисунок 3.14 – Час виконання тесту

У тесті для перевірки шрифту робляться скріншоти, щоб перевірити правильність виконання тесту (рис. 3.16, рис. 3.17).

```python
def test_functional_font_feature():
    app = Application(backend="uia").start("notepad.exe")
    app.UntitledNotepad.type_keys("Anton")
    pic = pyautogui.screenshot()
    pic.save('Screenshot1.png')
    for i in "Anton":
        pyautogui.press('backspace')
    app.UntitledNotepad.menu_select("Format->Font...")
    pyautogui.press('backspace')
    pyautogui.write('Algerian')
    pyautogui.press('enter')
    app.UntitledNotepad.type_keys("Anton")
    pic2 = pyautogui.screenshot()
    pic2.save('Screenshot2.png')
```

Рисунок 3.15 – Функціональний тест для функції Font

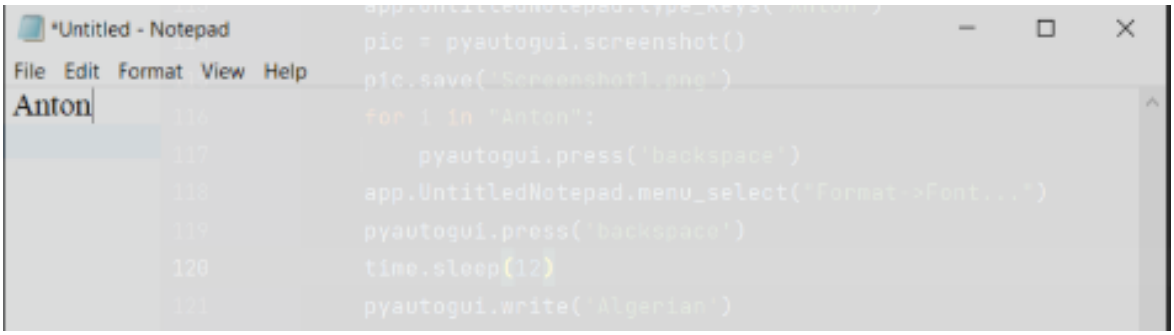


Рисунок 3.16 – Шрифт Times New Roman

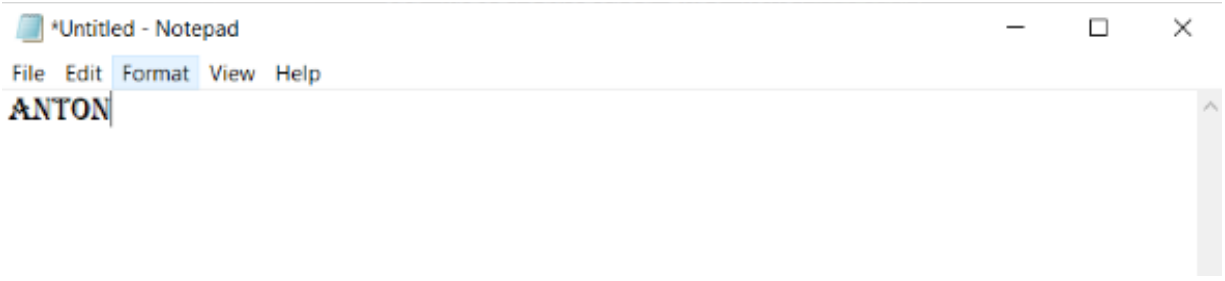


Рисунок 3.17 – Шрифт Algerian

Інтеграційний тест для функції «Font», який перевіряє, що діалог зміни шрифту відкривається за допомогою меню і гарячої клавіши (рис. 3.18, рис. 3.19).

```
def test_integration_font_feature():
    app = Application(backend="uia").start("notepad.exe")
    app.UntitledNotepad.menu_select("Format->Font...")
    dlg_font = app.UntitledNotepad.Font
    dlg_font.wait('visible')
    app.UntitledNotepad.Font.Cancel.click()
    pyautogui.hotkey('ctrl', 'p')
    dlg_font.wait('visible')
```

Рисунок 3.18 – Інтеграційний тест для функції Font

```
main.py::test_integration_font_feature PASSED                    [100%]

============================== 1 passed in 5.23s ===============================

Process finished with exit code 0
```

Рисунок 3.19 – Час виконання тесту

Функціональний тест для функції «Print», який перевіряє, що функція друку працює коректно (рис. 3.20).

