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 ВЕБТЕСТУВАННЯ
(тема)

Виконав:
студент 2 курсу, групи ІТКСм-20-1
Максіменко Г.А.
(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 Максіменку Георгію Артуровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та порівняння методів вебтестування

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, середовище розробки WebStorm, мова програмування JavaScript, фреймворк Cypress.

4. Перелік питань, що потрібно опрацювати в роботі

1. Класифікація та аналіз сучасних методів вебтестування.
2. Моделювання процесу вебтестування у стандарті IDEF0.
3. Розробка та реалізація проєкту з тестування вебзастосунка.
4. Аналіз отриманих результатів та висновки щодо можливості продовження розвитку проєкту.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність дослідження, об'єкт та мета 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	23.10.21-26.10.21	
3	Аналіз літератури з досліджуваної проблеми	27.10.21-29.11.21	
4	Дослідження обраних методів	30.10.21-31.10.21	
5	Застосування обраних методів	01.11.21-03.11.21	
6	Програмна реалізація	04.11.21-08.11.21	
7	Оформлення пояснювальної записки	09.11.21-24.11.21	
8	Перевірка на плагіат	25.11.2021	
9	Рецензування	26.11.2021	
10	Підготовка презентації та доповіді	28.11.2021	
11	Занесення роботи в електронний архів	30.11.2021	
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: 100 с., 7 табл., 43 рис., 59 джерел.

ВЕБТЕСТУВАННЯ, ВЕБЗАСТОСУНОК, МАНУАЛЬНЕ ТЕСТУВАННЯ, АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, МОВА ПРОГРАМУВАННЯ, ФРЕЙМВОРКИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ.

Об’єктом роботи є процес тестування вебзастосунків.

Метою роботи є дослідження та порівняння вибраних методів вебтестування задля визначення їх ефективності під час тестування окремих класів задач.

Використано методи вебтестування на макро- та мікрорівнях. Проведено дослідження методів вебтестування з метою визначення переваг та недоліків у подальшому застосуванні. Детально вивчено інтеграційний, модульний та регресійний методи вебтестування як найактуальніші при подальшому використанні у процесі вебтестування.

У результаті роботи здійснена програмна реалізація проєкту з тестування вебзастосунку предметної області онлайн-торгівлі.

WEB TESTING, WEB APPLICATION, MANUAL TESTING, AUTOMATED TESTING, PROGRAMMING LANGUAGE, AUTOMATED TEST FRAMES.

The object of the work is the process of testing web applications.

The aim of the work is to study and compare selected methods of web testing to determine their effectiveness when testing individual classes of tasks.

Methods of web testing at macro and micro levels are used. A study of web testing methods was conducted to determine the advantages and disadvantages of further application. Integration, modular and regression methods of web testing are studied in detail as the most relevant for further use in the web testing process.

As a result of the work, a software implementation of the project on testing the web application of the subject area of online commerce was carried out.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

QA – Quality Assurance (контроль якості)

SQL – Structured Query Language (структурована мова запитів)

ПЗ – програмне забезпечення

GUI – Graphic User Interface (графічний інтерфейс користувача)

GIT – Global Information Tracker (глобальна система контролю інформації)

GET та POST запити – основні види HTTP запитів

HTTP – Hypertext Transfer Protocol (протокол перенесення гіпертексту)

БД – база даних

CI – Continious Integration (безперервна інтеграція)

HTML – HyperText Markup Language (мова гіпертекстової розмітки)

IDE – Integrated Development Enviroment (інтегроване середовище розроблення)

BDD – Behavior Driven Development (поведінкове розроблення)

ВСТУП

У сучасному світі Інтернет та різноманітні інформаційні системи усередині нього використовуються всюди, тому значною частиною стабільної діяльності вебзастосунків по всьому світу є його тестування.

На тестування великих вебзастосунків (таких, наприклад, як Facebook, Google та ін.) виділяються великі кошти, але навіть значний бюджет не може гарантувати того, що тестування буде проведено на найвищому рівні, бо навіть найкращі розробники допускають помилки у системі, а експерти можуть їх пропустити, виконуючи процес тестування у ручному режимі.

Ручна перевірка якості системи, її компонентів та взаємодії окремих модулів має великий недолік, а саме необхідність великої кількості часу та ресурсів на перевірку (насамперед, якщо мова йде про застосунки великого розміру). Вирішенням даної проблеми є автоматизоване тестування, використовуючи яке компанія-розробник застосунку, зменшує витрати на розроблення та оновлення системи через швидкість виконання, можливість повторного використання, охоплення різноманітних кейсів та більш високої точності через зменшення ймовірності людської помилки.

