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)

ДОСЛІДЖЕННЯ ТА ПОРІВНЯННЯ МЕТОДІВ ТЕСТУВАННЯ МОБІЛЬНИХ ЗАСТОСУНКІВ
(тема)

Виконав:
студент 2 курсу, групи ІТКСм-20-1
Кузнецов М.В.
(прізвище, ініціали)

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

Тип програми освітньо-професійна

Освітня програма Консолідована інформація
(повна назва освітньої програми)

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

Допускається до захисту

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

2021 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 Консолідована інформація
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _______________
(підпис)
«____»______________ 2021 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Кузнецову Максиму Вікторовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Дослідження та порівняння методів тестування мобільних застосунків

затверджена наказом по університету від «22» жовтня 2021 року № 1575Ст.

2. Термін подання студентом роботи до екзаменаційної комісії 22 листопада 2021 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, теоретичні методи тестування мобільних застосунків, перелік використовуваних програмних засобів: інтерпретатор мови Python 3.7, середовище розробки PyCharm, фреймворки Pytest та Unittest, застосунок Appium, застосунок Youtube.

4. Перелік питань, що потрібно опрацювати в роботі

1. Дослідити роль тестування у життєвому циклі розроблення мобільних застосунків.
2. Дослідити існуючі методи тестування та провести їх класифікацію.
3. Змоделювати процес тестування за допомогою стандарту IDEF3.
4. Дослідити обрані методи тестування мобільних застосунків.
5. Розробити проєкт мобільної автоматизації обраного застосунку.
6. Протестувати мобільний застосунок обраними методами тестування.
7. Визначити перспективи подальшої роботи.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність, об'єкт та мета, постановка задачі дослідження, етапи дослідження, результати тестування, висновки, перспективи подальшої роботи, апробація результатів дослідження.

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	Доцент Бєлова Н.В.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	22.10.2021	
2	Аналіз завдання, підбір літератури	23.10.21-13.11.21	
3	Аналіз літератури з досліджуваної проблеми	14.11.21-17.11.21	
4	Дослідження обраних методів тестування	18.11.21-20.11.21	
5	Застосування обраних методів тестування	21.11.21-23.11.21	
6	Програмна реалізація	24.11.21-25.11.21	
7	Оформлення пояснювальної записки	26.11.21-27.11.21	
8	Перевірка на плагіат	28.11.2021	
9	Рецензування	28.11.2021	
10	Підготовка презентації та доповіді	29.11.21-30.11.21	
11	Занесення роботи в електронний архів	01.12.2021	
12	Попередній захист кваліфікаційної роботи	01.12.2021	

Дата видачі завдання 22 жовтня 2021 р.

Студент ______________________________
(підпис)

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

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 86 с., 4 табл., 40 рис., 46 джерел.

ТЕСТУВАННЯ, МОБІЛЬНІ ЗАСТОСУНКИ, МЕТОДИ ТЕСТУВАННЯ, ФУНКЦІОНАЛЬНЕ, ДИМОВЕ, АВТОМАТИЗОВАНЕ, СЕРЕДОВИЩЕ РОЗРОБКИ.

Об’єктом роботи є процес автоматизації тестування мобільного застосунку, заснованого на принципі взаємодії користувача з медіа-контентом.

Метою роботи є дослідження обраних методів тестування мобільного застосунку та виявлення найкращого щодо вирішення поставлених задач.

Використано методи тестування мобільних застосунків. Проведено дослідження ролі тестування у життєвому циклу розроблення мобільних застосунків, проаналізовано сучасний стан застосування методів тестування, та змодельовано процес тестування за допомогою стандарту IDEF3.

У результаті роботи здійснена програмна реалізація проєкту мобільної автоматизації застосунку.

TESTING, MOBILE APPLICATIONS, TESTING METHODS, FUNCTIONAL, SMOKE, AUTOMATED, DEVELOPMENT ENVIRONMENT.

The object of the work is the process of automating the testing of a mobile application based on the principle of user interaction with media content.

The aim of the work is to study the selected methods of testing a mobile application and identify the best for solving the tasks.

Methods of testing mobile applications are used. A study of the role of testing in the life cycle of mobile application development was conducted, the current state of application of testing methods was analyzed, and the testing process was modeled using the IDEF3 standard.

As a result of work the software realization of the project of mobile automation of application is carried out

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 7
Вступ 8
1 Аналіз існуючих методів тестування 10
1.1 Етапи розвитку напрямку тестування 10
1.2 Роль тестування під час розроблення мобільних застосунків 11
1.2.1 Роль тестування у формуванні вимог до проєкту 13
1.2.2 Роль тестування на стадії дизайну 14
1.2.3 Роль тестування на стадії розроблення 15
1.2.4 Роль тестування на передзаключній стадії та на стадії розгортання програмного забезпечення 16
1.3 Аналіз існуючих методів тестування та розроблення програмного забезпечення 17
1.3.1 Особливості методу оброблення водоспадної моделі 17
1.3.2 Особливості методу оброблення V-подібної моделі 19
1.3.3 Особливості методу оброблення ітераційної інкрементальної моделі 20
1.3.4 Особливості методу оброблення спіральної моделі 21
1.3.5 Особливості методу оброблення гнучкої моделі 23
1.4 Аналіз літературних джерел щодо апробації результатів застосування методів тестування мобільних застосунків 25
1.5 Постановка задачі дослідження 27
2 Дослідження вибраних методів тестування мобільних застосунків 29
2.1 Моделювання процесу тестування мобільних застосунків за допомогою стандарту IDEF3 29
2.2 Дослідження методу тестування «Функціональне тестування» 35
2.3 Дослідження методу тестування «Димове тестування» 38
2.4 Дослідження методу тестування «Тестування білого ящика» 40

2.5 Аналіз можливих непередбачуваних ситуацій під час виконання досліджуваних методів тестування 43

3 Застосування методів тестування мобільного застосунку щодо вибраної предметної області 49

3.1 Вибір інструментальних засобів для реалізації вибраних методів 49

3.2 Етапи програмної реалізації вибраних методів тестування мобільних застосунків 52

3.3 Впровадження методів тестування мобільних застосунків стосовно вибраної предметної області 62

3.4 Порівняльний аналіз досліджених методів тестування мобільних застосунків 73

3.5 Перспективи подальшої роботи 78

Висновки 80

Перелік джерел посилання 82

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення

QA – Quality Assurance

QC – Quality Control

ISTQB – International Software Testing Qualifications Board (міжнародна комісія з сертифікації тестування програмного забезпечення)

Юзабіліті-аудит – метод перевірки інтерфейсу з точки зору його зручності для користувачів. Головними «експертами» в юзабіліті-аудит стають представники аудиторії, для яких був створений проєкт

IDEF – I-CAM DEFinition або Integrated DEFinition

API – Application Programming Interface

IDE – Integrated development environment

ВСТУП

У вік технологій щодня створюється багато нових програм або сервісів, які покликані виконати ті чи інші функції. На цьому фоні стає очевидним, що кінцевий користувач може вибрати будь-яку програму або програмне забезпечення, адже вибір дуже великий. Це, зазвичай, робиться досвідченим шляхом, або за рекомендацією інших користувачів. У зв'язку з цим розробники вирішили встановити певний стандарт якості програмного забезпечення мобільних застосунків, який перевіряється спеціальною процедурою, що називається «тестування програмного забезпечення».

У процесі тестування виявляються дефекти, які згодом були б незручні користувачам, програмне забезпечення мобільних застосунків удосконалюється, приносячи користь усім. Усвідомивши той факт, що платформи високої якості розробленого програмного забезпечення мобільних застосунків – це реальний шлях «обійти» конкурентів, багато компаній у всьому світі вкладають все більше коштів у забезпечення якості своїх продуктів, створюючи власні групи та відділи, що займаються тестуванням, або передаючи тестування своїх продуктів стороннім організаціям.

Тестування програмного забезпечення мобільних застосунків – це основний вид діяльності тестувальника, який полягає в перевірці збігу та відповідності реальної поведінки програми з очікуваним, здійснюється шляхом проведення певного набору тестів над програмним забезпеченням у певному середовищі.

Якщо розмежовувати тестування та забезпечення якості, то тестування є однією із застосованих технік контролю якості над продуктом, яка включає:

- планування робіт із тестування;
- проєктування тестів;
- тестування програмного забезпечення мобільних застосунків;
- аналіз отриманих результатів робіт із тестування.

На практиці тестування програмного забезпечення мобільних застосунків полягає у дослідженні наданого продукту, пошуку дефектів, ґрунтуючись на специфікації та досвіді працівника.

Досить часті випадки, коли тестування проводитися тільки з досвіду тестувальника, таке тестування називають «дослідницьким», а іноді «ad-hoc» тестуванням. Головною перевагою такого тестування є те, що працівник наданий сам собі і вся робота будується на його припущенні про якісний продукт. З іншого боку, таке тестування може призвести до великих часових витрат, спотвореного кінцевого результату, особливо, у випадках відсутності достатнього досвіду у тестувальника.

Щоб уникнути більшості негативних наслідків, тестування програмного забезпечення мобільних застосунків передбачає досить складний процес планування забезпечення якості програмного продукту.

1 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ТЕСТУВАННЯ

1.1 Етапи розвитку напрямку тестування

Протягом десятиліть розвитку розроблення програмного забезпечення до питань тестування і забезпечення якості підходили дуже і дуже по-різному. Можна виділити кілька основних «епох тестування».

У 50-60-х роках 20 століття процес тестування був гранично формалізований, відділений від процесу безпосередньої розроблення програмного забезпечення. Фактично тестування представляло собою скоріше налагодження програм. Існувала концепція так званого «вичерпного тестування» – перевірки всіх можливих шляхів виконання коду з усіма можливими вхідними даними. Але дуже скоро було з’ясовано, що вичерпне тестування неможливо, тому що кількість можливих шляхів і вхідних даних велика, при такому підході складно знайти проблеми в документації.

У 70-х роках 20 століття фактично народилися дві фундаментальні ідеї тестування: тестування спочатку розглядалося як процес доказу працездатності програми в деяких заданих умовах (позитивне тестування), а потім – строго навпаки: як процес доказу непрацездатності програми в деяких заданих умовах (негативне тестування). Це внутрішнє протиріччя не тільки не зникло з часом, але і в наші дні багатьма авторами зовсім справедливо відзначається як дві взаємодоповнюючі мети тестування [1].

У 80-х роках 20 століття відбулася ключова зміна місця тестування в розробці програмного забезпечення: замість однієї з фінальних стадій створення проєкту тестування стало застосовуватися протягом усього циклу розроблення, що дозволило в багатьох випадках не тільки швидко виявляти й усувати проблеми, але навіть передбачати і запобігати їх появі. В цей же період часу відзначено бурхливий розвиток і формалізація методологій тестування і поява перших елементарних спроб автоматизувати тестування.

У 90-х роках 20 століття відбувся перехід від тестування до більш всеосяжного процесу, який називається «забезпечення якості», та який охоплює весь цикл розроблення програмного забезпечення і зачіпає процеси планування, проєктування, створення і виконання тест-кейсів, підтримку наявних тест кейсів і тестових оточень. Тестування вийшло на якісно новий рівень, який природним чином привів до подальшого розвитку методологій, появи досить потужних інструментів управління процесом тестування і інструментальних засобів автоматизації тестування, вже цілком схожих на своїх нинішніх нащадків.

З 2010 року розвиток тестування тривав в контексті пошуку все нових і нових шляхів, методологій, технік і підходів до забезпечення якості. Серйозний вплив на розуміння тестування мала поява гнучких методологій та таких підходів, як «розроблення під керуванням тестування». Автоматизація тестування вже сприймалася як звичайна невід’ємна частина більшості проєктів, а також стали популярні ідеї про те, що на чолі процесу тестування слід ставити не відповідність програми вимогам [1], а її здатність надати кінцевому користувачеві можливість ефективно вирішувати свої завдання.

1.2 Роль тестування під час розроблення мобільних застосунків

Якщо успішний запуск програмного продукту є для замовника більш пріоритетною метою, ніж висока швидкість розроблення і економне витрачання коштів, тестування повинно стати невід’ємною частиною кожної фази життєвого циклу розроблення програмного забезпечення. З огляду на складність сучасних програм, складно уявити собі ситуацію, коли процес розроблення зовсім обходиться без будь-якого тестування. Але, тим не менше, час від часу можна зіткнутися з ситуацією, коли розробники програмного забезпечення зазнають невдачі з тієї причини, що тестування

включено в загальний процес розроблення тільки як одна з окремих стадій загального процесу, відособлена від загального циклу.

Виходячи з особливостей сучасного програмного забезпечення, таких, наприклад, як складна архітектура, нехтування тестуванням протягом життєвого циклу розроблення майже завжди негативно позначається на якості безпеки, продуктивності, або функціональності кінцевого продукту. Цілком закономірним виявляється питання про те, яким же саме чином компанія-розробник може забезпечити найкращу якість продукту, що розробляється [2].

Тестування, проведене на всіх етапах розроблення, дозволяє значно поліпшити якість, надійність і продуктивність системи. У ході тестування команда тестувальників пересвідчується в тому, що програмний продукт належним чином виконує всі задокументовані функції і не робить того, що не має робити.

Для забезпечення високої якості кінцевого продукту, критично важливим є включення тестування в життєвий цикл розроблення програмного забезпечення. Особливе значення має впровадження тестування саме на ранніх стадіях роботи над проєктом, оскільки такий підхід дозволяє значно знизити витрати на усунення виявлених помилок.

Основні переваги впровадження тестування в життєвий цикл програмного забезпечення:

– тестування, введене на ранніх стадіях розроблення, значно знижує вартість виправлення помилок;

– з огляду на особливості сучасного ринку, тільки високоякісні продукти володіють конкурентоспроможністю. Таким чином, вкладаючи ресурси в розроблення програмного забезпечення, потрібно докласти всіх зусиль, щоб упевнитися в тому, що кінцевий продукт зможе скласти конкуренцію вже існуючим рішенням. Тестування програмного забезпечення протягом усього циклу розроблення є одним з ключових чинників, що забезпечують належну якість кінцевого результату;

– середовище, що використовується при розробці програмного забезпечення відрізняється від тієї, в якій кінцевий продукт буде використовуватися. Тестування, таким чином, дозволяє випробувати програмного забезпечення в умовах, схожих з реальними.

Залежно від обраної методології розроблення програмного забезпечення, повний життєвий цикл може складатися з різної кількості етапів. З метою спрощення, розглянемо чотири основні стадії розроблення, характерні для будь-якого проєкту:

– створення вимог до проєкту;
– аналіз і дизайн;
– стадія розроблення;
– передзаключна стадія і розгортання.

1.2.1 Роль тестування у формуванні вимог до проєкту

Головним завданням на даному етапі є збір бізнес-вимог до кінцевого продукту. У загальних рисах такі вимоги виглядають наступним чином: хто саме буде користуватися застосунком, доступ до яких даних необхідний і яким чином вони будуть використовуватися [2].

На даній стадії проводиться тестування вимог. Основною його метою є виявлення помилок і невідповідностей у бізнес-логіці програмного забезпечення на самій ранній стадії життєвого циклу. Після того, як була складена повноцінна документація, команда тестувальників може оцінити її за такими критеріями:

– повнота;
– надмірність;
– однозначність;
– відсутність суперечливості;
– ранжирування;
– перевірюваність.

Ретельне тестування документації дозволяє виявляти помилки на ранніх етапах, що призводить до зниження вартості їх виправлення, а значить і загальних витрат на розроблення. Більш якісна документація знижує трудомісткість проєкту і скорочує загальний час на розроблення. Однозначні і повні бізнес-вимоги дозволяють команді розробників краще оцінити обсяг роботи і опрацювати технічне завдання. За рахунок ретельної деталізації та зниження ризикової складової проєкту, знижується загальна вартість розроблення.

1.2.2 Роль тестування на стадії дизайну

На даному етапі використовується документація, складена на попередній стадії. На її основі створюється макет мобільного застосунку, а також проєктується архітектура майбутнього програмного продукту.

Створення і тестування прототипу допомагає оцінити якість майбутнього програмного продукту і його комерційні перспективи. Вивчення прототипу на початковому етапі роботи над проєктом дозволяє внести необхідні зміни відповідно до поставлених цілей. Тестуючи програмний продукт вже на етапі створення прототипу, можна заощадити час і скоротити витрати, так як проєкт буде ретельно опрацьовано з урахуванням специфікації ще до того, як команда розробників приступить до написання коду. Команда тестувальників приділяє особливу увагу виявленню логічних помилок в прототипі, які можуть привести до збою всієї системи в разі їх міграції на наступні етапи розроблення. Крім цього, прототип порівнюється зі схожими за призначенням програмними продуктами, що дозволяє запропонувати варіанти щодо його поліпшення. Тестування прототипу дозволяє розрахувати приблизні витрати на кожному етапі створення продукту і вибрати найбільш ефективну методологію розроблення [3].

Юзабіліті-аудит прототипу дозволяє оцінити зручність використання майбутнього програмного продукту. Команда тестувальників ретельно вивчає прототип, наданий замовником і, в разі необхідності, пропонує рекомендації щодо його поліпшення. У результаті з'являється можливість підвищити зручність використання програмного продукту кінцевим користувачем. Таке тестування зазвичай проходить в кілька етапів:

– тестування грубого схематичного прототипу на початковій стадії проєктування;

– тестування прототипу середньої деталізації;

– тестування готового точного прототипу до затвердженого дизайном.

Однією з головних переваг юзабіліті-аудиту на ранньому етапі розроблення є значне зниження вартості виправлення помилок в порівнянні з більш пізніми стадіями [3].

1.2.3 Роль тестування на стадії розроблення

На даному етапі відбувається написання вихідного коду майбутнього програмного продукту. У залежності від методології розроблення існують, наприклад, такі етапи:

– тестування компонентів дозволяє ґрунтовно перевірити кожен компонент програмного забезпечення (об'єкт, модуль, клас, і т.д.) і переконатися в коректності його роботи. Таке тестування перевіряє функціонал, який додається в міру розроблення програмного забезпечення. Кожен компонент при цьому тестується ізольовано, в штучно створеному середовищі. В якості основи для проведення модульного тестування використовується список затверджених вимог до тестування. Команда тестувальників створює список тест-кейсів з описом відповідних кроків і очікуваних результатів. Також складається список сценаріїв використання програмного продукту, які описують послідовність дій користувача і

очікувану реакцію системи на них. Тестування компонентів проводиться під час розроблення кожного окремого модуля системи. Таким чином, у разі виявлення помилок, знадобиться зміна дизайну тільки конкретно модуля, що тестується, а не всієї системи в цілому. Цей вид тестування дозволяє виявити недоліки в технічному завданні або архітектурі програми, а також оцінити працездатність окремих частин продукту на кожному етапі розроблення;

– автоматизація тестування вводиться в тих випадках, коли тестувальник має справу з проєктом великих розмірів, що призводить до дуже великої кількості перевірок. Команда тестувальників створює тест-кейси для програми та чек-лист необхідних перевірок, на основі яких створюються функціональні тести. Ці тести запускаються щодня або автоматично в певний час доби, наприклад, вночі. При цьому автоматично генерується звіт про пройдені тестах. Команда тестування підтримує вже написані тести, а також створює нові у міру необхідності. В ході такого тестування виявляються помилки, які набагато складніше було б виявити вручну. Прискорюється процес і якість проведення складних тестів, пов’язаних з обчисленням, наприклад, перевірка складних формул. Збільшується точність і надійність тестів.

1.2.4 Роль тестування на передзаключній стадії та на стадії розгортання програмного забезпечення

Фінальна стадія життєвого циклу розроблення програмного забезпечення зазвичай складається з двох етапів: попереднє розгортання (beta-deployment) і кінцеве розгортання (final deployment). Стадія попереднього розгортання необхідна для того, щоб у команди тестувальників була можливість відловити помилки в продукті до того, як він буде випущений на ринок. Результати тестування можуть бути використані командою розробників для внесення останніх коригувань перед остаточним

розгортанням продукту. На цьому етапі команда тестувальників перевіряє коректність виправлення помилок, виявлених на попередніх стадіях, а також тестує стійкість програмного забезпечення до високих навантажень і злому [4].