Димовий тест для функції «Save As», який перевіряє, що діалог функції зберігання працює коректно (рис. 3.21, рис. 3.22).

```python
def test_functional_print_feature():
    app = Application(backend="uia").start("notepad.exe")
    app.UntitledNotepad.menu_select("File->Print")
    dlg_print = app.UntitledNotepad.Print
    dlg_print.wait('visible')
    app.UntitledNotepad.Print.Cancel.click()
    pyautogui.hotkey('ctrl', 'p')
    pyautogui.press('enter')
    pyautogui.write("pdftest0")
    pyautogui.press('enter')
    app.UntitledNotepad.menu_select("File->Open")
    dlg_open = app.UntitledNotepad.Ope
    dlg_open.wait('visible')
    pyautogui.write('pdftest0.pdf')
    pyautogui.press('enter')
    dlg_new_open = app.window(title='pdftest0.pdf - Notepad')
    dlg_new_open.wait('visible')
```

Рисунок 3.20 – Функціональний тест для функції Print

```python
def test_smoke_save_as_feature():
    app = Application(backend="uia").start("notepad.exe")
    app.UntitledNotepad.menu_select("File->Save As")
    dlg_save = app.UntitledNotepad.SaveAs
    dlg_save.wait('visible')
```

Рисунок 3.21 – Димовий тест для функції Save As

```
main.py::test_smoke_save_as_feature PASSED                               [100%]

============================== 1 passed in 2.92s ===============================

Process finished with exit code 0
```

Рисунок 3.22 – Час виконання тесту

Автоматизований тест для функції «Open», який перевіряє, що всі поля у діалозі «Open» відображаються (рис. 3.23, рис. 3.24).

```
def test_auto_open_feature():
    app = Application(backend="uia").start("notepad.exe")
    app.UntitledNotepad.menu_select("File->Open")
    dlg_open = app.UntitledNotepad.Open
    dlg_open.Pane.wait('visible')
    dlg_open.Static.wait('visible')
    dlg_open.ComboBox.wait('visible')
    dlg_open.ComboBox2.wait('visible')
    dlg_open.Static3.wait('visible')
    dlg_open.ComboBox3.wait('visible')
    dlg_open.Open.wait('visible')
    dlg_open.Cancel.wait('visible')
    dlg_open.Pane5.wait('visible')
    dlg_open.Pane6.wait('visible')
    dlg_open.Pane7.wait('visible')
    dlg_open.Pane9.wait('visible')
    dlg_open.Pane10.wait('visible')
    dlg_open.Pane11.wait('visible')
    dlg_open.TitleBar.wait('visible')
```

Рисунок 3.23 – Автоматизований тест для функції Open

```
main.py::test_auto_open_feature PASSED                                   [100%]

============================== 1 passed in 31.78s ==============================

Process finished with exit code 0
```

Рисунок 3.24 – Час виконання тесту

3.4 Порівняльний аналіз досліджених методів тестування десктопних застосунків

Для порівняння методів тестування були вибрані наступні критерії:

– час створення;

– можливість використання тесту на іншому ноутбуці;

– час виконання.

Час створення відіграє важливу роль, занадто довге стоврення тестів займає час тестувальника, впливає на вартість проєкту і на його якість. Градація балів буде від 1 – 3, де 1 визначає нагірший показник, а 3 – найкращий.

Можливість використання тестів на різних машинах, є великим плюсом під час розробки тестів, коли вони не залежать від однієї машини. Для цього критерію буде два можливі варіанти балів, 0 – коли є залежність від середовища, і 3 – коли залежності немає.

Час виконання є останім критерієм, де градація балів буде від 1 – 3, де 1 визначає нагірший показник, а 3 – найкращий.

Час створення тестів для десктопного застосунку, використовуючи pywinauto (табл. 3.1).

Таблиця 3.1 – Критерій: час створення тестів

Функція	**Димове**	**Функціональне**	**Інтеграційне**	**Автоматизоване**
Основна сторінка	2	2	2	2
Save As	3	1	2	1
Open	3	3	2	3
Print	3	1	2	3
Font	3	1	2	3

Середнє значення:

– для димового тестування 2,8;

– для функціонального 1,6;

– для інтеграційного 2;

– для автоматизованого 2,4.