Отже, необхідним є проаналізувати сучасну інформацію з QA діяльності, щоб визначити елементи, які стосуються реалізації тестування вебзастосунків, виявити можливості для введення визначених елементів у процес вебтестування з подальшим створенням тестового проєкту та аналізом отриманих результатів, які дозволять зробити висновок щодо переваг та недоліків вебтестування та можливостей для подальшого розвитку процесу вебтестування за допомогою обраних методів та тестових задач.

1 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ТЕСТУВАННЯ

1.1 Роль тестування під час розроблення вебзастосунків

Тестування – це перевірка якості програмного забезпечення згідно з документацією до даного програмного забезпечення. Тестування може проводитися на застосунках будь-якої сфери діяльності (від онлайн магазинів до застосунків прийняття рішень на базі нечіткої логіки) [1]. Основна задача тестувальника – зменшити кількість неприємних сюрпризів для кінцевого користувача.

Основна мета тестування – перевірка функціонування ресурсу на відповідність до висунутих вимог. Перевірка застосунку без користування ним здається неможливим, але тестувальники спеціально створюють штучні ситуації, які можуть в майбутньому виникнути в роботі з ресурсом. Проводиться аналіз поведінки ресурсу на запропонованих умовах. Коли фахівець виявляє системну помилку) він передає свій звіт менеджеру з проєкту, який в подальшому розподіляє роботу по усуненню помилок серед інших учасників проєкту. Коли помилки усувають, сайт знову тестують. Тестування буде проводитися до тих пір, поки сайт не стане ідеальним.

Розглянемо безпосередньо тестування вебсайтів та вебсторінок.

У процесі вебтестування має бути розглянута ефективність застосунку у відповідності очікуванням кінцевого користувача [2].

Зовнішні і внутрішні характеристики відображають властивості самого ПЗ (працюючого або не працюючого), а також погляд замовника і розробника на таке ПЗ. Безпосереднього кінцевого користувача ПЗ цікавить експлуатаційна якість ПЗ – сукупний ефект від досягнення характеристик якості [2].

Гібридне розроблення застосунку, тестування вебсайту, як і тестування будь-якого компоненту в цілому, це механізм контролю якості, що перевіряє відповідність між реальною і очікуваною поведінкою сторінки [3].

Якість програмного забезпечення (ПЗ) – набір властивостей продукту (сервісу або програм), що характеризують його здатність задовольнити встановлені або передбачувані потреби замовника. Поняття якості має різні інтерпретації залежно від конкретної програмної системи і вимог до неї. Крім того, моделі оцінки якості мають різну кількість рівнів і повністю або частково збігаються щодо набору характеристик якості. Наприклад, модель якості МакКолла має 3 характеристики (функціональність, модифікованість, ревізійність), 11 підхарактеристик якості і 18 критеріїв.

Стандарт ISO 9126:2011 регламентує зовнішні і внутрішні характеристики якості [4].

Якість не є абсолютною, це суб'єктивне поняття. Тому тестування, як процес своєчасного виявлення помилок та дефектів, не може повністю забезпечити коректність роботи сторінки. Вимоги до поняття якості описані у стандарті ISO 9126 [4].

Склад та зміст необхідної для тестування документації наводиться у стандарті IEE 829-2008 [5].

Базовими поняттями тестування у процесі розроблення є наступні:

– верифікація (verification) – це процес оцінки системи або її компонентів з метою визначення відповідності процесу розроблення до плану розроблення;

– валідація (validation) – це визначення відповідності розроблюваного програмного забезпечення між очікуваннями і потребами користувача [6].

Верифікація та валідація програмного забезпечення мають найвище значення у таких застосунках, як, наприклад, для прийняття рішень у комплексних системах [6].

Ефективнішим у процесі розроблення є автоматизоване тестування.

Автоматизоване тестування програмного забезпечення є частиною процесу тестування на етапі контролю якості в процесі розроблення програмного забезпечення. Воно використовує програмні засоби для виконання тестів і перевірки результатів виконання, що допомагає скоротити час тестування.