1.3 Аналіз існуючих методів тестування та розроблення програмного забезпечення

Щоб краще розібратися в тому, як тестування співвідноситься з програмуванням і іншими видами проєктної діяльності, розглянемо моделі розроблення програмного забезпечення (як частини життєвого циклу програмного забезпечення) [4].

Вибір моделі розроблення програмного забезпечення серйозно впливає на процес тестування, так як визначають вибір стратегії, розклад, необхідні ресурси і т.д. У загальному випадку класичними можна вважати водоспадну, V-подібну, ітераційну інкрементальну, спіральну і гнучку моделі.

1.3.1 Особливості методу оброблення водоспадної моделі

Водоспадна модель зараз представляє швидше історичний інтерес, так як у сучасних проєктах практично непридатна. Вона передбачає одноразове виконання кожної з фаз проєкту, які, в свою чергу, суворо слідують один за одним (рис. 1.1). Дуже спрощено можна сказати, що в рамках цієї моделі в будь-який момент часу команді «видна» лише попередня і наступна фаза.

У реальному ж розробці програмного забезпечення доводиться «бачити весь проєкт цілком» і повертатися до попередніх фаз, щоб виправити недоробки або щось уточнити.

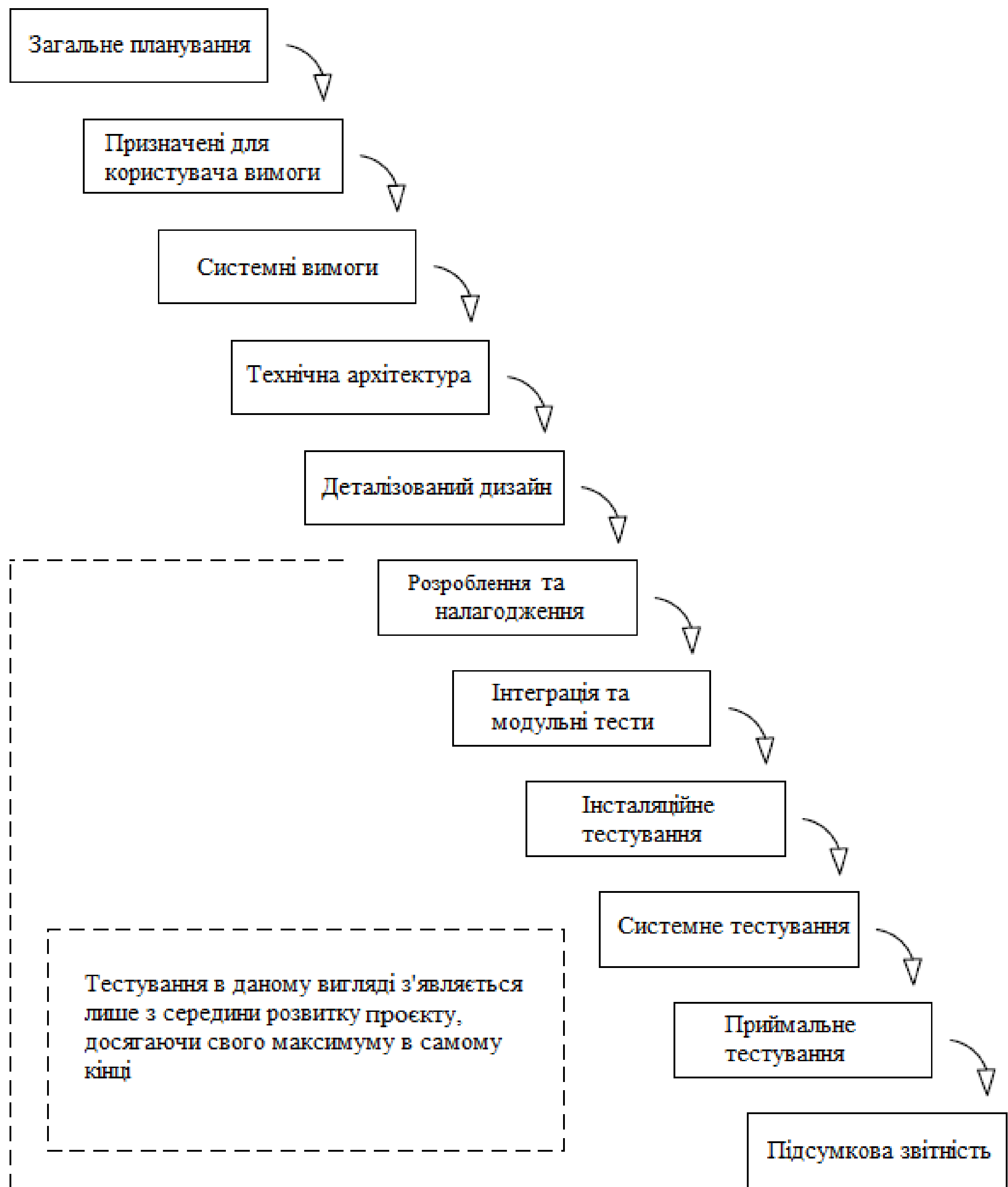

Рисунок 1.1 – Водоспадна модель розроблення програмного забезпечення

До недоліків водоспадної моделі прийнято відносити той факт, що участь користувачів програмного забезпечення в ній або не передбачено взагалі, або передбачено лише побічно на стадії одноразового збору вимог. З точки зору ж тестування ця модель погана тим, що тестування в явному вигляді з’являється тут лише з середини розвитку проєкту, досягаючи свого максимуму в самому кінці [5].

Проте водоспадна модель часто інтуїтивно застосовується при виконанні відносно простих завдань, а її недоліки стали прекрасним відправним пунктом для створення нових моделей. Також ця модель у вдосконаленому вигляді використовується на проєктах, де вимоги стабільні та можуть бути добре сформульовані на початку проєкту, наприклад, аерокосмічна галузь, медичне програмне забезпечення і т.д. [5].

1.3.2 Особливості методу оброблення V-подібної моделі

V-подібна модель є логічним розвитком водоспадної (рис. 1.2). Можна помітити, що в загальному випадку як водоспадна, так і V-подібна моделі життєвого циклу програмного забезпечення можуть містити один і той же набір стадій, але принципова відмінність полягає в тому, як ця інформація використовується в процесі реалізації проєкту.

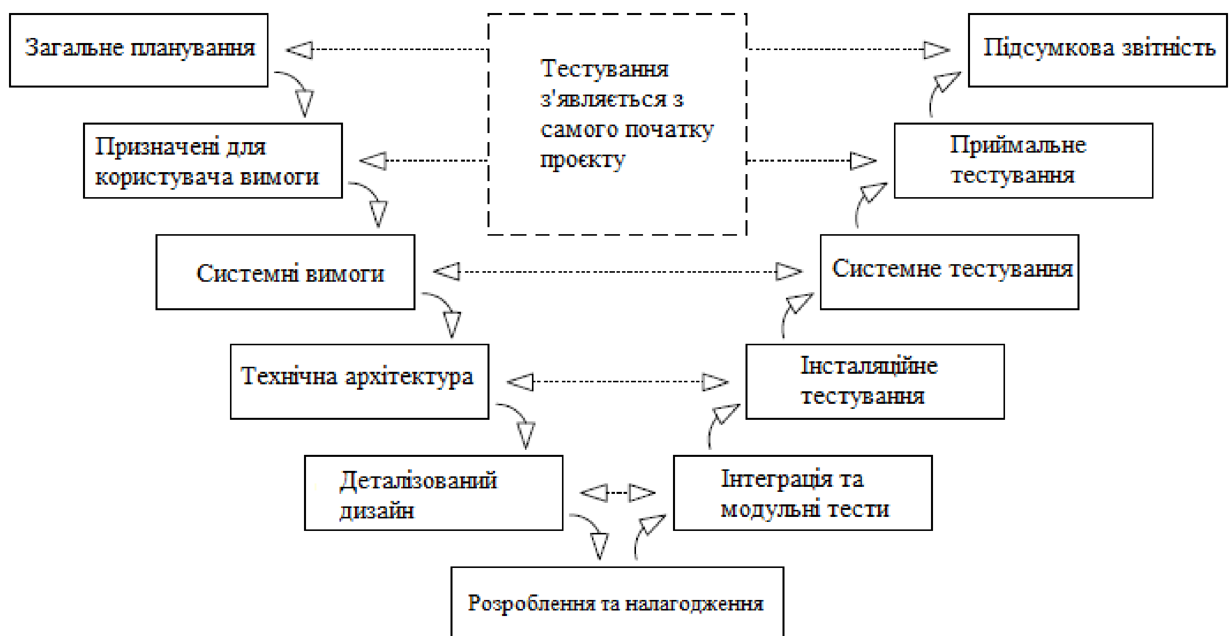

Рисунок 1.2 – V-подібна модель

Можна сказати, що при використанні V-подібної моделі на кожній стадії «на спуску» потрібно думати про те, що і як буде відбуватися на відповідній стадії «на підйомі». Тестування тут з’являється вже на самих ранніх стадіях розвитку проєкту, що дозволяє мінімізувати ризики, а також виявити і усунути безліч потенційних проблем до того, як вони стануть проблемами реальними [6].

1.3.3 Особливості методу оброблення ітераційної інкрементальної моделі

Ітераційна інкрементальна модель є фундаментальною основою сучасного підходу до розроблення програмного забезпечення. Як випливає з назви моделі, їй властива певна подвійність, а ISTQB-глосарій навіть не надає єдиного визначення, розбиваючи його на окремі частини:

– з точки зору життєвого циклу модель є ітераційною, тому що має на увазі багаторазове повторення одних і тих же стадій;

– з точки зору розвитку продукту (збільшення його корисних функцій) модель є інкрементальною.

Ключовою особливістю даної моделі є розбиття проєкту на відносно невеликі проміжки (ітерації), кожен з яких в загальному випадку може включати в себе всі класичні стадії, властиві водоспадній та V-подібній моделям (рис. 1.3).

Підсумком ітерації є приріст (інкремент) функціональності продукту, який виражається у демоверсії застосунку.

Довжина ітерацій може змінюватися в залежності від багатьох факторів, однак сам принцип багаторазового повторення дозволяє гарантувати, що і тестування, і демонстрація продукту кінцевому замовнику (з отриманням зворотного зв’язку) буде активно застосовуватися з самого початку і протягом усього часу розроблення проєкту [6].

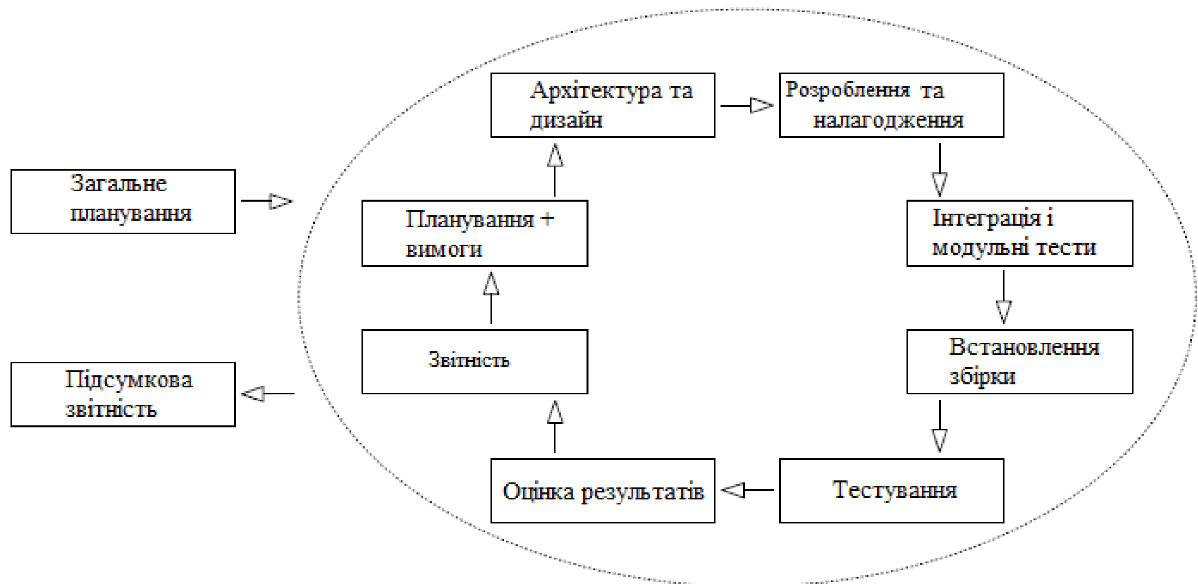

Рисунок 1.3 – Ітераційна інкрементальна модель

У багатьох випадках допускається розпаралелювання окремих стадій всередині ітерації і активна доробка з метою усунення недоліків, виявлених на будь-який з (попередніх) стадій.

Ітераційна інкрементальна модель дуже добре зарекомендувала себе на об’ємних і складних проєктах, які виконуються великими командами протягом тривалих термінів. Однак, до основних недоліків цієї моделі часто відносять великі витрати, що викликані громіздкістю моделі.

1.3.4 Особливості методу оброблення спіральної моделі

Спіральна модель являє собою окремий випадок ітераційної інкрементальної моделі, в якому особлива увага приділяється управлінню ризиками, яке особливо впливає на організацію процесу розроблення проєкту і контрольні точки [7].

Схематично суть спіральної моделі представлена на (рис. 1.4), на ньому явно виділено чотири ключові фази:

– опрацювання цілей, альтернатив і обмежень;

– аналіз ризиків і прототипування;

– розроблення (проміжної версії) продукту;

– планування наступного циклу.

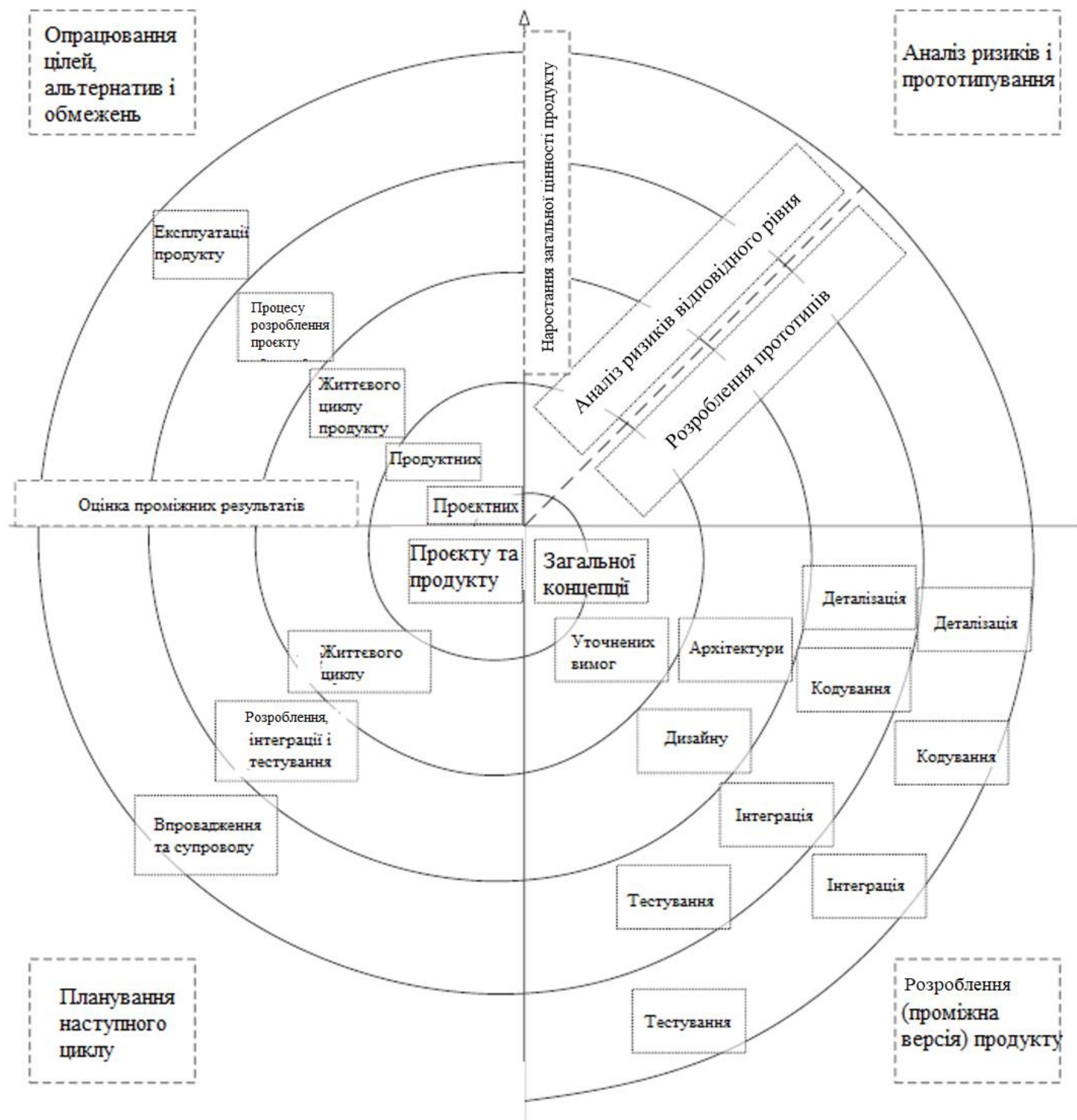

Рисунок 1.4 – Спіральна модель

З точки зору тестування і управління якістю підвищена увага до ризиків є відчутною перевагою при використанні спіральної моделі для розроблення концептуальних проєктів, в яких вимоги природним чином є складними і нестабільними (можуть багаторазово змінюватися по ходу виконання проєкту).

1.3.5 Особливості методу оброблення гнучкої моделі

Гнучка модель є сукупністю різних підходів до розроблення програмного забезпечення (рис. 1.5) і базується на маніфестах Agile:

– люди і взаємодія важливіше процесів та інструментів;
– працюючий продукт важливіше вичерпної документації;
– співпраця з замовником важливіше узгодження умов контракту;
– готовність до змін важливіше проходження попереднім планом.