Критерій можливості використання тестів на різних машинах (табл. 3.2).

Таблиця 3.2 – Критерій: можливість використання тестів на різних машинах

Функція	**Димове**	**Функціональне**	**Інтеграційне**	**Автоматизоване**
Основна сторінка	3	3	3	3
Save As	3	3	3	3
Open	3	0	3	3
Print	3	3	3	0
Font	3	3	3	3

Середнє значення:

– для димового тестування 3;

– для функціонального 2,4;

– для інтеграційного 3;

– для автоматизованого 2,4.

Критерій за часом виконання тестів (табл. 3.3).

Середнє значення:

– для димового тестування 3;

– для функціонального 1,8;

– для інтеграційного 2,8;

– для автоматизованого 2.

Таблиця 3.3 – Критерій: час виконання

Функція	**Димове**	**Функціональне**	**Інтеграційне**	**Автоматизоване**
Основна сторінка	3	3	3	2
Save As	3	2	3	2
Open	3	2	2	2
Print	3	1	3	2
Font	3	1	3	2

Фінальний результат:

– димове тестування 8,8;

– функціональне тестування 5,8;

– інтеграційне тестування 7,8;

– автоматизоване тестування 6,8.

За результатами аналізу отриманих результатів димвое тестування виявилось більш продуктивним через швидкість виконання тестів, а також через низьку складність створення тестів. Інтеграційне тестування показало гірший результат, через час створення тестів.

Кожен метод тестування, наразі являється провідним і часто використовується на різних проєктах. У майбутньому використання димового тестування, у більшій мірі через швидкість тестування, стане пріоритетним.

3.5 Перспективи подальшої роботи

Проведене дослідження і отримані результати вказують на те, що функціональне, димове, інтеграційне і автоматизоване тестування є актуальними на даний час.

Результати дослідження можна вдосконалити, провівши аналіз використання конкретного виду тестування на реальних проєктах. Також існує можливість додати додаткові вхідні дані і розширити критерії оцінювання.

Існує багато різних видів тестування, які можно проаналізувати. Звісно, не всі види тестування є сенс аналізувати, як, наприклад, вид тестування, яке може будет проаналізоване – санітарне тестування.

Має перспективу проведення такого ж аналізу для веб і мобільних застосунків, через те, що мобільні і вебплатформи займають більшу частину ринку.

ВИСНОВКИ

У рамках кваліфікаційної роботи були вивчені та порівняні функціональний, інтеграційний, димовий і автоматизований методи тестування десктопних застосунків, які дозволять проводити тестування з найбільшою продуктивністю. Розроблено фреймворк автоматизації для застосунка Notepad.

Виконано всі поставлені задачі:

– прокласифіковані та проаналізовані існуючі методи тестування;

– вибрані та вивчені методи тестування десктопних застосунків;

– визначені фреймворки для реалізації тестів;

– вибране і налаштоване середовище розробки під Python, PyCharm IDE;

– реалізовані тести для десктопного застосунку Notepad;

– визначені критерії для оцінювання методів тестування десктопних застосунків;

– проведено порівняльний аналіз методів тестування десктопних застосунків, який показав, димове тестування є найбільш продуктивним поміж проаналізованих видів тестування.

Результати роботи апробовано у вигляді тез доповідей під час VIII Міжнародної науково-практичної конференції «The main prospects for the development of science in modern life» [53].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Sankar R. Burpsuite – A Beginner’s Guide For Web Application Security or Penetration Testing. URL: https://kalilinuxtutorials.com/ burpsuite/ (дата звернення 03.10.2022).

2. ORDINATUS. 5 инструментов непрерывной интеграции. URL: http://ordinatus.ru/5-instrumentov-nepreryvnoj-integracii (дата звернення 15.10.2022).

3. Кріспін Л. (2011). Гнучке тестування: практичне керівництво для тестувальників ПО та гнучких команд.

4. Ромашин Р.І. (2015) Веб безпека.

5. Tvoroshenko, I. (2019). Development of models of spatial analysis of status of interactive processes of complex systems.

6. Майерс Г., Баджетт Т., Сандлер К. (2008). Искусство тестирования программ.

7. Бейзер Б. (2004). Тестирование чёрного ящика. Технологии функционального тестирования программного обеспечения и систем.