Градація тестування ПЗ у процесі розроблення вебпрограмного забезпечення:

– тестування частин ПЗ (модулів, компонентів) з метою перевірки актуальності реалізації алгоритмів виконується розробниками;

– функціональне тестування підсистем, ПЗ в цілому з метою з'ясувати рівень виконання функціональних вимог до ПЗ – рекомендується проводити окремою групою тестувальників, не підпорядкованої керівнику розроблення;

– навантажувальне тестування для виявлення характеристик функціонування ПЗ при зміні навантаження (інтенсивності звернень до нього, наповнення бази даних і т.п.), наприклад, у програмних застосунках що класифікують зображення [7], для виконання цієї роботи потрібні висококваліфіковані спеціалісти.

Основні об'єкти автоматизації тестування – системи, що реалізують роботу клієнтської частини. Ключовою особливістю тестування клієнт–серверних систем є можливість перевірки коректності функціонування та задовільної швидкодії системи через роботу клієнтської частини. Таким чином, ретельно і всебічно перевіривши ці можливості, отримано гарантію працездатності системи.

Ще один важливий аспект організації робіт – зберігання створених тестів. Рекомендовано ставитися до тестів так само, як і до вихідного коду, тобто потрібно використовувати версійність з метою відтворення його у разі необхідності (наприклад, GIT).

Це стане в нагоді на етапі супроводу ПЗ і дасть можливість повторного використання готових тестів на декількох версіях системи [7].

Також тестування використовується у процесі створення не тільки вебсторінок але й вебзастосунків.

Будь-який вебзастосунок представляє набір статичних і динамічних сторінок. Статична вебсторінка – це сторінка, яка завжди відображається перед користувачем в незмінному вигляді. Сервер відправляє сторінку за запитом веббраузера без будь-яких змін. На противагу цьому, сервер вносить зміни в динамічну сторінку перед відправкою її браузеру. У зв'язку з тим, що сторінка змінюється, вона називається динамічною [8].

Майже всі сучасні програми орієнтовані на роботу з мережею. Зберігання даних вебзастосунків здійснюється, переважно, на сервері, обмін інформацією відбувається по мережі. Коли користувач виявляє помилку в мережевому середовищі, то часто складно точно вказати, де саме вона відбулася, і тому режим роботи, або повідомлення про помилку, яке отримано, може бути результатом помилок, що сталися в різних частинах мережевої систем, наприклад, помилка у геоінформаційній системі може статися усередині навігаційної системи [9].

До основних вразливостей вебзастосунків відносяться:

– повна відсутність перевірки вхідних даних (у вебформах будь-яких систем) або тільки часткова перевірка даних;

– некоректна обробка вхідних даних (нульовий байт, символи рівня директорій);

– переповнення буферу;

– необережна робота програми з файлами, у випадку коли ім’я файлу передається програмі ззовні (GET або POST);

– неврахування особливостей GET та POST запитів;

– некоректна робота з паролями (під час зберігання, передачі та обробки даних);

– неправильні права доступу;

– неправильні права програм на сервері;

– неврахування особливостей роботи програм завантаження файлів на сервер;

– некоректна логіка роботи вебпрограми, яка при деяких допустимих вхідних даних приводить до непередбачуваних наслідків;

– виведення інформації при помилках програми або доступу до бази даних, коли виводиться додаткова службова інформація, не призначена для сторонніх очей;

– некоректна робота з базами даних (паролі, виведення службової інформації, завищена кількість запитів до БД);

– вразливості недостатньої обробки вхідних даних при роботі з базою даних (SQL-ін’єкції);

– неоптимізований програмний код, котрий приводить до значних навантажень на вебсервер (при своїй звичайній роботі та особливо у випадку збою при передачі некоректних вхідних даних);

– вразливість вебзастосунків та систем до DoS та DDoS атак [10].

Через вразливість вебзастосунків до хакерських атак необхідно приділяти найбільшу увагу тестуванню платформ, від яких напряму залежить діяльність суспільства, наприклад, програмне забезпечення для багатофункціонального геоінформаційного довідника, що підтримує державну пожежно-рятувальну частину за допомогою сучасних геоінформаційних технологій або тривимірні моделі ландшафтних парків.

Зі збільшенням залежності компаній будь-якого напрямку діяльності від IT-технологій, гостро постає питання забезпечення інформаційної безпеки. Одним з ключових заходів в забезпеченні інформаційної безпеки компанії є тестування на проникнення. Це дозволяє упевнитися в надійності захисту від несанкціонованого доступу та інших загроз інформаційної безпеки [11].