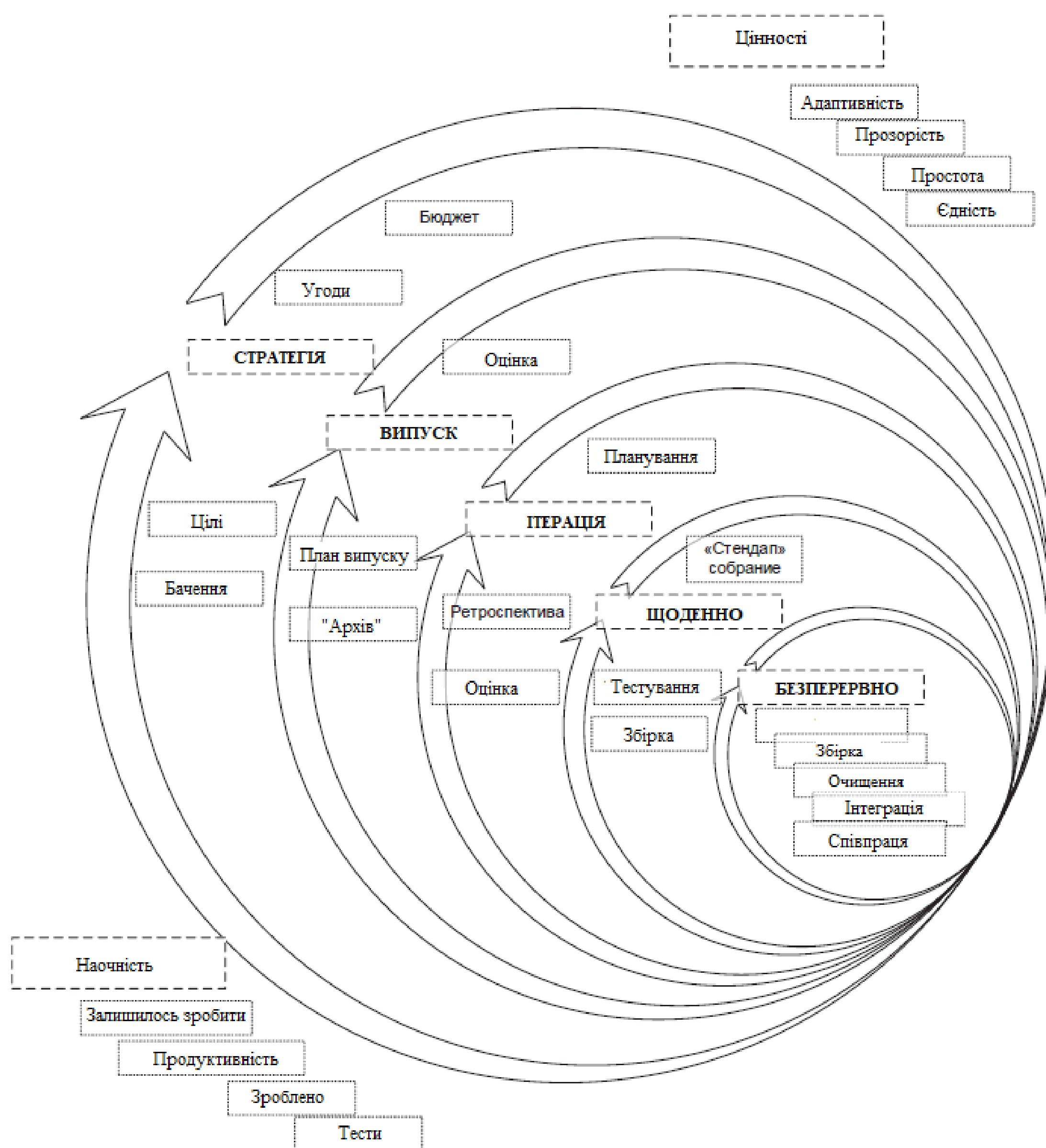

Рисунок 1.5 – Гнучка модель

Покладені в основу гнучкої моделі підходи є логічним розвитком і продовженням всього того, що було за десятиліття створено і випробувано у водоспадній, V-подібній, ітераційній інкрементальній, спіральній та інших моделях. Причому тут вперше був досягнутий відчутний результат в зниженні бюрократичної складової і максимальної адаптації процесу розроблення програмного забезпечення до миттєвих змін ринку і вимог замовника [7].

Головним недоліком гнучкої моделі вважається складність її застосування до великих проєктів (рис. 1.6), а також часте помилкове впровадження її підходів, викликане нерозумінням фундаментальних принципів моделі.

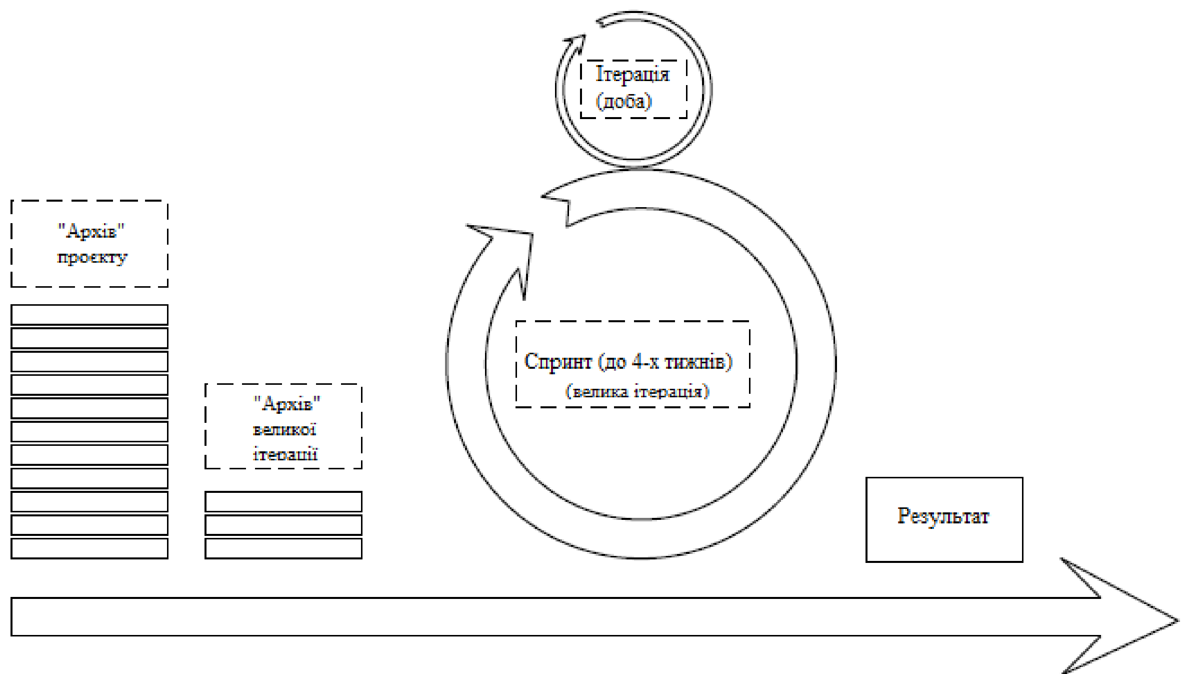

Рисунок 1.6 – Ітераційний підхід в рамках гнучкої моделі і Scrum

1.4 Аналіз літературних джерел щодо апробації результатів застосування методів тестування мобільних застосунків

Тестування мобільного застосунку є одним з найважливіших процесів у моделі його розроблення, тому що саме на цій стадії виявляються найбільш критичні помилки та перевіряється вся працездатність програмного забезпечення. За десятки років була створена велика кількість методів тестування, та написано багато робіт про роботу з ними, то ж розглянемо деякі з них:

У джерелі [1] розглядається поняття тестування, розкриваються технологічні та інфраструктурні принципи створення модулів тестування для сучасних програмних продуктів, їх застосування та оцінка корисності з точки зору життєвого циклу програми. У контексті даної роботи стають зрозумілими принципи створення модулів тестування, тому можна проконтролювати критерій цілісності процесу тестування.

Джерело [2] демонструє дослідження, у якому розглядається створення комп'ютерних засобів для автоматизації тестування мобільних застосунків. Результати роботи дозволяють запровадити автоматизацію тестування застосунків на базах iOS та Android; перевірити правильність роботи застосунків за короткий проміжок часу; провести перевірку логіки роботи програми і правильність роботи API-сервера. Даний матеріал допоможе зрозуміти принцип тестування мобільних застосунків.

У статті [3] описується важливість етапу тестування для розроблення мобільних застосунків. Виявлено основні види та етапи тестування мобільних застосунків, описані їх складності. Описано функції тестування мобільних застосунків, що дає змогу зрозуміти глобальний процес тестування на прикладі тестування застосунків. Наукова праця дозволяє зрозуміти етапи процесу тестування мобільних застосунків на доступних прикладах.

Стаття [4] описує процес планування тестування програмного продукту, показує приклад визначення цілей тестування, представлені кроки з підготовки тестових варіантів для мобільного застосунку FesCam, реалізованого для мобільних операційних систем Android і iOS. Стаття допоможе оглянути глобальний процес тестування на прикладі планування позитивних та негативних тестів.

Джерело [5] надає детальний опис особливостей двох основних підходів до тестування мобільних застосунків, а саме – «ручне» та «автоматизоване» тестування.

У статті [6] розглядаються способи тестування мобільних застосунків для різних операційних систем, а також можливості даного процесу. Пояснюється, чому процес тестування є надзвичайно важливим, визначено основні техніки тест-дизайну, які широко використовуються у негативних тестових сценаріях. Техніки «тест-дизайну», такі як «попарне тестування» або «граничні значення» дають розуміння коректного тестування застосунків, що є дуже важливим у процесі тестування програмного забезпечення.

Робота [7] виявляє основні моменти тестування мобільних застосунків: чим тестування застосунків для мобільних пристроїв відрізняється від тестування застосунків для операційних систем, на що варто звертати особливу увагу тестувальникам для досягнення своєї мети – створення популярного застосунку. Ця робота аналізує «підводні камені» під час тестуванні мобільних застосунків.

Наступна робота [8] описує тестування мобільних застосунків з точки зору компанії, що дозволяє тестувати застосунок користувачам. Розглянуто три основні методи тестування користувачами та розробниками, а саме: «Альфа-тестування», «Бета-тестування», «Відкритий бета-тест». У контексті цієї роботи таке тестування допоможе зрозуміти як впливає «тестова програма доступу Google Play» на тестування мобільного застосунку користувачами.

Праця [9] досліджує та надає результати тестування «у цілому». У ній розглядається тестування з точки зору контролю якості, описані фундаментальні правила та структури тестування програмного забезпечення, принципи тісної інтеграції процесу тестування з бізнес-логікою та бізнесом узагалі. Наведені класифікації типів дефектів та ідентифікації ризиків під час процесу контролю якості, проаналізовано такі поняття: контроль якості та забезпечення якості.

Остання досліджена робота [10] надає поетапний опис процесу автоматизації тестування. Наведено приклади застосування мов програмування, основ автоматизації, роботи з суміжними програмами для автоматизації тестування, а також особливості керування процесом виконання тестування. Зазначено можливості для підвищення технічного розуміння щодо написання автоматичних тестів для мобільного застостунку.

1.5 Постановка задачі дослідження

Актуальність даної роботи полягає в тому, що у сучасному світі процес тестування став невід’ємним від процесу розроблення програмного забезпечення. Розуміння процесів тестування допоможе побачити важливість цих процесів та зрозуміти роль тестування у життєвому циклі програмного забезпечення, а також дасть можливість впровадити підходи реалізації методів тестування у повсякденні процеси.

Об’єктом роботи є процес автоматизації тестування мобільного застосунку, заснованого на принципі взаємодії користувача з медіа-контентом.

Метою роботи є дослідження обраних методів тестування мобільного застосунку та виявлення найкращого щодо вирішення поставлених задач.

Враховуючи мету роботи, необхідно вирішити такі завдання:

– дослідити роль тестування у життєвому циклі розроблення мобільних застосунків;

– дослідити існуючі методи тестування та провести їх класифікацію;

– проаналізувати сучасний стан застосування методів тестування мобільних застосунків, здійснити вибір декількох методів тестування;

– змоделювати процес тестування за допомогою стандарту IDEF3;

– дослідити обрані методи тестування мобільних застосунків;

– проаналізувати непередбачувані ситуації під час тестування;

– обрати інструментальні засоби для реалізації проєкту автоматизації;

– розробити проєкт мобільної автоматизації обраного застосунку;

– протестувати мобільний застосунок обраними методами тестування та здійснити порівняльний аналіз;

– визначити перспективи подальшої роботи.

2 ДОСЛІДЖЕННЯ ВИБРАНИХ МЕТОДІВ ТЕСТУВАННЯ МОБІЛЬНИХ ЗАСТОСУНКІВ

2.1 Моделювання процесу тестування мобільних застосунків за допомогою стандарту IDEF3

Інформаційні технології здійснили еволюційний розвиток практично у всіх сферах діяльності людини. За рахунок автоматизації збільшується швидкість обробки інформації і, отже, підвищується продуктивність праці.

Процес моделювання будь-якої інформаційної системи дозволяє автоматизувати її управління, передбачає збільшення швидкості передачі інформації та підвищення контролю якості. Змодельована система має самонастроювальний характер, тобто наявний однозначний алгоритм дій, де кожен байт інформації спрямований у відповідний центр оброблення даних. Зміна навколишнього середовища спричинить ланцюжок змін у системі організації бізнесу.

Технічно, моделювання не менш трудомісткий процес, чим складання бізнес-плану. У ході роботи необхідно знову і знову повертатися до вже прописаних моментів, вносити зміни і будувати все заново, а потім тестувати та удосконалювати отриманий IT-продукт.

Вимоги є відправною точкою для визначення того, що проєктна команда буде моделювати, реалізовувати та тестувати. Якщо у вимогах присутня невизначеність [11–23], то і реалізовано буде не те, що очікували, тобто робота людей буде виконаною даремно. Візуалізований приклад цієї ситуації наведений на рисунку 2.1.

Мобільні застосунки особливо потребують глибокого моделювання процесів тестування, тому що один інструментарій може бути встановлений на різні пристрої з різними операційними системами, програмними оболонками, діагоналями, тощо.

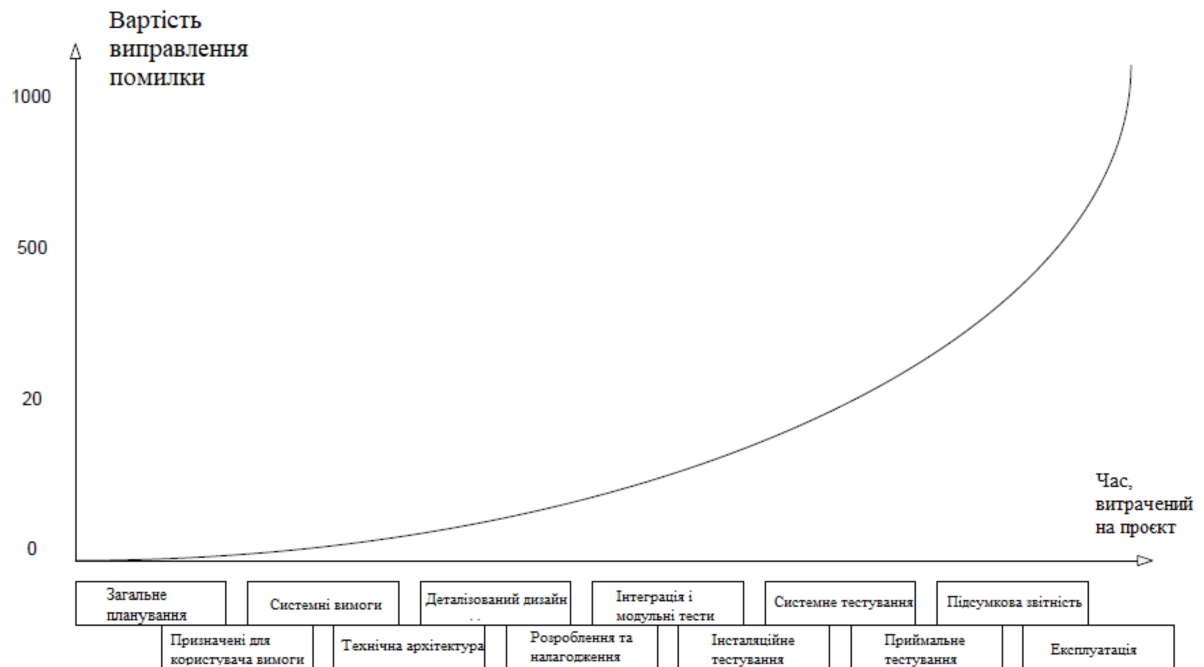

Рисунок 2.1 – Графік залежності вартості виправлення помилки від моменту її виявлення

Для моделювання процесів нижчого рівня доцільно використовувати методологію стандарту IDEF3.

IDEF3 – спосіб опису процесів з використанням структурованого методу, що дозволяє експерту в предметної області уявити стан речей як упорядковану послідовність подій з одночасним описом об’єктів, що мають безпосереднє відношення до процесу.

Дана методологія дозволяє показувати можливі розгалуження в процесі. Наприклад, коли результат однієї дії може ініціювати запуск декількох дій або навпаки, щоб почати якусь дію, необхідно завершити декілька попередніх дій.

Контекстна діаграма процесу тестування мобільного застосунку наведена на рисунку 2.2.

Діаграма декомпозиції першого рівня процесу тестування мобільного застосунку наведена на рисунку 2.3.

Рисунок 2.2 – Контекстна діаграма процесу
тестування мобільного застосунку

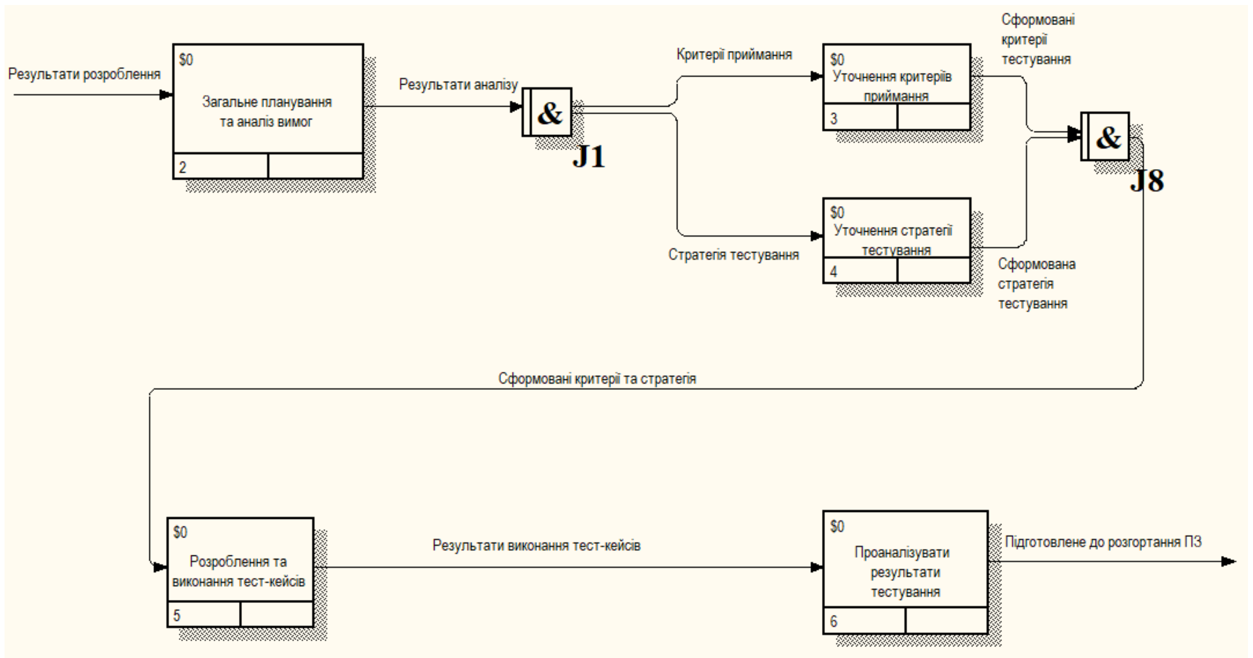

Рисунок 2.3 – Діаграма декомпозиції першого рівня процесу
тестування мобільного застосунку

На цьому етапі життєвого циклу ПЗ (життєвий цикл програмного забезпечення – SDLC) тестувальники отримують розроблений продукт і починають створювати тестовий план, керуючись загальними вимогами до проєкту та документацією розробленого ПЗ.

Опрацьовуються різні варіанти використання застосунку, розлядаються атрибути якості згідно з бізнес-вимогами та вимогами потенційних користувачів (рис. 2.4).