8. Tvoroshenko, I., & Dziubenko, M. (2020). Modern methods of analysis of the movement scheme using video detection of vehicles.

9. Тестування на проникнення. URL: https://uk.myservername.com/ beginners-guide-web-application-penetration-testing(дата звернення 03.10.2022).

10. Ляшко, С.Ю. (2019). Автоматизоване тестування веб-додатків.

11. Дудка В.О. (2021) Система автоматичного тестування веб-додатків.

12. Удовенко С.Г. (2019) Використання шаблонів автоматичного тестування в проектах з розробки веб-додатків.

13. Порошин С.М. (2016) Методологія проведення pen-тестування веб-додатків.

14. Попович Х. (2015) Розробка стратегії тестування для веб-додатків.

15. Бондаренко О.В. (2020) Дослідження інструментів тестування веб-додатків.

16. Слободян Р. (2020) Автоматизоване тестування веб-додатків шляхомвзаємодії з користувацьким інтерфейсом.

17. Коцюбинський В.Ю. (2019) Аналіз ефективних сучасних інструментів для тестування web-додатків.

18. Мороз Б.П. (2021) Система тестування вмісту веб-сайтів.

19. Твердохліб І. (2019) Методики тестування безпеки веб-додатку на основі інформації з відкритих джерел.

20. Бродський Ю.Б. (2018) Методологічні аспекти структурно-функціонального програмування в економіці.

21. Дубовік Я.С. (2019) Необхідність використання сучаснихтехнологій моделювання бізнес-процесів.

22. Регресійне тестування. URL: https://qalearning.com.ua/theory/lectures/material/regression-testing-crossbrowser/ (дата звернення 02.10.2022).

23. Модульне тестування. URL:https://msn.khnu.km.ua/pluginfile.php/208290/mod_resource/content/2/%D0%9B%D0%A0%20%E2%84%964.pdf. (дата звернення 30.09.2022).

24. Джек Фолк, Сем Канер, Енг Кек Нгуєн (2001). Тестування програмного забезпечення.

25. Gorokhovatskyi, V. A., Rusakova, N., & Tvoroshenko, I. S. (2020). The application of image analysis methods and predicate logic in applied problems of magnetic monitoring. *Telecommunications and Radio Engineering*, *79*(20).

26. Гаркавий С.О (2020). Розробка фреймворку для автоматизації тестування інтерфейсу веб-сайту.

27. Tvoroshenko I., and Gorokhovatskyi V. (2022) The Application of Hybrid Intelligence Systems for Dynamic Data Analysis, *International Journal of Engineering and Information Systems*, 6(2), pp. 40-48.

28. Померанцева С.А (2020). Розробка фреймворку для автоматизації тестування інтерфейсу веб-сайту.

29. Tvoroshenko, I., & Tkachenko, D. (2020). Mechanisms of image classification based on descriptors of local features.

30. Daradkeh, Y. I., & Tvoroshenko, I. (2020). Technologies for making reliable decisions on a variety of effective factors using fuzzy logic. *International Journal of Advanced Computer Science and Applications*, *11*(5).

31. Asaad Ma. Babker, Abd Elgadir A. Altoum, Irina Tvoroshenko, and Vyacheslav Lyashenko (2019) Information Technologies of the Processing of the Spaces of the States of a Complex Biophysical Object in the Intellectual Medical System HEALTH, *International Journal of Advanced Trends in Computer Science and Engineering*, 8(6), pp. 3221-3227.

32. Кучеренко, Є. І., Творошенко, І. С., Анопрієнко, Т. В. (2016) Моделювання та оцінювання станів складних об'єктів із застосуванням формальної логіки. *Системи обробки інформації*, (2), 76-82.

33. Кучеренко, Е. И., Филатов, В. А., Творошенко, И. С., & Байдан, Р. Н. (2005). Интеллектуальные технологии в задачах принятия решений технологических комплексов на основе нечеткой интервальной логики. *Восточно-Европейский журнал передовых технологий*, (2), 92-96.

34. Кобилін, О. А., & Творошенко, І. С. (2021). Методи цифрової обробки зображень.