На даний час найбільш розповсюдженими методологіями проведення тестування на проникнення є:

– The Open Source Security Testing Methodology Manual (OSSTMM);

– The National Institute of Standards and Technology (NIST) Special Publication 800-115;

– OWASP Testing Guide;

– Penetration Testing Execution Standard (PTES);

– Information Systems Security Assessment Framework (ISSAF);

– BSI – Study A Penetration Testing Model for intensive development [12].

Конфігураційне керування (Software Configuration Management, SCM) в програмній інженерії – комплекс методів, спрямованих на систематичний облік змін, що вносяться розробниками в програмний продукт у процесі його

розроблення та супроводу, збереження цілісності системи після змін, запобігання небажаних і непередбачуваних ефектів, формалізація процесу внесення змін, наприклад, конфігураційне тестування у геоінформаційному застосунку паркової системи [13] буде проведене у процесі картографічних змін.

Найпопулярніші сервери безперервної інтеграції програмного забезпечення (табл. 1.1).

Таблиця 1.1 – Аналіз сервісів інтеграції ПЗ

Сервер	**Опис**
Jenkins	Інструмент безперервної інтеграції, написаний на Java. Запускається в контейнері сервлетів, таких як Apache Tomcat або GlassFish. Підтримує інструментарій для роботи з різними системами контролю версій, включаючи CVS, Subversion, Mercurial, Git і Clearcase, може збирати проєкти Apache Ant і Apache Maven, а також виконувати скрипти і команди Windows, наприклад, у системах прийняття рішень [14].
BuildBot	BuildBot написаний в основному на Python і заснований на Twisted фреймворку. Починався як більш легка альтернатива Mozilla Tinderbox і зараз використовується в таких проєктах як Mozilla, Chromium, Webkit і декількома іншими. Buildbot встановлюється у вигляді основного сервера (master) і підлеглих (slave). З основного сервера здійснюється моніторинг змін в репозиторії, координація роботи підлеглих серверів і розсилка звітів користувачам і розробникам. Підлеглий сервер може запускатися під різними операційними системами. Налаштовується за допомогою конфігураційних скриптів Python на основному сервері. Ці скрипти для настройки компонентів збірки дуже прості в розумінні, однак володіють всією потужністю Python [15].

Продовження таблиці 1.1

Сервер	**Опис**
CruiseControl	Інструмент безперервної інтеграції програмного забезпечення на платформі Java, націлений на автоматизацію процесу складання. По збірки здійснюється через вебінтерфейс. Інтегрується з Apache Ant, різними системами управління версіями [15]. Є відкритим програмним забезпеченням, поширюється під BSD-подібною ліцензією.
GoCD	Як і інші просунуті CI сервери, GoCD дозволяє виконувати збірки в різних системах і контролювати їх в одному місці. Регулярно виконувані дії можуть бути додані в джерела і потім вони викликаються для виконання. GoCD має простий і зрозумілий інтерфейс, а також детальну документацію [15].
TeamCity	Серверне програмне забезпечення від компанії JetBrains написане на мові Java, сервер побудови для забезпечення безперервної інтеграції [15].
Travis CI	Ймовірно найпростіший CI сервер, для того щоб почати. Крім того, що він розповсюджується з відкритим вихідним кодом і відповідно безкоштовний при установці на свій сервер. Travis CI має SaaS версію, яка є безкоштовною для проєктів з відкритим кодом [15]. При реєстрації та налаштування просто потрібно закріпити свій GitHub аккаунт, отримати необхідні права і додати .travis.yaml файл в своєму проєкті. Travis CI збере нову збірку після додавання змін на GitHub.

1.2 Класифікація існуючих методів тестування

1.2.1 Мануальні види тестування

Мануальне тестування – є по факту найпростішим видом тестування, коли програмний продукт перевіряється вручну спеціалістами Manual Quality Assurance на відповідність вимогам програмного забезпечення та стандартам якості продукту [16].

У більшості проєктів мануальне тестування є основним видом тестування. Навіть при реалізації автоматизації основних тестових сценаріїв, виключення даного етапу неможливе. Всі автоматичні тести пишуться на основі мануальних кейсів і спершу перевіряються в ручну.

Дослідницьке тестування є одним із підходів до тестування, найчастіше мануального. У деяких проєктах воно може виявитися більш продуктивним, ніж звичайне тестування за сценаріями, під час яких пишуться тест плани та тест кейси [16].