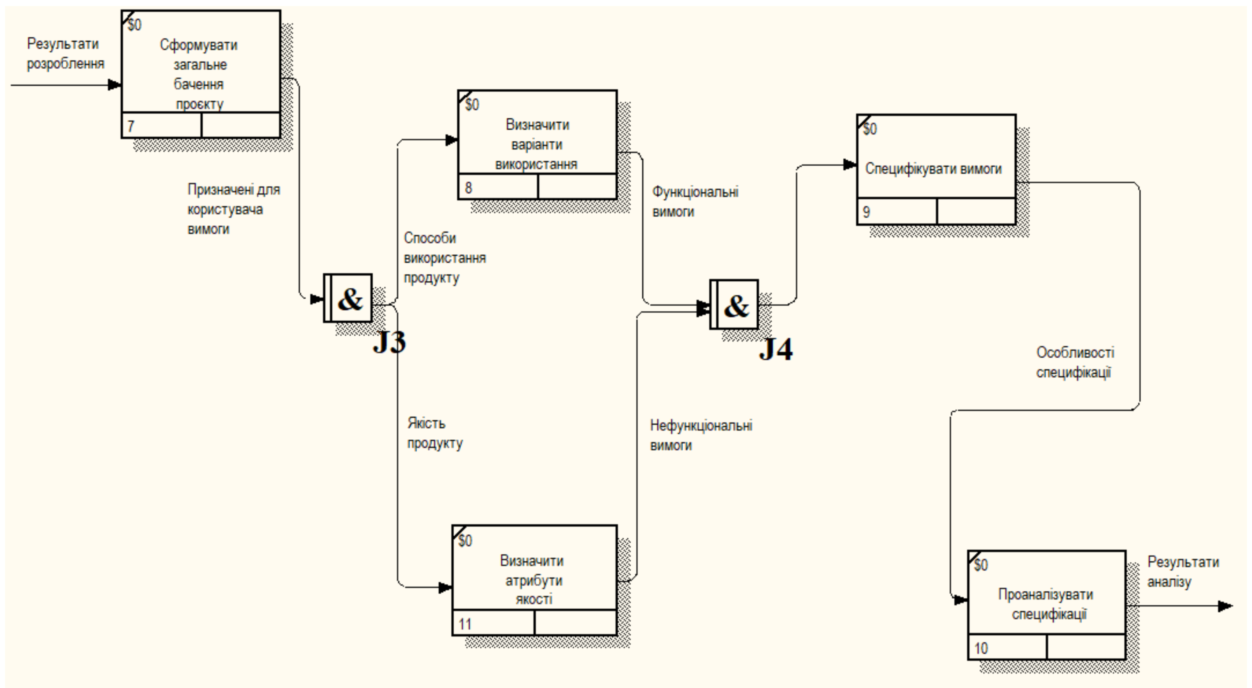

Рисунок 2.4 – Діаграма декомпозиції другого рівня процесу загального планування та аналізу вимог

Відповідно до вимог команда тестувальників уточнює критерії приймання результатів задля розуміння, коли продукт може бути допущений до використання користувачем (рис. 2.5).

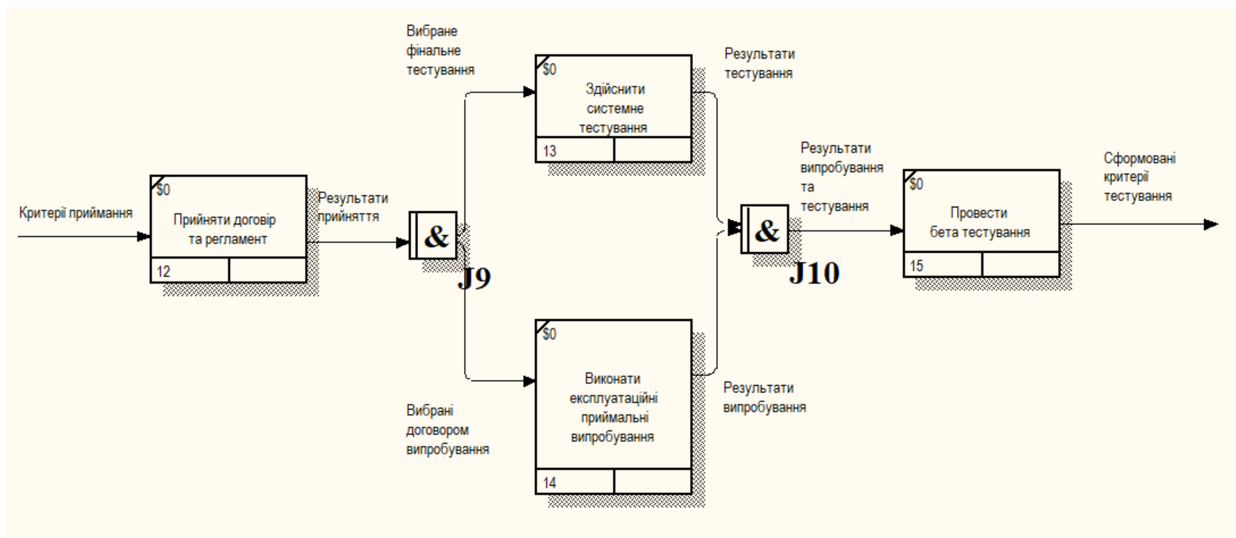

Рисунок 2.5 – Діаграма декомпозиції другого рівня процесу уточнення критеріїв приймання продукту

Паралельно з цим команда займається уточненням стратегії тестування, відштовхуючись від строків презентації застосунку користувачам, специфіки проєкту та загального досвіду кожного з членів команди (рис. 2.6).

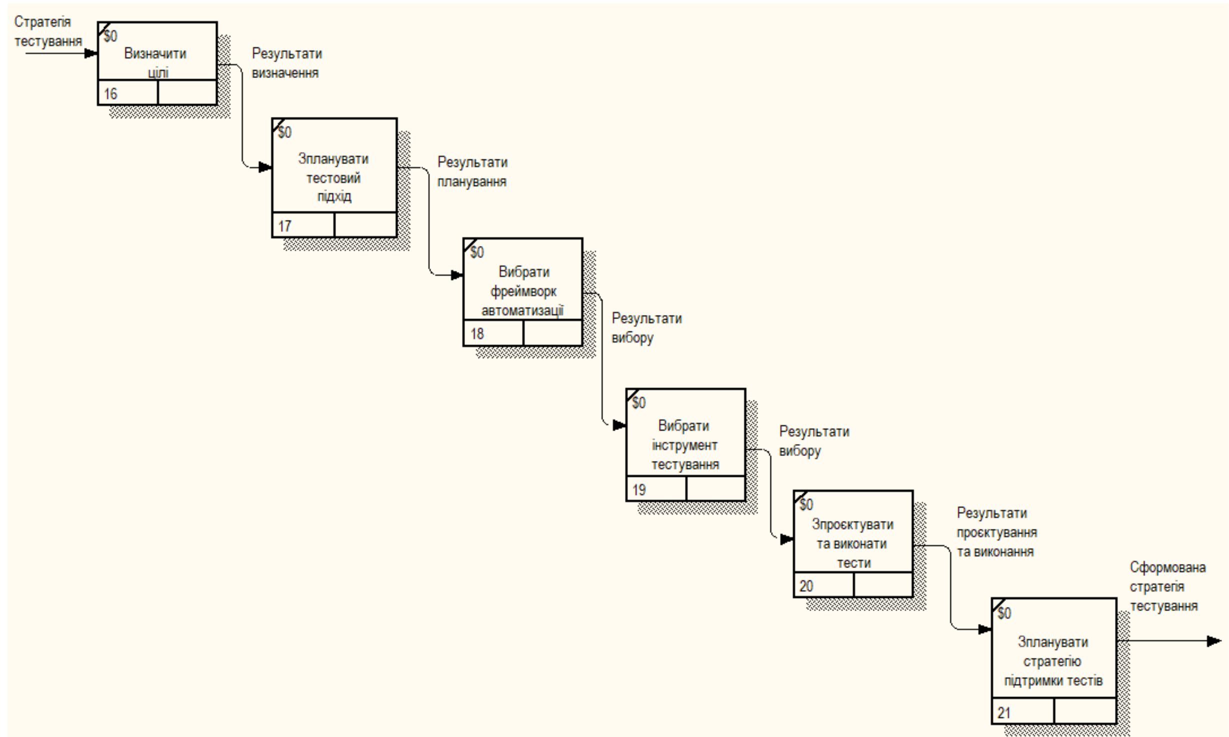

Рисунок 2.6 – Діаграма декомпозиції другого рівня процесу уточнення стратегії тестування продукту

Після результатів уточнення стратегії тестування та критеріїв приймання тестувальники починають розроблювати чек-листи та тест-кейси задля документування усіх типів перевірок. Команда тестувальників виконує тестування згідно з цими тест-кейсами, знаходить та документує помилки у розробленому ПЗ.

Команда ж «тестувальників автоматизаторів» після ручних перевірок пише код, який автоматизує ці перевірки, задля економії часу та збільшення надійності створених перевірок (рис. 2.7).

Після декількох циклів тестування команда починає приймальне тестування, за результатами якого замовник та команда вирішують чи можна випускати створений продукт на ринок (рис. 2.8).

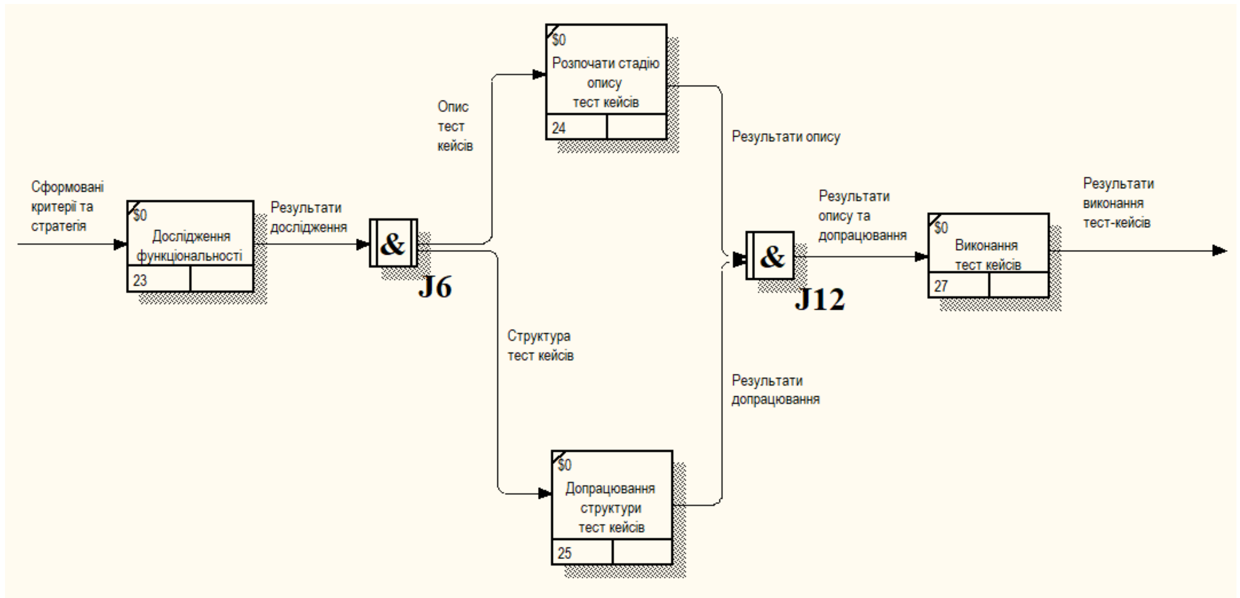

Рисунок 2.7 – Діаграма декомпозиції другого рівня процесу розроблення та виконання тест-кейсів

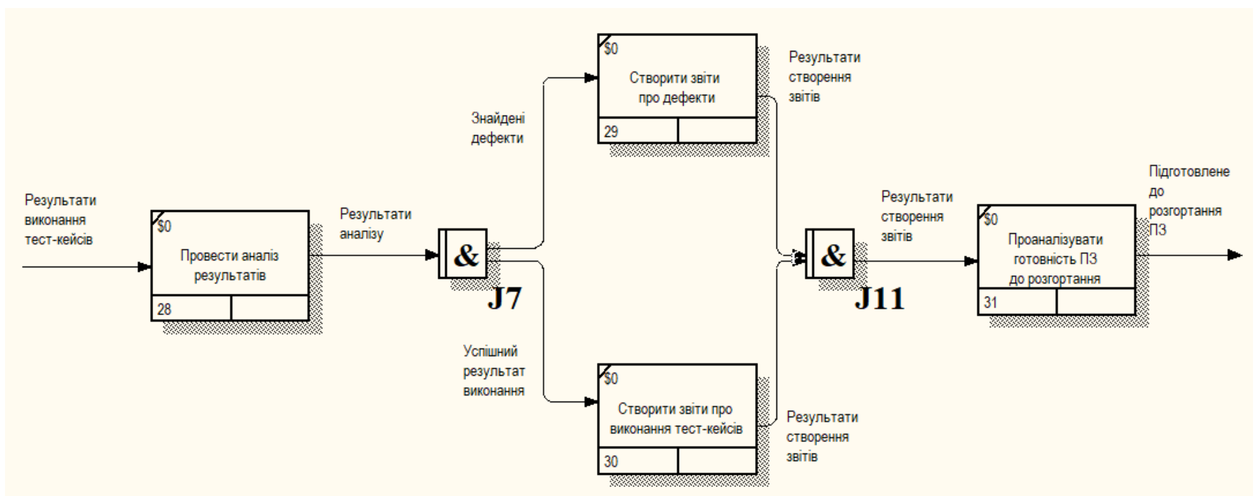

Рисунок 2.8 – Діаграма декомпозиції другого рівня процесу створення звітів

Створені тест-плани, тест-кейси, чек-листи та задокументовані помилки ПЗ зберігаються і використовуються при наступних циклах тестування, якщо вони будуть, а також на етапі підтримки створеного продукту (останній пункт SDLC).

2.2 Дослідження методу тестування «Функціональне тестування»

Усі види тестування програмного забезпечення, у залежності від переслідуваних цілей, можна умовно розділити на наступні групи:

– функціональні;

– нефункціональні;

– пов’язані зі змінами.

Функціональні тести базуються на функціях і особливостях, а також взаємодії з іншими системами, і можуть бути представлені на всіх рівнях тестування: компонентному або модульному (Component / Unit testing), інтеграційному (Integration testing), системному (System testing) і приймальному (Acceptance testing) [24].

Як правило, ці функції описуються у вимогах, функціональних специфікаціях або у вигляді випадків використання системи (Use Cases).

Функціональні види тестування розглядають зовнішню поведінку системи і ґрунтується на аналізі специфікацій функціональності компонента або системи в цілому.

Тестування функціональності може проводитися у двох аспектах:

– вимоги (рис. 2.9);

– бізнес-процеси (рис. 2.10).

Тестування у перспективі «вимоги» використовує специфікацію функціональних вимог до системи як основу для дизайну тестових випадків (Test Cases).

У цьому випадку необхідно зробити список того, що буде тестуватися, а що ні, пріоритезувати вимоги на основі ризиків (якщо це не зроблено в документі з вимогами), на основі цього пріоритезувати тестові сценарії (Test Cases). Це дозволить сфокусуватися і не упустити при тестуванні найбільш важливий функціонал [25].

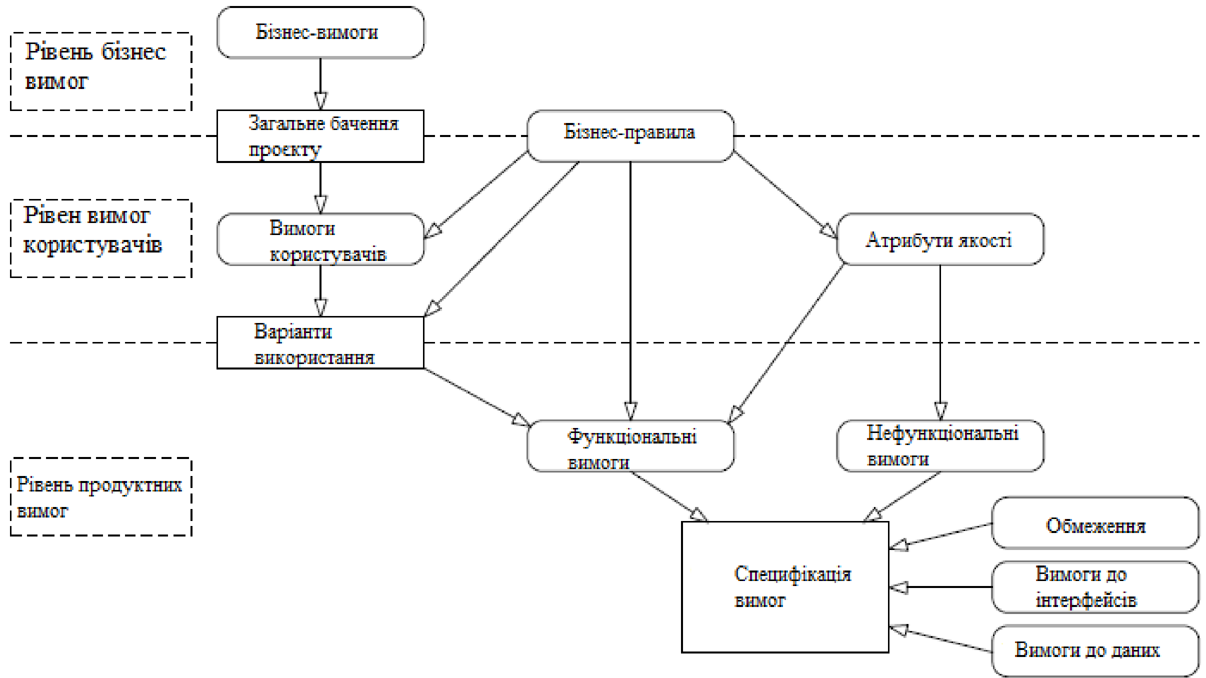

Рисунок 2.9 – Блок-схема специфікації вимог

до функціонального тестування згідно з вимогами

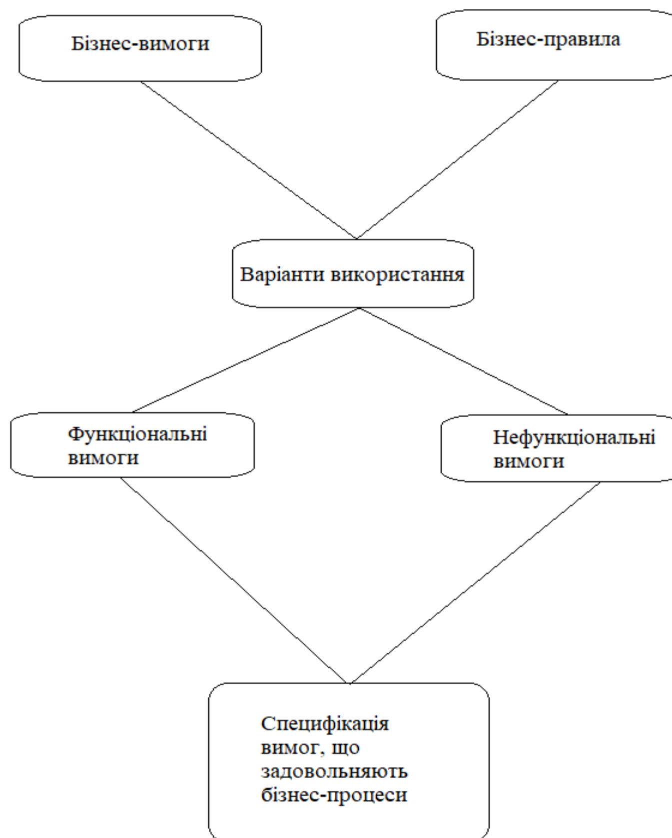

Рисунок 2.10 – Блок-схема специфікації вимог

до функціонального тестування згідно з бізнес-процесами

Тестування у перспективі «бізнес-процеси» використовує знання цих бізнес-процесів, які описують сценарії щоденного використання системи, тестові сценарії (Test Scripts), як правило, ґрунтуються на випадках використання системи (Use Cases).

До переваг функціонального тестування можна віднести «імітування фактичного використання системи», до недоліків – можливість упущення логічних помилок в програмному забезпеченні та ймовірність надмірного тестування.

Розглянемо програмний код прикладу функціонального тестування: необхідно перевірити, що користувач може створити профіль, дати йому назву та побачити створений профіль, згідно з вимогам до продукту [26].