35. Творошенко, І. С., & Табашник, В. А. (2018). Розробка просторової моделі геоінформаційної підтримки людей з обмеженими можливостями, що пересуваються на інвалідних колясках, у місті Харків. *Збірник наукових праць Харківського національного університету Повітряних Сил*, (1), 122-128.

36. Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., & Al-Dhaifallah, M. (2022). Classification of Images Based on a System of Hierarchical Features. *Computers, Materials & Continua*, *72*(1), 1785-1797.

37. Гороховатський, В. О., & Творошенко, І. С. (2022). Аналіз багатовимірних даних за описом у формі множини компонент.

38. Ayaz, A. M., Gorokhovatskyi, V., Tvoroshenko, I., Vlasenko, N., & Khalid, M. S. (2021). The Research of Image Classification Methods Based on the Introducing Cluster Representation Parameters for the Structural Description.

39. Tvoroshenko, I. S., & Kramarenko, O. O. (2019). Software determination of the optimal route by geoinformation technologies. *Radio Electronics, Computer Science, Control*, (3), 131-142.

40. Творошенко, И. С. (2010). Анализ процессов принятия решений в интеллектуальных системах. *Системи обробки інформації*, (2), 248-253.

41. Кучеренко, Е. И., Творошенко, И. С. (2010) Прикладные аспекты моделирования нечетких процессов в сложных системах. *Збірник наукових праць Харківського університету Повітряних сил*, (1), 127-131.

42. Кучеренко, Е. И., Корниловский, А. В., Творошенко, И. С. (2010) О методах настройки функций принадлежности в нечетких системах. *Системы управления, навигации и связи*, (1), 13.

43. Tvoroshenko I., and Zarivchatskyi R. (2020) Analysis of existing methods for searching object in the video stream, Abstracts of VI International Scientific and Practical Conference «About the problems of science and practice, tasks and ways to solve them» (October 26-30, 2020). Milan, Italy, pp. 500-505.

44. Tvoroshenko I.S., and Gorokhovatsky V.O. (2019) Modification of the branch and bound method to determine the extremes of membership functions in fuzzy intelligent systems, Telecommunications and Radio Engineering, 78(20), pp. 1857-1868.

45. Творошенко, І. С. (2018). Особливості застосування сучасних принципів штучного інтелекту до розробки ефективних механізмів моделювання складних систем. *Science and Technology of the Present Time: Priority Development Directions of Ukraine and Poland*, 118-121.

46. Творошенко, І. С., Шевченко, А. Р. (2018) Удосконалення просторової мережі навчальних закладів міста Сєвєродонецька на основі геоінформаційного аналізу. *Системи обробки інформації*, (1), 46-52.

47. Matarneh Rami, Tvoroshenko Irina, and Lyashenko Vyacheslav (2019) Improving Fuzzy Network Models For the Analysis of Dynamic Interacting Processes in the State Space, International Journal of Recent Technology and Engineering, 8(4), pp. 1687-1693.

48. Гороховатський В.О., Творошенко І.С., Чмутов Ю.В. (2022) Застосування систем ортогональних функцій для формування простору ознак у методах класифікації зображень. *Сучасні інформаційні системи*, 6 (3), С. 5–12.

49. Lyashenko V., Mustafa S.K., Tvoroshenko I., and Ahmad M.A. (2020) Methods of Using Fuzzy Interval Logic During Processing of Space States of Complex Biophysical Objects, *International Journal of Emerging Trends in Engineering Research*, 8(2), pp. 372-377.

50. Tvoroshenko, I., & Babochkin, O. (2021). Object identification method based on image keypoint descriptors.

51. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Zeghid M. (2022) Cluster representation of the structural description of images for effective classification, *Computers, Materials & Continua*, 73 (3), pp. 6069–6084.

52. Гороховатський В., Творошенко І., Сидоренко Д. (2021) Класифікація зображень із використанням кластерного подання. Міжнародний науковий симпозіум «Інтелектуальні рішення-С». Обчислювальний інтелект (результати, проблеми, перспективи). Теорія прийняття рішень: праці міжн. Наук. Симпозіуму (Вересень 29, 2021). Київ – Ужгород, С. 44-45.

53. Полубєхін А. (2022) Вивчення методів тестування десктопних застосунків, *Abstracts of XXXVI International Scientific and Practical Conference «The main prospects for the development of science in modern life» (September 13 – 16, 2022). Warsaw, Poland*, pp. 362-366.