Визначення дослідницького тестування можна дати наступне – це написання тестової документації і виконання тестування в один і той же час, вже після того як розробники закінчили частину функціоналу, що може потрапити в реліз. Такий підхід є протилежністю або доповненням сценарного підходу, що був описаній вище, коли тестувальник маючи задокументовані вимоги до продукту підключається на етапі проєктування або вже розроблення, для написання тестових сценаріїв. Дослідницькі тести, на відміну від сценарних тестів, не визначені попередньо, і часто не задокументовані, можуть бути представленні в неповністю деталізованих чек-листах, що пишуться в самим тестувальником, і не виконуються в точній відповідності з планом.

Навіть за наявності документації, добре пророблених тестових сценаріїв, в процесі безпосереднього тестування, тестувальник буде відходити частково від сценаріїв, для роботи з великою кількістю деталей (наприклад, як швидко друкувати на клавіатурі, або які види поведінки

програми визнати помилковими). Крім того, спершу аби познайомитися з функціоналом застосунку підходить саме дослідницьке тестування, так як під час проєктування, та написання тестової документації, без ознайомлення на практиці з продуктом, важко образу включити тестувальника в роботу над тестування по сценаріях.

1.2.2 Автоматизовані види тестування

Автоматичне тестування це частина процесу тестування на етапі контролю якості в процесі розроблення програмного забезпечення. Воно використовує програмні засоби для виконання тестів і перевірки результатів виконання, що допомагає скоротити час тестування і спростити процес [17].

Головна мета автоматизації – прискорити процес тестування не втрачаючи якість а також максимально знизити кількість рутини у написанні та реалізації тест-кейсів.

Існує кілька рівнів автоматичного тестування:

– Unit tests;
– Integration tests;
– GUI tests [18].

Unit і Integration tests розробляються розробникам, Manual тестування здійснюють тестувальники. Варто зупинитися більш детально на GUI-автоматизації, що набирає популярність величезними темпами і може в рази зменшити об’єм робіт з Manual тестування.

GUI-автоматизація – найбільш поширений вид автоматизації тестування шляхом тестування програми через графічний інтерфейс користувача (GUI). Головна його перевага в тому, що застосунок тестують точно так, як його буде використовувати кінцевий користувач. Також цей підхід дозволяє тестувати без доступу до вихідного коду програми [18].

Основні плюси GUI автоматичного тестування [18]:

– виключений людський фактор – автоматизація гарантує, що тест-скрипт завжди буде виконаний однаково, виключаючи помилки з необережності;

– швидкість виконання – час виконання автоматизованого тест-скрипту в рази менше мануального проходження;

– менші витрати на підтримку – один раз написані скрипти вимагають менше часу на підтримку і аналіз результатів;

– звіти – в результаті прогону генерується звіт з подальшим розсиланням всім зацікавленим особам;

– виконання тестів у зручний час – автоматичні тести можуть бути запущені в будь-який зручний час або по певній події, часто це нічні прогони і тестування в неробочий час, що дозволяє раціональніше використовувати тест-ресурси.

Загальна класифікація видів тестування наведена у таблиці 1.2.

Таблиця 1.2 – Класифікація видів тестування

Ознака тестування	**Види тестування**
За розташуванням програмно-апаратних засобів	Зовнішнє
	Внутрішнє
За доступом до коду	Білого ящику
	Сірого ящику
	Чорного ящику
За обізнаністю про проведення тестування	Відкрите
	Приховане
За повнотою виконання тестування	Повне
	Обмежене
	Сфокусоване
За видом	Мануальне
	Автоматизоване

Основні мінуси GUI автоматичного тестування [18]:

– на написання навіть невеликого набору автоматичних тестів бути витрачено багато часу;

– процес розроблення автоматичних тестів може бути запущеній тільки після закінчення розроблення, оскільки якщо писати автоматичні кейси на проміжній версії функціоналу, що буде дороблюватися, велика вірогідність того, що автоматичні тести необхідно буде перероблювати;

– при зміні функціоналу застосування майже завжди необхідно буде проводити рефакторинг автотестів.

Тестування починається не з того моменту, коли вже готовий робочий вебзастосунок, а раніше – як тільки почалися погоджуватися специфікації та вимоги до програмного продукту.

1.2.3 Тестування на проникнення

Також важливою перевіркою є перевірка на проникнення. Вебзастосунки можна перевірити на проникнення двома способами. Тести можуть бути розроблені для імітації внутрішньої або зовнішньої атаки.

Тестування на внутрішнє проникнення, як випливає з назви, внутрішнє тестування пера проводиться в організації через локальну мережу, отже, воно включає тестування вебзастосунків, розміщених в інтрамережі. Це допомагає з’ясувати, чи можуть бути вразливі місця, що існують у корпоративному брандмауері.