Лістинг 2.1. Скрипт функціонального тестування за допомогою програмного коду:

```
LoginActions.login(self, creds)
HomeScreenActions(self).tap_settings_menu_button()
settings_menu = SettingsMenu(self)
settings_menu.tap_change_profile_button()

# Validate profiles page
self.validate_manage_profiles_page()
self.tap_add_profile_button()
edit_profile = EditProfilePage(self)

edit_profile.enter_nickname(self.NEW_PROFILE)
edit_profile.tap_save_changes_button()

# Validate that profile is added
self.wait_for_load()
self.validate_profiles_after_adding(self, profiles_before, self. NEW_PROFILE)
```

2.3 Дослідження методу тестування «Димове тестування»

Поняття «димове» тестування пішло від результатів інженерних експериментів: «При введенні в експлуатацію нового обладнання («заліза») вважалося, що тестування пройшло вдало, якщо з установки не пішов дим».

В області програмного забезпечення димове тестування розглядається як короткий цикл тестів, що виконуються для підтвердження того, що після збірки коду (нового або відредагованого) застосунок стартує і виконує основні функції [27].

Висновок про працездатність основних функцій робиться на підставі результатів поверхневого тестування найбільш важливих модулів програми на предмет можливості виконання необхідних завдань і наявності критичних і блокуючих дефектів.

У разі відсутності таких дефектів димове тестування оголошується пройденим, і застосунок передається для проведення повного циклу тестування, в іншому випадку, димове тестування оголошується проваленим, і застосунок йде на доопрацювання (рис. 2.11).

Аналогами димового тестування є тестування збірки (Build Verification Testing) і приймальне тестування (Acceptance Testing), що виконуються на функціональному рівні командою тестувальників, за результатами яких робиться висновок про те, приймається чи ні дана версія програмного забезпечення у тестування, експлуатацію або на передачу замовнику [26].

Розглянемо програмний код (лістинг 2.2) прикладу димового тестування: необхідно перевірити функціональність логіну до застосунку.

Це одна з найголовніших функцій, її помилки можуть призвести до великих втрат на виправлення дефектів, якщо про це стане відомо занадто пізно.

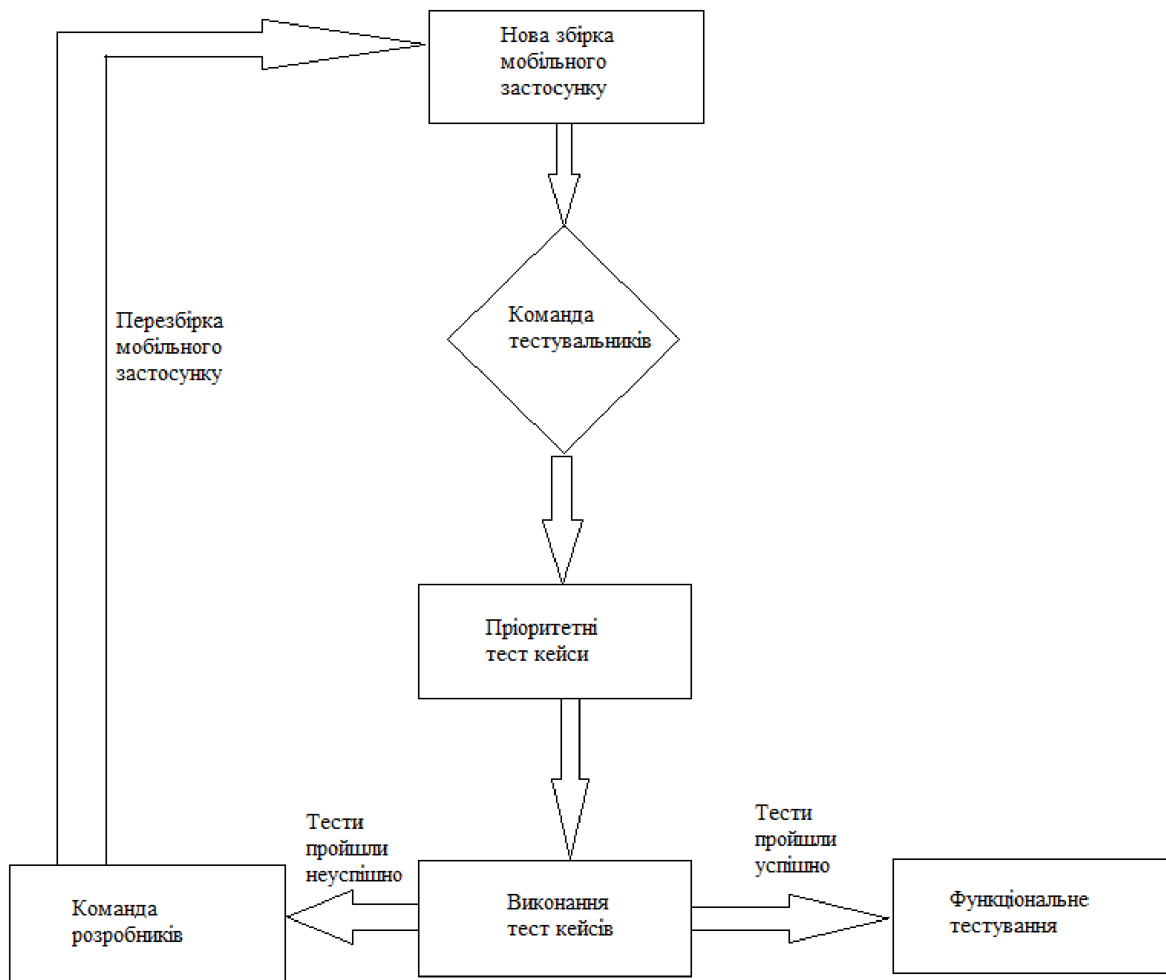

Рисунок 2.11 – Блок-схема структури «димового» тестування

Лістинг 2.2. Скрипт димового тестування за допомогою програмного коду:

```
home_page = HomePage(self)
home_page.tap_login_button()
subscription_page = SubscriptionPage(self)
subscription_page.wait_for_load()
subscription_page.type_email(self.creds.login)
subscription_page.type_password(self.creds.password)
subscription_page.tap_login_button()
assertFalse(home_page.is_login_button_displayed())
```

Сучасні методології розроблення практикують підхід безперервної інтеграції (Continuous Integration), який є частиною збірки програмного продукту.

Збірка не завжди буває належної якості, вони можуть містити дефекти у функціональності, критичній для бізнесу, тому перевірка повинна здійснюватися безпосередньо після складання і перед передачею на тестування.

Це дозволяє скоротити витрати часу на тестування збірки, що містить блокуючі дефекти.

Ключові переваги димового тестування:

– виявлення критичних помилок в перші кілька годин (хвилин) після установки;

– зниження ризиків виведення неякісного продукту;

– мінімізація ризиків при інтеграції систем;

– скорочення витрат на виправлення дефектів;

– прискорення перевірки за рахунок автоматизації.

2.4 Дослідження методу тестування «Тестування білого ящика»

Тестування білого ящика – це тестування внутрішньої структури, дизайну і кодування програмного рішення. У цьому типі тестування код видно тестувальнику. Основна увага приділяється перевірці потоку вхідних і вихідних даних через застосунок, поліпшенню дизайну, зручності використання, посиленню безпеки. Тестування білого ящика також відомо як тестування чистого ящика (Clear Box), тестування відкритого ящика (Open Box), структурне тестування, тестування прозорого боксу, тестування на основі коду і тестування скляного ящика (Glass Box) [27].

Тестування білого ящика – це одна з двох частин підходу тестування ящиків (Box Testing) до тестування програмного забезпечення.

Тестування «чорного ящику», включає тестування з точки зору зовнішнього або кінцевого користувача. Тестування «білого ящику» засноване на внутрішній роботі програми та обертається навколо внутрішнього тестування.

Термін «білий ящик» був використаний через концепції прозорої коробки, що символізує здатність бачити крізь зовнішню оболонку програмного забезпечення (або коробку) його внутрішню роботу.

«Чорний ящик» символізує неможливість побачити внутрішню роботу програмного забезпечення, тому може бути протестований тільки досвід кінцевого користувача.

Тестування білого ящика включає тестування програмного коду для наступного:

– внутрішні дефекти в безпеці;

– зламані або погано структуровані шляхи в процесах кодування;

– функціональність умовних циклів;

– тестування кожного оператора, об’єкта і функції на індивідуальній основі.

Тестування може проводитися на системному, інтеграційному та модульному рівнях розроблення програмного забезпечення. Однією з основних цілей тестування «білого ящика» є перевірка робочого процесу для застосунку. Тестування «білого ящика» включає в себе тестування ряду зумовлених вхідних даних по відношенню до очікуваних або бажаних вихідних даних. Якщо конкретне введення даних не призводить до очікуваного вихідного сигналу, то це вказує на дефект системи [28].

Перше, що часто робить тестувальник, він вивчає і вникає у вихідний код програми. Оскільки тестування «білого ящика» включає в себе тестування внутрішньої роботи програми, тестувальник повинен бути дуже добре обізнаним у мовах програмування, що використовуються у тестових застосунках.

Крім того, тестувальник повинен бути добре обізнаним у методах безпечного кодування. Безпека часто є однією з основних задач тестування програмного забезпечення.

Тестувальник повинен вміти виявляти проблеми з безпекою, запобігати атакам хакерів і наївних користувачів, які можуть вводити шкідливий код у застосунок як свідомо, так і несвідомо.

Другий крок тестування «білого ящика» включає в себе тестування вихідного коду програми на предмет належного функціонування та структури. Одним із способів є написання більшої кількості коду для перевірки вихідного коду програми. Тестувальник розробить невеликі тести для кожного процесу або серії процесів у застосунку [29].

Лістинг 2.3. Скрипт тестування «білого ящика» за допомогою програмного коду:

```
def printme (a, b):
    result = a + b;
    if result > 0:
        print ("Positive: ", str(result))
    else:
        print ("Negative: ", str(result))
```

Метою тестування «білого ящика» є перевірка всіх гілок рішень, циклів та операторів в коді (рис. 2.12).

Щоб виконати тестування у наведеному вище коді, тестовими випадками «білого ящика» будуть:

- a = 1, b = 1;
- a = -1, b = -3.

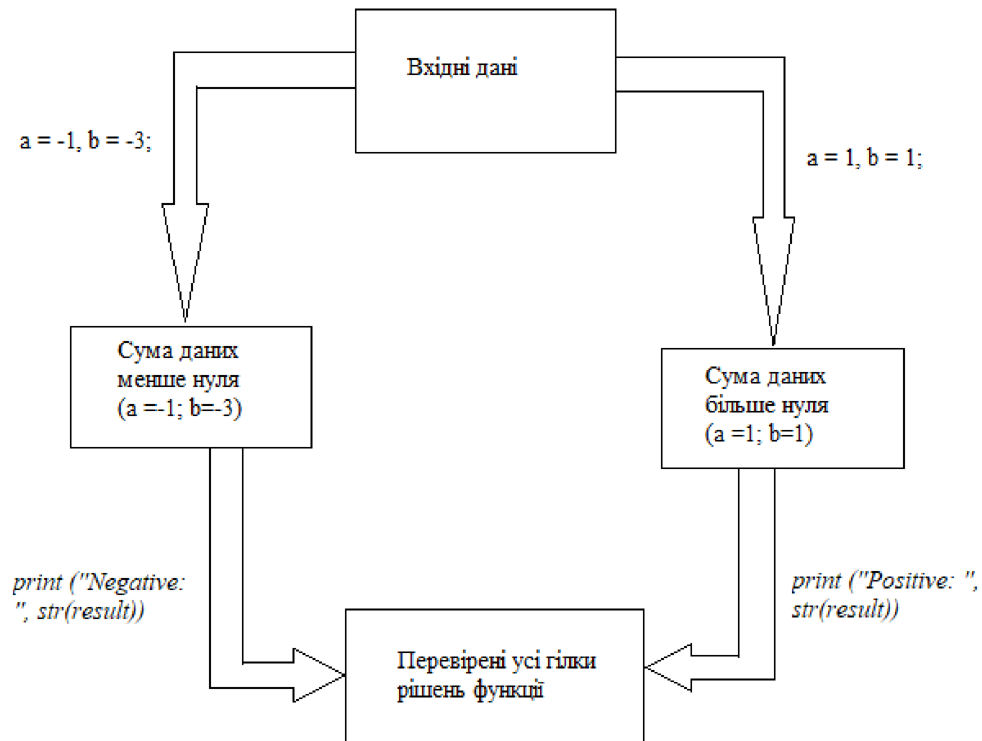

Рисунок 2.12 – Блок-схема методу тестування «білий ящик»

2.5 Аналіз можливих непередбачуваних ситуацій під час виконання досліджуваних методів тестування

Тестування грає життєву важливу роль в розробленні якісного програмного забезпечення. Тим не менше, у багатьох компаніях, що займаються розробленням ПЗ, процеси тестування недостатньо організовані, тому виконавці змушені йти важким шляхом, намагаючись домогтися бажаних результатів [30].

Тестування програмного забезпечення – процес дослідження, випробування програмного продукту, що має дві різні цілі:

– продемонструвати розробникам і замовникам, що програма відповідає вимогам;

– виявити ситуації, в яких поведінка програми є неправильною, небажаною або не відповідає специфікації.

На основі уявлення про способи використання продукту створюються випадки використання системи. По конкретному випадку використання можна визначити один або більше сценаріїв.

На перевірку кожного сценарію пишуться тест-кейси, які повинні бути протестовані. Як правило, функціональні тести пишуться тестувальниками.

До того ж, функціональними тестами набагато простіше покривати готовий продукт, ніж модульними, так як набагато простіше зрозуміти, що конкретно повинна і не повинна робити певна частина призначеного для користувача інтерфейсу, аніж визначити, що повинна робити функція [31].

Функціональне тестування розглядає заздалегідь вказану поведінку і ґрунтується на аналізі специфікацій функціональності компонента або системи в цілому. Функціональні тести ґрунтуються на функціях, виконуваних системою, і можуть проводитися на всіх рівнях тестування (компонентному, інтеграційному, системному, приймальному).

Як правило, ці функції описуються в вимогах, функціональних специфікаціях або у вигляді випадків використання системи (use cases).

Перевагою функціонального тестування є те, що воно імітує фактичне використання системи.

Недоліки функціонального тестування:

– можливість упущення логічних помилок в програмному забезпеченні;

– імовірність надлишкового тестування.

Непередбачувані ситуації під час тестування можна розділити на дві групи: бажані та небажані. Найчастішими бажаними непередбачуваними ситуаціями під час функціонального тестування є відхилення фактичного результату тесту від очікуваного. Наприклад, якщо знаємо, що, натиснувши на кнопку сайт має перенаправити користувача на конкретну сторінку, але сайт цього не робить, або перенаправляє на неправильну сторінку – це є «бажаною» непередбачуваною поведінкою, це помилка, яка знайшлась під час процесу тестування, а не під час користування користувачем.

Прикладом небажаних непередбачуваних ситуацій під час функціонального тестування можуть бути несправності технічного обладнання, на котрому працює застосунок. Наприклад, якщо це мобільний застосунок для сервісу перегляду медіаконтенту, то небажаною ситуацією цілком може бути несправність у мікросервісі, котрий надає цьому застосунку інформацію щодо фільмів та серіалів. Результатами такої непередбачуваної ситуації може бути працюючий застосунок, але без відображеної інформації важливої для користувача.

«Димове» тестування – це процес, в якому збірка програмного забезпечення розгортається в середовищі тестування і перевіряється для забезпечення стабільності застосунку. Він також називається «Тестування перевірки збірки» або «Перевірка достовірності». Тобто, перевіряється, чи працюють важливі функції, і чи в тестованій збірці немає дефектів.

Швидкий регресійний тест показує, що продукт готовий до тестування. Він допомагає визначити, чи є збірка дефектною, показує, що подальше тестування є марною тратою часу і ресурсів.

Тестування «диму» грає важливу роль в розробленні програмного забезпечення, оскільки воно забезпечує правильність роботи системи на початкових етапах. У результаті «димові» тесті призводять систему в хороший стан. Як тільки закінчується тестування «диму», тоді починається функціональне тестування [32].

Основні переваги «димового» тестування:

– просте тестування;
– дефекти виявляє на ранніх стадіях;
– покращує якість системи;
– знижує ризик;
– економить зусилля і час тесту;
– легко виявляє критичні помилки і виправляє помилки;
– швидко працює;
– мінімізує інтеграційні ризики.

Основним прикладом бажаних непередбачуваних ситуації є будь-який знайдений дефект нової збірки, котрий сигналізує, що подальше функціональне тестування немає сенсу проводити, поки знайдені дефекти не будуть виправлені.

Наприклад, ситуація, коли після нової збірки у мобільному застосунку перегляду медіаконтенту перестала відображатися кнопка оформлення підписки на цей сервіс. І завдяки тому, що цей дефект був знайдений під час «димових» тестів, змогли швидко задокументувати помилку і відправити її на доопрацювання команді розробників. Якби не було тесту на відображення кнопки серед «димових» тестів, то збірка перейшла б до функціонального тестування, що витратило б багато часу та ресурсів на її знаходження, і знадобилося б набагато більше часу, ніж під час «димового» тесту аби задокументувати помилки та почати тести спочатку.

Прикладом небажаних непередбачуваних ситуацій під час «димового» тестування може бути недостатня кількість актуальних до такого типу тестування тест-кейсів, через що велика частина помилок може бути знайдена під час функціональних тестів.

Наприклад, якщо під час «димового» тестування не перевіряється якийсь розділ мобільного застосунку, то велику частину функціональних помилок цього розділу знайде користувач, що зменшить якість мобільного застосунку на досить великому конкурентному ринку мобільних застосунків.

Ручне та автоматизоване тестування грають ключову роль в процесі забезпечення якості ПЗ. Перш за все, перевірка коду дозволяє упевнитися в якості продукту, що розробляється, це підвищує репутацію застосунку за рахунок позитивних відгуків користувачів. Найчастіше на об’ємних і складних проєктах тестувальники зустрічаються із застосуванням як автоматизованого, так і ручного тестування одночасно.

Автоматизація тестування застосунків заснована на написанні коду. Методика автоматизованого тестування дозволяє створити очікуваний сценарій, а потім порівняти його з реальним і вказати розбіжності.

Автоматизація найбільш застосована у складних застосунках з великою функціональною частиною.

Особливою популярністю користується автоматизація тестування важкодоступних місць застосунку, валідаційних форм, базових операцій, часто використовуваної функціональності [33].

Переваги автоматизованого тестування:

– коли використовується автоматизоване тестування, стає можливим моделювання великого навантаження, яке наближене до реального використання застосунку;

– ручне тестування – це довгий і ресурсномісткий процес, у той час, як код для сценарію пишеться один раз;

– код автоматизованих тестів може бути використаний неодноразово, особливо, при впровадженні нової функціональності.

Недоліки автоматизованого тестування:

– автоматизоване тестування не здатне надати зворотний зв’язок щодо якості продукту – воно лише виконує запрограмовані сценарії;

– іноді в застосунку залишаються помилки, які можуть бути не покриті автоматизованими тестами;

– відсутність можливості тестування дизайну та ергономіки;

– автоматизовані тести можуть не працювати по причинах, наприклад, при великій завантаженості тестової машини або при проблемах з мережею;

– для невеликих проєктів інструменти автоматизованого тестування можуть виявитися досить витратними, тому раціонально їх використовувати для довгострокових проєктів.

Прикладів бажаних непередбачуваних ситуацій автоматизованого тестування майже немає, тому що, зазвичай, автоматизований процес має завжди працювати правильно та коректно, тому що, зазвичай, більшість непередбачуваних ситуацій виникає через неправильне написання коду або через нестабільність самих автоматизованих тестів, що робить ці непередбачувані ситуації небажаними.

Прикладів небажаних непередбачуваних ситуацій автоматизованого тестування багато, але основні з них це «негнучкість» написаного коду.

Оскільки тестова машина строго виконує код для тестування, виникають ситуації, коли написаний код не може врахувати усі гілки поведінки мобільного застосунку. У такій ситуації автоматизований тест буде «провалено», і тестувальнику необхідно буде розбиратися чому саме тест «провалився», тому що він знайшов помилку, чи тому, що сам код був написаний недостатньо «гнучким».

Наприклад, потрібно написати автоматизований тест, що складається з шести кроків, але з кожним наступним кроком варіанти поведінки застосунку збільшуються майже в геометричній прогресії. На останньому кроці створений тест покриває 35 варіантів поведінки застосунку, необхідно покращити код таким чином, щоб, якщо застосунок на одному з варіантів поведінки почав йти по іншому варіанту, то тест це зрозумів, і продовжив працювати згідно з поведінкою застосунку. Якщо неправильно написати код для цього тесту, то «провалений» тест буде небажаною непередбачуваною ситуацією.

3 ЗАСТОСУВАННЯ МЕТОДІВ ТЕСТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ ЩОДО ВИБРАНОЇ ПРЕДМЕТНОЇ ОБЛАСТІ

3.1 Вибір інструментальних засобів для реалізації вибраних методів

Задля імплементації тест-кейсів необхідно спочатку обрати та налаштувати інтегроване середовище розроблення IDE.

Інтегроване середовище розроблення – комплексне програмне рішення для розроблення програмного забезпечення. Зазвичай, складається з редактора початкового коду, інструментів для автоматизації складання та відлагодження програм. Більшість сучасних середовищ розроблення мають можливість автодоповнення коду.

Інтегровані середовища розроблення створені для того, щоб максимізувати продуктивність програміста, надавши йому пов'язані інструменти розроблення зі схожими інтерфейсами як одну програму, в якій відбуватиметься весь процес розроблення, яка надає необхідні функції для модифікації, компілювання, розгортання та налагодження програмного забезпечення [34].

Одним із завдань IDE є зменшення часу, необхідного на конфігурацію різноманітних інструментів розроблення, натомість, пропонуючи той самий набір, як єдине ціле. Це може збільшити продуктивність розробника, у випадку, коли навчання, як працює інтегроване середовище розроблення швидше, ніж освоєння всіх інструментів зокрема.

Крім того, більша інтеграція між вбудованими інструментами потенційно може сприяти додатковому збільшенню продуктивності. Наприклад, синтаксичний аналіз коду може відбуватися безпосередньо під час його редагування, тим самим виявляючи помилки ще до трансляції коду.

Опираючись на професійну мову програмування, у цій кваліфікаційній роботі було використано IDE під назвою «PyCharm».

Ця IDE спрямована на роботу з мовою програмування Python. PyCharm має зручний редактор коду з усіма корисними функціями, такими як:

– підсвічування синтаксису;

– автоматичне форматування;

– автоматичне доповнення і відступи.

PyCharm дозволяє перевіряти версії інтерпретатора мови на сумісність, а також використовувати шаблони коду.

Python – високорівнева мова програмування загального призначення з динамічною строгою типізацією і автоматичним управлінням пам'яттю, орієнтована на підвищення продуктивності розробника, читання коду і його якості, а також на забезпечення переносності написаних на ній програм.

Мова є повністю об'єктно-орієнтованою – все є об'єктами, особливістю є виділення блоків коду пробільними відступами [35-41].

Синтаксис ядра мови мінімалістичний, за рахунок чого на практиці рідко виникає необхідність звертатися до документації.

Недоліками мови є часто більш низька швидкість роботи і більш високе споживання пам'яті написаних на ній програм у порівнянні з аналогічним кодом, написаним на компільованих мовах, таких як C або C^{++}.

Для реалізації тест-кейсів були обрані фреймворки «Unittest» та «Pytest».

У Python є вбудований модуль unittest, який підтримує автоматизацію тестів, використання загального коду для налаштування і завершення тестів, об'єднання тестів у групи, а також дозволяє відокремлювати тести від фреймворку для виведення інформації.

Для автоматизації тестів, unittest підтримує деякі важливі концепції:

– випробувальний стенд (test fixture) – виконується підготовка, яка необхідна для виконання тестів і всі необхідні дії для очищення після виконання тестів. Це може включати, наприклад, створення тимчасових баз даних або запуск серверного процесу;

– тестовий випадок (test case) – мінімальний блок тестування. Він перевіряє відповіді для різних наборів даних. Модуль unittest надає базовий

клас TestCase, який можна використовувати для створення нових тестових випадків;

– набір тестів (test suite) – кілька тестових випадків, наборів тестів або і того, і іншого. Він використовується для об'єднання тестів, які повинні бути виконані разом;

– виконавець тестів (test runner) – компонент, який управляє виконанням тестів і надає користувачеві результат. Виконавець може використовувати графічний або текстовий інтерфейс або повертати спеціальне значення, яке повідомляє про результати виконання тестів.

Pytest – це засноване на Python середовище тестування, яке використовується для написання і виконання тестових кодів. У даний час служби REST pytest в основному використовуються для тестування API, хоча можна використовувати pytest для написання простих і складних тестів, тобто можливо писати коди для тестування API, бази даних, призначеного для користувача інтерфейсу і т.д. [42-44].

Переваги pytest полягають у такому:

– pytest може виконувати кілька тестів паралельно, що скорочує час виконання набору тестів;

– у pytest є власний спосіб автоматичного визначення тестового файлу і тестових функцій, якщо це не вказано явно;

– pytest є безкоштовним і відкритим вихідним кодом;

– завдяки простому синтаксису, pytest дуже простий для запуску.

Задля запуску тестів на мобільному пристрої потрібно завантажити та налаштувати застосунок «Appium».

Appium – це кросплатформовий інструмент з відкритим вихідним кодом, який допомагає автоматизувати тестування мобільних застосунків для Android та iOS. Він буде інтерпретувати код програми у зрозумілі для мобільного пристрою запити.

Задля створення діаграм процесу тестування з детальною логікою взаємодії наявних компонентів обрано стандарт IDEF3, доцільно використати

програмний застосунок ERwin Data Modeler, згідно якого буде проводитись тестування.

Erwin Data Modeler – це комп’ютерна програма для проєктування і документування баз даних. Моделі даних допомагають візуалізувати структуру даних, забезпечуючи ефективний процес організації та управління даними.

3.2 Етапи програмної реалізації вибраних методів тестування мобільних застосунків

Задля реалізації обраних методів тестування мобільного застосунку потрібно виконати декілька етапів.

Етап 1. Визначення вихідних даних для практичної реалізації.

Вихідними даними для реалізації тестування мобільного застосунку обраними методами є:

– створена модель процесу тестування у вигляді діаграм IDEF3;

– завантажене та налаштоване інтегроване середовище розроблення PyCharm;

– завантажений та налаштований інтерпретатор мови Python 3.7;

– створене віртуальне середовище з обраним інтерпретатором мови у середовищі розроблення PyCharm;

– завантажений фреймворк pytest;

– завантажений та налаштований застосунок Appium;

– мобільний застосунок YouTube для тестування.

Етап 2. Визначення особливостей предметної області для застосування методів.

У якості мобільного застосунку для тестування обрано мобільний застосунок YouTube, який відноситься до категорії застосунків, що працюють з потоковим медіа-контентом.

Задачами тестування мобільного застосунку YouTube є:

– перевірка правильної поведінки елементів на екрані «Home»;

– перевірка правильної поведінки елементів на екрані «Search»;

– перевірка правильної поведінки елементів на екрані «Shorts»;

– перевірка правильної поведінки елементів на екрані «Subscriptions»;

– перевірка правильної поведінки елементів на екрані «Settings».

Відповідно до обраних методів тестування, основною метою тестування буде валідація основних функцій мобільного застосунку на кожному із розглянутих екранів, без яких його подальше використання стане неможливим.

Етап 3. Класифікація задач предметної області.

Опираючись на обрані задачі стає можливим класифікувати їх:

– тестування відображення елементів;

– тестування функціоналу, у якому необхідно застосовувати керування через проведення пальцем по екрану;

– тестування коректної поведінки кнопок.

Етап 4. Програмна реалізація перевірки правильної поведінки елементів на екрані «Home» за допомогою вибраних методів тестування.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Home» за допомогою функціонального тестування наведена у лістингу 3.1.

Лістинг 3.1 Реалізація перевірки правильної поведінки елементів на екрані «Home» за допомогою функціонального тестування:

```
def test_YT15_video_elements(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    ScreenActions.swipe_half_screen(self)
    home_screen.validate_video_elements()
```

Даний тест дозволяє перевірити правильність роботи кнопок навігації застосунку.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Home» за допомогою димового тестування наведена у лістингу 3.2.

Лістинг 3.2 Реалізація перевірки правильної поведінки елементів на екрані «Home» за допомогою димового тестування:

```
def test_YT11_open_app(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
```

Різниця між лістингом 3.2 і лістингом 3.1 полягає у методі тестування. Тобто, якщо уявити, що у тестовому плані є 10 тестів, які перевіряють поведінку різних кнопок керування, тоді тест, який перевіряє найголовніші кнопки і буде «димовим».

Якщо цей «димовий» тест не проходить успішно, тоді немає сенсу тестувати застосунок далі, тому що більшість тестів використовують ці «найголовніші кнопки» і наступні тести теж не пройдуть успішно.

Однак, цей же «димовий» тест є і тестом, який перевіряє функціонал «найголовніших кнопок», тому що в тесті перевіряється коректна поведінка при натисненні на головну кнопку керування. Різниця тільки у тому, що «димове» тестування має по декілька найголовніших тестів з кожного тестового плану, у якому може бути до декількох десятків тестів, а «функціональне» тестування має перевірку функцій елементів застосунку. Тож доволі часто «функціональні» тести попадають до списку «димових».

Програмна реалізація перевірки правильної поведінки елементів на екрані «Home» за допомогою автоматизованого тестування наведена у лістингу 3.3.

Лістинг 3.3 Реалізація перевірки правильної поведінки елементів на екрані «Home» за допомогою автоматизованого тестування:

```
def test_YT14_top_buttons (self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.validate_top_buttons()
```

Потрібно зазначити, що різні типи тестування, такі як «димове» та «функціональне» все ще є лише методами, але не реалізацією.

Тобто такий тип тестування як «функціональне» тестування можна провести як руками (мануально), а саме – просто натиснути на кнопку руками і особисто переконатись, що кнопка працює і виконує свою функцію, так і за допомогою засобів автоматизації (автоматизовано), а саме – написати програмний код, який сам натисне на кнопку і перевірить, що кнопка виконує свої функції.

У лістингу 3.3 наведений приклад «функціонального» тестування реалізованого способом «автоматизації» тестування.

Етап 5. Програмна реалізація перевірки правильної поведінки елементів на екрані «Search» за допомогою вибраних методів тестування.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Search» за допомогою функціонального тестування наведена у лістингу 3.4.

Лістинг 3.4 Реалізація перевірки правильної поведінки елементів на екрані «Search» за допомогою функціонального тестування:

```
def test_YT22_search_results (self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_search_button()
    search_screen = SearchScreen(self)
    search_screen.wait_for_load()
    search_screen.fill_text_field(‘Weather’)
    search_screen.validate_search_results()
```

Даний тест переходить на екран «Search», виконує пошуковий запит, та перевіряє результати пошуку.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Search» за допомогою димового тестування наведена у лістингу 3.5.

Лістинг 3.5 Реалізація перевірки правильної поведінки елементів на екрані «Search» за допомогою димового тестування:

```
def test_YT21_search_field (self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_search_button()
    search_screen = SearchScreen(self)
    search_screen.wait_for_load()
```

Даний тест переходить на екран «Search» і перевіряє наявність на ньому найважливіших елементів, без яких подальше тестування не має сенсу.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Search» за допомогою автоматизованого тестування наведена у лістингу 3.6.

Лістинг 3.6 Реалізація перевірки правильної поведінки елементів на екрані «Search» за допомогою автоматизованого тестування:

```
def test_YT25_search_videos (self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_search_button()
    search_screen = SearchScreen(self)
    search_screen.wait_for_load()
    search_item = ‘Weather’
    search_screen.fill_text_field(search_item)
    self.driver.press_keycode(66)
    home_screen.validate_video_elements()
```

Даний тест переходить на екран «Search», виконує пошуковий запит та перевіряє наявність елементів відео блоку.

Етап 6. Програмна реалізація перевірки правильної поведінки елементів на екрані «Shorts» за допомогою вибраних методів тестування.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Shorts» за допомогою функціонального тестування наведена у лістингу 3.7.

Лістинг 3.7 Реалізація перевірки правильної поведінки елементів на екрані «Shorts» за допомогою функціонального тестування:

```
def test_YT42_back_button (self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_shorts_button()
    shorts_screen = ShortsScreen(self)
    shorts_screen.wait_for_load()
    shorts_screen.tap_on_back_button()
    home_screen.wait_for_load()
```

Даний тест переходить на екран «Shorts», перевіряє чи завантажився екран, та контролює функціонал кнопки «Back».

Програмна реалізація перевірки правильної поведінки елементів на екрані «Shorts» за допомогою димового тестування наведена у лістингу 3.8.

Лістинг 3.8 Реалізація перевірки правильної поведінки елементів на екрані «Shorts» за допомогою димового тестування:

```
def test_YT41_shorts_video (self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_shorts_button()
    shorts_screen = ShortsScreen(self)
    shorts_screen.wait_for_load()
```

Даний тест переходить на екран «Shorts» і перевіряє наявність на ньому найважливіших елементів, без яких подальше тестування не має сенсу.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Shorts» за допомогою автоматизованого тестування наведена у лістингу 3.9.

Лістинг 3.9 Реалізація перевірки правильної поведінки елементів на екрані «Shorts» за допомогою автоматизованого тестування:

```
def test_YT43_validate_info (self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_shorts_button()
    shorts_screen = ShortsScreen(self)
    shorts_screen.wait_for_load()
    shorts_screen.validate_video_details()
    shorts_screen.validate_control_buttons()
```

Даний тест переходить на екран «Shorts» та перевіряє наявність на екрані необхідних елементів.

Етап 7. Програмна реалізація перевірки правильної поведінки елементів на екрані «Subscriptions» за допомогою вибраних методів тестування.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Subscriptions» за допомогою функціонального тестування наведена у лістингу 3.10.

Даний тест переходить на екран «Subscriptions», натискає на кнопку «Sign in», та перевіряє елементи у відкритому вікні.

Лістинг 3.10 Реалізація перевірки правильної поведінки елементів на екрані «Subscriptions» за допомогою функціонального тестування:

```
def test_YT51_sign_in (self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_subscriptions_button()
    subscription_screen = SubscriptionScreen(self)
    subscription_screen.wait_for_load()
    subscription_screen.tap_on_sign_in_button()
    if not subscription_screen.is_add_account_modal_displayed():
        raise Exception("'Add account' modal isn't displayed!")
```

Програмна реалізація перевірки правильної поведінки елементів на екрані «Subscriptions» за допомогою димового тестування наведена у лістингу 3.11.

Лістинг 3.11 Реалізація перевірки правильної поведінки елементів на екрані «Subscriptions» за допомогою димового тестування:

```
def test_YT52_validate_screen (self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_subscriptions_button()
    subscription_screen = SubscriptionScreen(self)
    subscription_screen.wait_for_load()
    if not subscription_screen.is_sign_in_button_displayed():
        raise Exception("Sign in' button isn't displayed!")
```

Даний тест переходить на екран «Subscriptions» і перевіряє наявність на ньому найважливіших елементів, без яких подальше тестування не має сенсу.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Subscriptions» за допомогою автоматизованого тестування наведена у лістингу 3.12.

Лістинг 3.12 Реалізація перевірки правильної поведінки елементів на екрані «Subscriptions» за допомогою автоматизованого тестування:

```
def test_YT53_reload_page(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_subscriptions_button()
    subscription_screen = SubscriptionScreen(self)
    subscription_screen.wait_for_load()
    while ScreenActions.swipe_half_screen(self, direction="up"):
        if not subscription_screen.is_reload_spinner_is_displayed():
        raise Exception("Reload spinner isn’t displayed!")
        subscription_screen.wait_for_load()
```

Даний тест переходить на екран «Subscriptions», оновлює екран та перевіряє наявність на ньому елементів, що були до оновлення.

Етап 8. Програмна реалізація перевірки правильної поведінки елементів на екрані «Settings» за допомогою вибраних методів тестування.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Settings» за допомогою функціонального тестування наведена у лістингу 3.13.

Лістинг 3.13 Реалізація перевірки правильної поведінки елементів на екрані «Settings» за допомогою функціонального тестування:

```
def test_YT31_open_settings(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_profile_button()
    settings_screen = SettingsScreen(self)
    settings_screen.tap_on_settings_button()
```

```
settings_screen.wait_for_load(
settings_list = [c for c in settings_screen.get_settings() if not c.get_attribute(‘text’) == ‘Try new features’]
for setting in settings_list:
    time.sleep(1)
setting_name = setting.get_attribute(‘text’)
settings_screen.tap_on_single_setting_button(setting_name)
    settings_screen.tap_on_back_button()
```

Даний тест переходить на екран «Settings», потім до екрану кожного налаштування та перевіряє роботу кнопки «Back».

Програмна реалізація перевірки правильної поведінки елементів на екрані «Settings» на основі димового тестування наведена у лістингу 3.14.

Лістинг 3.14 Реалізація перевірки правильної поведінки елементів на екрані «Settings» за допомогою димового тестування:

```
def test_YT32_validate_settings (self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_profile_button()
    settings_screen = SettingsScreen(self)
    settings_screen.tap_on_settings_button()
    settings_screen.wait_for_load()
        settings_screen.validate_settings()
```

Даний тест переходить на екран «Settings» і перевіряє наявність на ньому найважливіших елементів, без яких подальше тестування не має сенсу.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Settings» за допомогою автоматизованого тестування наведена у лістингу 3.15.

Лістинг 3.15 Реалізація перевірки правильної поведінки елементів на екрані «Settings» за допомогою автоматизованого тестування:

```
def test_YT33_check_app_version(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_profile_button()
    settings_screen = SettingsScreen(self)
    settings_screen.tap_on_settings_button()
    settings_screen.wait_for_load()
    settings_screen.tap_on_single_setting_button(‘About’)
    expected_version = settings_screen.expected_app_version
    actual_version = settings_screen.get_app_version_info()
    if expected_version != actual_version:
        raise Exception(f"App versions don’t match! Expected version: {expected_version} | Actual version: {actual_version}")
```

Даний тест переходить на екран «Settings», потім на екран «About», та звіряє очікувану версію застосунку з актуальною.

3.3 Впровадження методів тестування мобільних застосунків стосовно вибраної предметної області

Для запуску тестів перш за все необхідно налаштувати віртуальне середовище, в якому потрібно встановити усі необхідні бібліотеки. Зробити це можна, відкривши інтегроване середовище розробки PyCharm, та перейшовши до налаштувань IDE.

У налаштуваннях середовища необхідно перейти до розділу з інтерпретатором для відкритого проєкту (рис. 3.1).

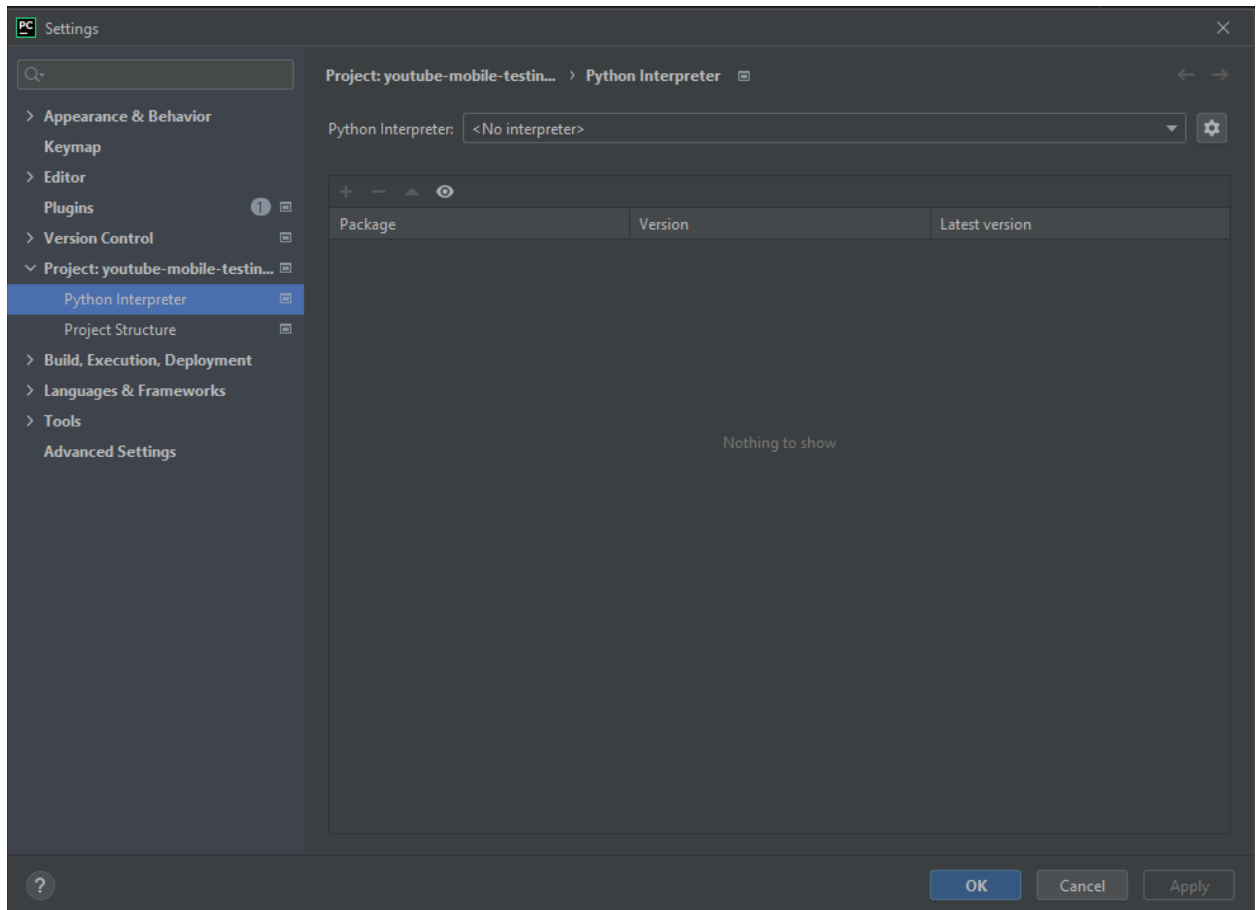

Рисунок 3.1 – Вікно налаштувань інтерпретатора проєкта

Наступним кроком буде створення віртуального середовища для відкритого проєкту. Зробити це можна, натиснувши на поле «No interpreter», а потім на кнопку «Show All..». Далі потрібно натиснути на кнопку «+», після чого відкриється вікно, у якому можна створити нове віртуальне середовище.

У пункті «New environment» треба вказати шлях до папки проєкту та вибрати базовий інтерпретатор, встановлений у операційній системі, після чого натиснути на кнопку «Ok» і IDE створить нове віртуальне середовище для відкритого проєкту.

Далі потрібно встановити усі необхідні бібліотеки для запуску тестів. Список необхідних для запуску бібліотек знаходиться у файлі «requirements.txt». Приклад усіх необхідних бібліотек для цього проєкту наведений на рисунку 3.2.

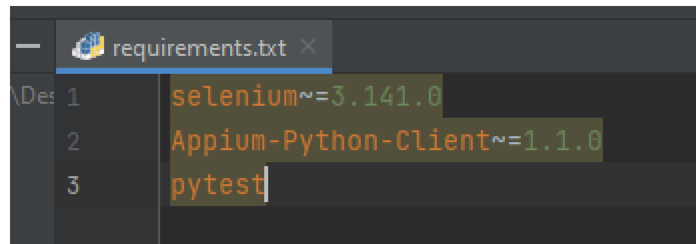

Рисунок 3.2 – Список усіх необхідних для запуску тестів бібліотек

Встановити усі бібліотеки автоматично можна скориставшись командою «pip install -r /path/to/requirements.txt». Приклад використання цієї команди наведений на рисунку 3.3.

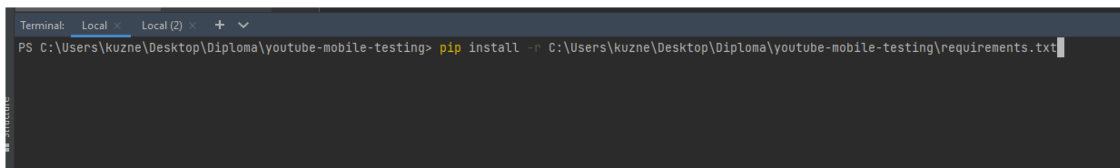

Рисунок 3.3 – Приклад виконання команди для автоматичного встановлення усіх необхідних бібліотек

Наступним кроком буде налаштування застосунку «Appium». Цей застосунок необхідний для того аби «інтерпретувати» команди програми у API запити до мобільного застосунку.

Appium можна встановити як «термінальну» версію, яка буде працювати із командним рядком, та версію із зручним інтерфейсом для користувача. Після того як Appium буде встановлено його потрібно запустити і натиснути на кнопку «Start Server». Тепер запущений застосунок може виконувати команди програми у мобільному застосунку.

Далі необхідно знайти унікальний ідентифікатор пристрою (udid – International Mobile Equipment Identifier). Якщо у якості мобільного пристрою використовувати емулятор, то його udid буде статичним – (emulator-5554).

Знайдений ідентифікатор потрібно вставити у файл config.json, де знаходяться усі параметри для запуску тестів на мобільних пристроях.

Приклад файлу config.json із вставленим унікальним ідентифікатором пристрою наведений на рисунку 3.4.

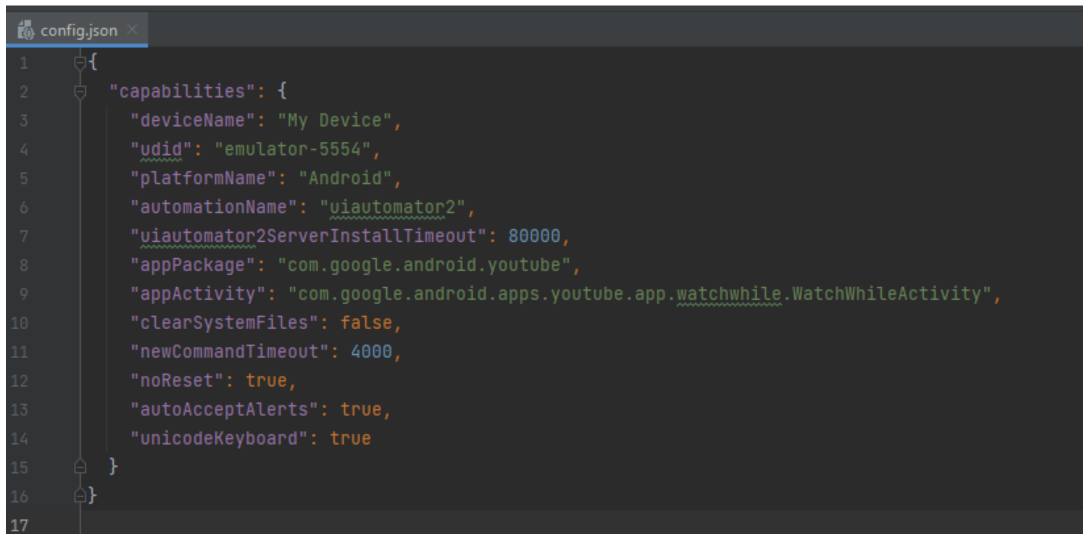

Рисунок 3.4 – Приклад файлу з параметрами запуску тестів на мобільних пристроях

Налаштувавши середовище, Appium та config.json можна перейти до виконання автоматизованих тестів. Зробити це можна, запустивши усі тести на виконання, тести у конкретному файлі або запустивши тільки один тест. Для запуску одного тесту потрібно вказати спеціальний маркер, який відповідає за запуск тільки одного тесту – «@pytest.mark.custom_test».

Приклад використання цього маркеру наведений на рисунку 3.5.

Рисунок 3.5 – Приклад використання спеціального маркеру для запуску тільки одного тесту

Запустити тести можна декількома способами, але у цій роботі наведено приклад запуску тесту засобами інтегрованого середовища розроблення.

Для запуску тесту потрібно відкрити файл із тестом та натиснути правою кнопкою миші на вкладку з файлом. Відкриється контекстне меню, у якому потрібно натиснути на зелену кнопку «Run».

Приклад запуску тесту наведений на рисунку 3.6.

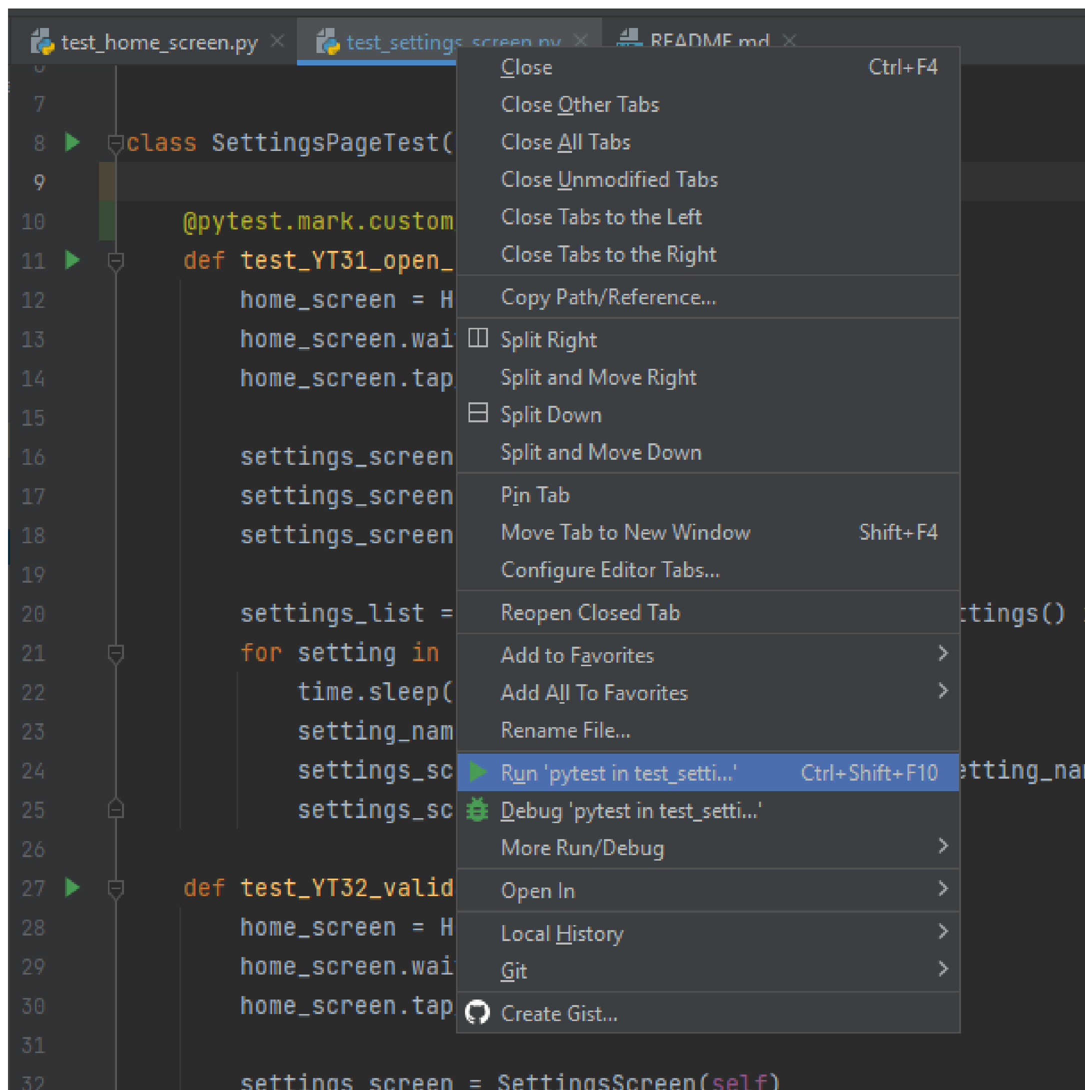

Рисунок 3.6 – Приклад запуску тесту засобами IDE

Натиснувши на кнопку «Run», почнеться виконання тестового сценарію. Приклад виконання тестового сценарію засобами інтегрованого середовища розроблення наведений на рисунку 3.7.

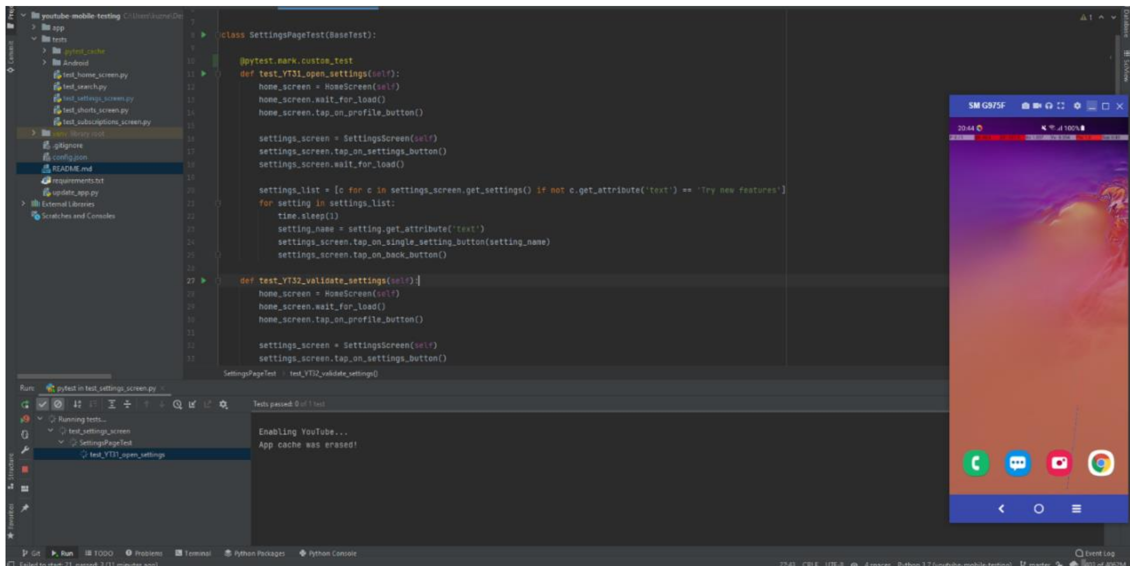

Рисунок 3.7 – Приклад виконання тестового сценарію методами IDE

Програмна реалізація перевірки правильної поведінки елементів на екрані «Home» за допомогою функціонального тестування наведена на рисунку 3.8.

```
def test_YT15_video_elements(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    ScreenActions.swipe_half_screen(self)
    home_screen.validate_video_elements()
```

Рисунок 3.8 – Приклад реалізації тесту для перевірки функції кнопки «Shorts» на головному екрані

Час виконання тестового сценарію складає 47 секунд.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Home» за допомогою димового тестування наведена на рисунку 3.9.

```
def test_YT11_open_app(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
```

Рисунок 3.9 – Приклад реалізації тесту для перевірки відкриття застосунку

Час виконання тестового сценарію складає 19 секунд.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Home» за допомогою автоматизованого тестування наведена на рисунку 3.10.

```
def test_YT14_top_buttons(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.validate_top_buttons()
```

Рисунок 3.10 – Приклад реалізації тесту для валідації кнопок керування застосунком

Час виконання тестового сценарію складає 20 секунд.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Search» за допомогою функціонального тестування наведена на рисунку 3.11.

```
def test_YT22_search_results(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_search_button()

    search_screen = SearchScreen(self)
    search_screen.wait_for_load()
    search_screen.fill_text_field('Weather')
    search_screen.validate_search_results()
```

Рисунок 3.11 – Приклад реалізації тесту для перевірки результатів пошуку

Час виконання тестового сценарію складає 25 секунд.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Search» за допомогою димового тестування наведена на рисунку 3.12.

```
def test_YT21_search_field(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_search_button()

    search_screen = SearchScreen(self)
    search_screen.wait_for_load()
```

Рисунок 3.12 – Приклад реалізації тесту для перевірки відображення екрану «Search»

Час виконання тестового сценарію складає 20 секунд.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Search» за допомогою автоматизованого тестування наведена на рисунку 3.13.

```
def test_YT25_search_videos(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_search_button()

    search_screen = SearchScreen(self)
    search_screen.wait_for_load()

    search_item = 'Weather'
    search_screen.fill_text_field(search_item)
    self.driver.press_keycode(66)

    home_screen.validate_video_elements()
```

Рисунок 3.13 – Приклад реалізації тесту
для перевірки необхідних для відео елементів

Час виконання тестового сценарію складає 25 секунд.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Shorts» за допомогою функціонального тестування наведена на рисунку 3.14.

```
def test_YT42_back_button(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_shorts_button()

    shorts_screen = ShortsScreen(self)
    shorts_screen.wait_for_load()
    shorts_screen.tap_on_back_button()

    home_screen.wait_for_load()
```

Рисунок 3.14 – Приклад реалізації тесту
для перевірки функції кнопки «Back»

Час виконання тестового сценарію складає 45 секунд.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Shorts» за допомогою димового тестування наведена на рисунку 3.15.

```
def test_YT41_shorts_video(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_shorts_button()

    shorts_screen = ShortsScreen(self)
    shorts_screen.wait_for_load()
```

Рисунок 3.15 – Приклад реалізації тесту
для перевірки відображення екрану «Shorts»

Час виконання тестового сценарію складає 42 секунди.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Shorts» за допомогою автоматизованого тестування наведена на рисунку 3.16.

```
def test_YT43_validate_info(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_shorts_button()

    shorts_screen = ShortsScreen(self)
    shorts_screen.wait_for_load()
    shorts_screen.validate_video_details()
    shorts_screen.validate_control_buttons()
```

Рисунок 3.16 – Приклад реалізації тесту
для перевірки кнопок керування застосунком

Час виконання тестового сценарію складає 36 секунд.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Subscriptions» за допомогою функціонального тестування наведена на рисунку 3.17.

```
def test_YT51_sign_in(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_subscriptions_button()

    subscription_screen = SubscriptionScreen(self)
    subscription_screen.wait_for_load()
    subscription_screen.tap_on_sign_in_button()
    if not subscription_screen.is_add_account_modal_displayed():
        raise Exception("'Add account' modal isn't displayed!")
```

Рисунок 3.17 – Приклад реалізації тесту
для перевірки функції кнопки «Sign In»

Час виконання тестового сценарію складає 21 секунду.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Subscriptions» за допомогою димового тестування наведена на рисунку 3.18.

```
def test_YT52_validate_screen(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_subscriptions_button()

    subscription_screen = SubscriptionScreen(self)
    subscription_screen.wait_for_load()
    if not subscription_screen.is_sign_in_button_displayed():
        raise Exception("'Sign in' icon isn't displayed!")
```

Рисунок 3.18 – Приклад реалізації тесту
для перевірки відображення кнопки «Sign In»

Час виконання тестового сценарію складає 20 секунд.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Subscriptions» за допомогою автоматизованого тестування наведена на рисунку 3.19.

```
def test_YT53_reload_page(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_subscriptions_button()

    subscription_screen = SubscriptionScreen(self)
    subscription_screen.wait_for_load()
    while ScreenActions.swipe_half_screen(self, direction="up"):
        if not subscription_screen.is_reload_spinner_is_displayed():
            raise Exception("Reload spinner isn't displayed!")
    subscription_screen.wait_for_load()
```

Рисунок 3.19 – Приклад реалізації тесту
для перевірки функції оновлення екрану

Час виконання тестового сценарію складає 25 секунд.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Settings» за допомогою функціонального тестування наведена на рисунку 3.20.

```
def test_YT31_open_settings(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_profile_button()

    settings_screen = SettingsScreen(self)
    settings_screen.tap_on_settings_button()
    settings_screen.wait_for_load()

    settings_list = [c for c in settings_screen.get_settings() if not c.get_attribute('text') == 'Try new features']
    for setting in settings_list:
        time.sleep(1)
        setting_name = setting.get_attribute('text')
        settings_screen.tap_on_single_setting_button(setting_name)
        settings_screen.tap_on_back_button()
```

Рисунок 3.20 – Приклад реалізації тесту для перевірки функції кнопки «Back» у кожному меню налаштування

Час виконання тестового сценарію складає 66 секунд.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Settings» за допомогою димового тестування наведена на рисунку 3.21.

```
def test_YT32_validate_settings(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_profile_button()

    settings_screen = SettingsScreen(self)
    settings_screen.tap_on_settings_button()
    settings_screen.wait_for_load()
    settings_screen.validate_setting_items()
```

Рисунок 3.21 – Приклад реалізації тесту для перевірки відображення кнопок налаштувань

Час виконання тестового сценарію складає 23 секунди.

Програмна реалізація перевірки правильної поведінки елементів на екрані «Settings» за допомогою автоматизованого тестування наведена на рисунку 3.22.

```
def test_YT33_check_app_version(self):
    home_screen = HomeScreen(self)
    home_screen.wait_for_load()
    home_screen.tap_on_profile_button()

    settings_screen = SettingsScreen(self)
    settings_screen.tap_on_settings_button()
    settings_screen.wait_for_load()

    settings_screen.tap_on_single_setting_button('About')

    expected_version = settings_screen.expected_app_version
    actual_version = settings_screen.get_app_version_info()
    if expected_version != actual_version:
        raise Exception(f"App versions don't match! Expected version: {expected_version} | Actual version: {actual_version}")
```

Рисунок 3.22 – Приклад реалізації тесту для перевірки версії застосунку

Час виконання тестового сценарію складає 22 секунди.

3.4 Порівняльний аналіз досліджених методів тестування мобільних застосунків

Задля аналізу отриманих результатів тестування мобільного застосунку застосуємо метод критеріального оцінювання за наступними критеріями:

– час виконання тестового сценарію;

– кількість відправлених на мобільний пристрій API запитів через застосунок Appium;

– якість тестування (кількість виконаних перевірок).

Порівняльний аналіз розглянутих методів тестування мобільного застосунку стосовно критерію час (у секундах) наведений у таблиці 3.1.

Порівняльний аналіз розглянутих методів тестування мобільного застосунку стосовно кількості відправлених на мобільний пристрій API запитів через застосунок Appium наведений у таблиці 3.2.

Порівняльний аналіз розглянутих методів тестування мобільного застосунку стосовно якості тестування (кількості виконаних перевірок) наведений у таблиці 3.3.

Таблиця 3.1 – Порівняний аналіз результатів стосовно критерію час

	Димове	**Функціональне**	**Автоматизоване**
Перевірка правильної поведінки елементів на екрані «Home»	19	47	20
Перевірка правильної поведінки елементів на екрані «Search»	20	25	25
Перевірка правильної поведінки елементів на екрані «Shorts»	42	45	36
Перевірка правильної поведінки елементів на екрані «Subscriptions»	20	21	25
Перевірка правильної поведінки елементів на екрані «Settings»	23	66	22

Таблиця 3.2 – Порівняний аналіз результатів стосовно кількості відправлених на мобільний пристрій API запитів

	Димове	**Функціональне**	**Автоматизоване**
Перевірка правильної поведінки елементів на екрані «Home»	1…3 Мала	4...6 Середня	1…3 Мала
Перевірка правильної поведінки елементів на екрані «Search»	1…3 Мала	4...6 Середня	4...6 Середня
Перевірка правильної поведінки елементів на екрані «Shorts»	4...6 Середня	4...6 Середня	7...10 Велика
Перевірка правильної поведінки елементів на екрані «Subscriptions»	4...6 Середня	4...6 Середня	4...6 Середня
Перевірка правильної поведінки елементів на екрані «Settings»	4...6 Середня	7...10 Велика	4...6 Середня

Таблиця 3.3 – Порівняний аналіз результатів стосовно кількості виконаних перевірок

	Димове	**Функціональне**	**Автоматизоване**
Перевірка правильної поведінки елементів на екрані «Home»	1...5 Мала	9...12 Велика	6...8 Середня
Перевірка правильної поведінки елементів на екрані «Search»	1...5 Мала	6...8 Середня	9...12 Велика
Перевірка правильної поведінки елементів на екрані «Shorts»	1...5 Мала	9...12 Велика	6...8 Середня
Перевірка правильної поведінки елементів на екрані «Subscriptions»	1...5 Мала	9...12 Велика	6...8 Середня
Перевірка правильної поведінки елементів на екрані «Settings»	1...5 Мала	9...12 Велика	6...8 Середня

Так як у третьому розділі усі типи тестування проводяться у вигляді автоматизованого методу тестування, поданий приклад «автоматизованого» тестування у таблиці 3.3 представлений у вигляді «юніт» тестування («юніт» тестування – тестування тільки якогось конкретного об’єкта у застосунку).

Проаналізувавши результати порівняльного аналізу, можна зробити висновок, що димове тестування у рази швидше за функціональне та автоматизоване. Це пов’язано з тим, що димове тестування націлене на швидке виявлення дефектів у новій збірці застосунку. Цю перевагу димового тестування дуже часто використовують у великих та середніх проєктах задля економії людських ресурсів та швидкого знаходження дефектів.

Також можна побачити, що автоматизоване (юніт) та функціональне тестування виконуються майже однаково по часу. Це можна пояснити тим, що, зазвичай, більшість елементів має тільки одну функцію, а «юніт» тестування перевіряє тільки один об’єкт.

Для фінального аналізу результатів сформуємо тестові шкали для кожного критерію.

Для критерію «час виконання тестового сценарію» бали будуть такими:

– для димового тестування: від 10 секунд до 20 секунд – 3 бали, від 20 секунд до 30 секунд – 2 бали, від 30 секунд – 1 бал;

– для функціонального тестування: від 20 секунд до 30 секунд – 3 бали, від 30 секунд до 50 секунд – 2 бали, від 60 секунд – 1 бал;

– для автоматизованого (юніт) тестування: від 20 секунд до 30 секунд – 3 бали, від 30 секунд до 50 секунд – 2 бали, від 60 секунд – 1 бал.

Для критерію «кількість відправлених на мобільний пристрій API запитів через застосунок Appium» бали будуть такими:

– для димового тестування: мала (1…3) – 3 бали, середня (4...6) – 2 бали, велика (7...10) – 1 бал;

– для функціонального тестування: мала (1…3) – 3 бали, середня (4...6) – 2 бали, велика (7...10) – 1 бал;

– для автоматизованого (юніт) тестування: мала (1…3) – 3 бали, середня (4...6) – 2 бали, велика (7...10) – 1 бал.

Для критерію «якість тестування (кількість виконаних перевірок)» бали будуть такими:

– для димового тестування: велика (9...12) – 3 бали, середня (6...8) – 2 бали, мала (1...5) – 1 бал;

– для функціонального тестування: велика (9...12) – 3 бали, середня (6...8) – 2 бали, мала (1...5) – 1 бал;

– для автоматизованого (юніт) тестування: велика (9...12) – 3 бали, середня (6...8) – 2 бали, мала (1...5) – 1 бал.

Результати оцінки за балами наведені у таблиці 3.4.

Таблиця 3.4 – Результати оцінювання тестових результатів за тестовими шкалами для кожного критерію

	Димове	**Функціональне**	**Автоматизоване**
Перевірка правильної поведінки елементів на екрані «Home»	3 + 3 + 1 = = 7 балів	2 + 2 + 3 = = 7 балів	3 + 3 + 2 = = 7 балів
Перевірка правильної поведінки елементів на екрані «Search»	3 + 3 + 1 = = 7 балів	3 + 2 + 2 = = 7 балів	3 + 2 + 3 = = 8 балів
Перевірка правильної поведінки елементів на екрані «Shorts»	1 + 2 + 1 = = 4 бали	2 + 2 + 3 = = 7 балів	2 + 1 + 2 = = 5 балів
Перевірка правильної поведінки елементів на екрані «Subscriptions»	3 + 2 + 1= = 6 балів	3 + 2 + 3 = 8 = балів	3 + 2 + 2 = = 7 балів
Перевірка правильної поведінки елементів на екрані «Settings»	2 + 2 + 1 = = 5 балів	1 + 3 + 3 = = 7 балів	3 + 2 + 2 = = 7 балів

Відштовхуючись від отриманих результатів можна зробити висновок, що найоптимальнішим тестуванням, з точки зору співвідношення витраченого часу, кількості запитів на мобільний пристрій та кількості перевірок, є:

– для перевірки правильної поведінки елементів на екрані «Home» – будь-яке з наведених, тому що це головний екран мобільного застосунку і саме на цьому екрані необхідно дуже мало часу на перевірку наявності елементів, мало перевірок та внаслідок цього мало запитів до мобільного пристрою;

– для перевірки правильної поведінки елементів на екрані «Search» – автоматизоване (юніт);

– для перевірки правильної поведінки елементів на екрані «Shorts» – функціональне тестування;

– для перевірки правильної поведінки елементів на екрані «Subscriptions» – автоматизоване (юніт);

– для перевірки правильної поведінки елементів на екрані «Settings» – автоматизоване (юніт) або функціональне, тому що кількість елементів налаштувань однакова.

На основі отриманих результатів можна зробити висновок, що, розглядаючи проєкт, створений під час дослідження кваліфікаційної роботи, раціонально буде використовувати «суміш» автоматизованого (у цьому випадку – «юніт») та функціонального тестувань. Так як має місце мале наповнення проєкту тестами, та через те, що більшість тестів була націлена саме на функціонал застосунку, з наведених таблиць та результатів аналізу стає очевидно, що саме ці два типи тестувань більше за все підходять для даного проєкту. Але треба зауважити, що не слід недооцінювати димове тестування, тому що, якщо продовжити розвивати цей проєкт та наповнювати його тестами, то рано чи пізно коефіцієнт «оптимальності» цього тестування почне зростати відповідно до кількості створених тестів.

Таким чином, рано чи пізно саме димове тестування може стати найоптимальнішим, тому що саме це тестування дозволить у короткий проміжок часу дізнатись статус про працездатність усього проєкту та рівень якості розглянутого застосунку.

3.5 Перспективи подальшої роботи

Розглянуті у даній роботі три методи тестування наявно демонструють актуальність кожного з них щодо запропонованого проєкта, але методів тестування мобільних застосунків існує дуже багато. Проте, деякі з методів не має сенсу використовувати у даному випадку, тому що досліджений проєкт знаходиться на початковій стадії.

У перспективі подальшої роботи, задля використання більшої кількості методів тестування мобільних застосунків, потрібно розширювати функціональні можливості проєкту та створювати більшу кількість тестових сценаріїв. Такі кроки дозволять додати до заключної таблиці та запропонованих шкал більше вхідних даних, що забезпечить підвищення точності аналізу на знаходження найоптимальнішого методу тестування.

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений та реалізований проєкт автоматизації тестування мобільного застосунку, який працює на принципі взаємодії людини з медіа-контентом.

Виконано всі поставлені задачі, а саме:

– досліджено роль тестування у життєвому циклі розроблення програмного забезпечення для мобільних застосунків, переваги впровадження тестування в життєвий цикл програмного забезпечення та роль тестування у кожній ітерації життєвого циклу;

– досліджено існуючі методи тестування та розроблення, такі як водоспадна, V-подібна, ітераційно інкрементальна та спіральна. Наведені схеми кожної із моделей, розглянуті переваги та недоліки цих методів тестування;

– обрано такі методи тестування мобільного застосунку: «димове», «функціональне» та «автоматизоване». Саме вони допомогли якнайкраще зрозуміти мету тестування на прикладі розглянутого застосунку;

– змодельовано етапи процесу тестування у форматі IDEF3, наведено приклад створення вимог;

– досліджено кожний з обраних методів тестування мобільного застосунку, вказано переваги та недоліки, наведено блок-схеми та лістинги програмних кодів;

– проаналізовано непередбачувані ситуації під час виконання досліджуваних методів тестування, наведено приклади, досліджено переваги та недоліки кожного з методів у розглянутих ситуаціях;

– обрано інструментальні засобі для реалізації проєкту автоматизації мобільного застосунку. Описано переваги цих засобів та їх можливості;

– розроблено проєкт мобільної автоматизації щодо мобільного застосунку «YouTube», в основі якого лежить структуроване розбиття кожного екрану застосунку на окремі тестові плани;

– проведено тестування мобільного застосунку, за результатами якого зібрано детальну інформацію щодо часу виконання кожного тесту, кількості перевірок та кількості відправлених запитів до мобільного пристрою. На основі цієї інформації виявлено найоптимальніші методи тестувань для кожного з екранів:

1) перевірка правильної поведінки елементів на екрані «Home»: димове, функціональне або автоматизоване тестування;

2) перевірка правильної поведінки елементів на екрані «Search»: автоматизоване тестування;

3) перевірка правильної поведінки елементів на екрані «Shorts»: функціональне тестування;

4) перевірка правильної поведінки елементів на екрані «Subscriptions»: функціональне тестування;

5) перевірка правильної поведінки елементів на екрані «Settings»: функціональне або автоматизоване тестування;

– визначено перспективи дальшої роботи.

Результати роботи апробовано у вигляді 2 тез доповідей під час міжнародних науково-практичних конференцій «PROBLEMS OF MODERN SCIENCE AND PRACTICE» [45] та «TRENDS IN SCIENCE AND PRACTICE OF TODAY» [46].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Єщенко, М. (2020). Автоматизоване тестування мобільних застосувань.
2. Сенеджук, А.Ю. (2021). Комп’ютерні засоби для автоматизації тестування додатків (Bachelor’s thesis, КПІ ім. Ігоря Сікорського).
3. Мурадосилова, С.Н., & Сейдаметова, З.С. (2017). Тестирование мобильных продуктов: разработка тестовых вариантов на примере приложения FesCam. Информационно-компьютерные технологии в экономике, образовании и социальной сфере, (2), 13-18.
4. Балашкова, Е.М. (2020). Особенности тестирования мобильных приложений.
5. Соколов, С.А., & Шутов, А.В. (2018). Особенности при тестировании мобильных приложений. Постулат, (6).
6. Молько, О.Д., & Иванова, Е.А. (2019). ОТЛИЧИТЕЛЬНЫЕ ОСОБЕННОСТИ ТЕСТИРОВАНИЯ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ. ТЕХНИЧЕСКИЕ НАУКИ, 17.
7. Абдулназаров, Ф.М., & Анарбеков, О.А. (2016). Тестирование мобильных приложений, функционирующих на Android. In Современные технологии поддержки принятия решений в экономике: сборник трудов III Всероссийской научно-практической конференции студентов, аспирантов и молодых ученых, 24-25 ноября 2016 г., г. Юрга. Томск, 2016. (pp. 196-198).
8. Tian, J. (2005). Software quality engineering: testing, quality assurance, and quantifiable improvement. John Wiley & Sons.
9. Винниченко, И.В. (2005). Автоматизация процессов тестирования. Издательский дом «Питер».
10. Daradkeh Y.I., and 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), pp. 43-50.

11. Tvoroshenko I.S., and Gorokhovatsky V.O. (2020) Effective tuning of membership function parameters in fuzzy systems based on multi-valued interval logic, *Telecommunications and Radio Engineering*, 79(2), pp. 149-163.

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

13. Gorokhovatskyi V.O., Tvoroshenko I.S., and Vlasenko N.V. (2020) Using fuzzy clustering in structural methods of image classification, *Telecommunications and Radio Engineering*, 79(9), pp. 781-791.

14. 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.

15. 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.

16. Daradkeh, Y.I., Tvoroshenko, I., Gorokhovatskyi, V., Latiff, L.A., and Ahmad, N. (2021) Development of Effective Methods for Structural Image Recognition Using the Principles of Data Granulation and Apparatus of Fuzzy Logic, *IEEE Access*, 9, pp. 13417-13428.

17. 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. С. 92-96.

18. Кучеренко, Е.И., Творошенко, И.С. (2003) Процессы принятия решений в сложных системах на основе нечетких интервальных представлений. *Вісник Національного технічного університету «ХПІ». Тематичний випуск: Системний аналіз, управління та інформаційні технології. – Х.: НТУ «ХПІ»*, *1*(7), 79-86.

19. Кучеренко, Є.І., Творошенко, І.С. (2011) Оперативне оцінювання простору станів складних розподілених об’єктів з використанням нечіткої інтервальної логіки. *Искусственный интеллект*. № 3. С. 382-387.

20. 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.

21. 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.

22. Бодянский, Е.В., Кучеренко, Е.И., Творошенко, И.С. (2004). О синтезе нечетких алгоритмов на основе композиции фрагментов правил и моделей. *АСУ и приборы автоматики*, (128), 19-28.

23. Tvoroshenko, I., & Kukharchuk, V. (2021). Current state of development of applications for recognition of faces in the image and frames of video captures.

24. Tvoroshenko, I., & Temchur, K. (2021). Features of software application development for food recognition using deep machine learning methods.

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

26. I. Tvoroshenko (2020). Information technologies for decision-making on the conditions of spatially distributed objects, in Abstracts of I International Scientific and Practical Conference. Problems and perspectives of modern science and practice, Austria. pp. 45-50.

27. Творошенко, И.С., Мгеброва, В.Р., & Белый, В.В. (2015). Практические аспекты применения современных геоинформационных систем для создания муниципальной геоинформационной системы города Харькова. *Системи обробки інформації*, (7), 65-70.

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

29. Tvoroshenko, I., & Andrieieva, A. (2021). Development of web applications for remote learning of English.

30. Творошенко, И.С., Дехтярь, А.П. (2005, June) Информационные технологии в задачах компьютерной диагностики с использованием интеллектуальных систем. In *Клиническая информатика и Телемедицина. Компьютерная Медицина–2005: материалы междунар. научн.-технич. конф., Харьков* (p. 138).

31. Творошенко, І.С., Мгеброва, В.Р., & Білий, В.В. (2016). Практичні аспекти створення вихідної інформації для проведення геоінформаційного аналізу у сфері управління нерухомістю.

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

33. Кобилін О.А., Творошенко І.С. (2021). Методи цифрової обробки зображень: навч. посібник. *Харків: ХНУРЕ*.

34. Tvoroshenko, I., & Almakaieva, A. (2020). Application of procedural generation of game content using software algorithms.

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

36. 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.

37. 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.

38. Kobylin O., Gorokhovatskyi V., Tvoroshenko I., and Peredrii O. (2020) The application of non-parametric statistics methods in image classifiers based on structural description components, *Telecommunications and Radio Engineering*, 79(10), pp. 855-863.

39. Gorokhovatskyi V., and Tvoroshenko I. (2020) Image Classification Based on the Kohonen Network and the Data Space Modification, *In CEUR Workshop Proceedings: Computer Modeling and Intelligent Systems (CMIS-2020)*, 2608, pp. 1013-1026.

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

41. Tvoroshenko, I.S. (2004) Structure and functions of intelligent decision-making tools in complex systems. *Artificial Intelligence*, *4*, 462-470.

42. Turnquist, G.L. (2011). Python testing cookbook (pp. 117-163). Packt Publishing.

43. Arbuckle, D. (2010). Python Testing: Beginner's Guide. Packt Publishing Ltd.

44. Pajankar, A. (2017). Python Unit Test Automation: Practical Techniques for Python Developers and Testers. Apress.

45. Tvoroshenko I., and Kuznetsov M. (2021) About the role of testing in process of mobile application development, *Abstracts of I International scientific-practical conference «Problems of modern science and practice» (September 21-24, 2021). Boston, USA*, pp. 416-421.

46. Tvoroshenko I., and Kuznetsov M. (2021) Research results of functional, white box and smoke testing methods for mobile applications, *Abstracts of V International scientific-practical conference «Trends in science and practice of today» (October 19-22, 2021). Ankara, Turkey*, pp. 418-422.