По суті, це включає зловмисні атаки працівників незадоволених співробітників або підрядників, які звільнилися б, але, знаючи про внутрішню політику безпеки та паролі, атаки соціальної інженерії, моделювання зловмисних атак та атаки з використанням привілеїв користувача або неправильне використання розблокованого терміналу.

Тестування зовнішнього проникнення – ці атаки проводяться зовні поза організацією та включають тестування вебзастосунків, розміщених в Інтернеті.

Тестери поводяться як хакери, які не дуже обізнані з внутрішньою системою. Для моделювання таких атак тестувальникам надається IP адреса цільової системи, а інша інформація не надається, для перевірки таких застосунків, як, наприклад, система розпізнавання фото [19, 20].

1.3 Аналіз сучасних методів вебтестування

1.3.1 Особливості методів вебтестування за рівнями

До сучасних методів вебтестування за рівнями відносяться такі:

– модульне тестування (Unit Testing). Цей тип тестування проводиться розробниками до того, як збірка буде передана команді тестувальників для офіційного виконання тест кейсів. Модульне тестування проводиться відповідними розробникам. Розробники використовують тестові дані, окремі від тестових даних групи з забезпечення якості.

У рамках цього тестування проводиться аналіз модулів інтегрованих в систему (рис. 1.1).

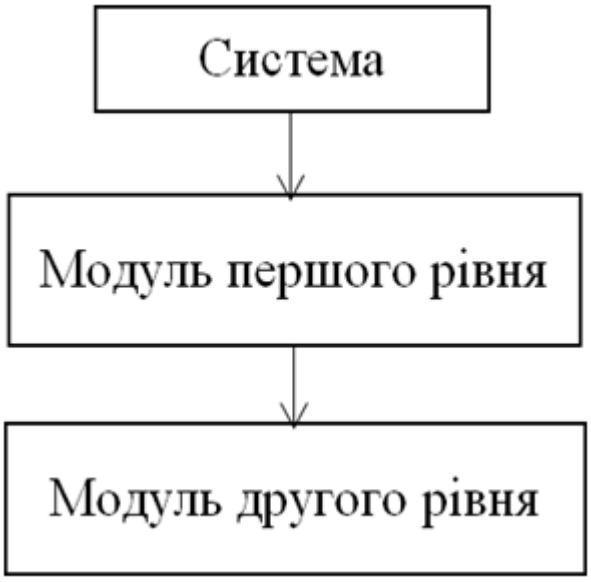

Рисунок 1.1 – Загальна схема інтеграції системних модулів

Мета модульного тестування – виділити кожну частину програми та показати, що окремі частини працюють правильно з точки зору вимог та функціональності.

Модульне тестування не може знайти кожну помилку програми. Неможливо оцінити кожен шлях виконання у кожному програмному застосуванні. Те саме стосується одиничного тестування. Існує обмеження на кількість сценаріїв і тестових даних, які розробник може використовувати для перевірки програмного коду. Тож після того, як він вичерпав усі варіанти, не залишається іншого вибору, як зупинити тестування модуля та об'єднати сегмент коду з іншими;

– інтеграційне тестування (Integration Testing). Тестування комбінованих частин програми, щоб визначити, чи правильно вони працюють разом, називається інтеграційним тестуванням. Також перевіряється взаємодії застосунку з зовнішніми системами. Існує два методи проведення тестування інтеграції: тестування знизу вгору та тестування інтеграції зверху вниз. Схема інтеграції модулів наведена на рисунку 1.1.

Тестування інтеграції знизу вгору починається з тестування окремих одиниць, після чого поступово випробовуються поєднання вищих рівнів одиниць, що називаються модулями або збірками.

У тестуванні інтеграції зверху вниз, модулі вищого рівня перевіряються спочатку, а модулі нижчого рівня тестуються після цього. У всеосяжному середовищі розроблення програмного забезпечення зазвичай спочатку проводять тестування знизу вгору, після чого – тестування зверху вниз;

– системне тестування (System Testing). Це наступний рівень тестування – тестування системи в цілому. Після інтеграції всіх компонентів застосунок у цілому ретельно перевіряється на предмет того, що він відповідає стандартам якості. Цей тип тестування проводиться спеціалізованою групою тестування.

Варіанти системного тестування з урахуванням усіх сценаріїв або сценаріїв з найвищим пріоритетом наведені на рисунку 1.2;

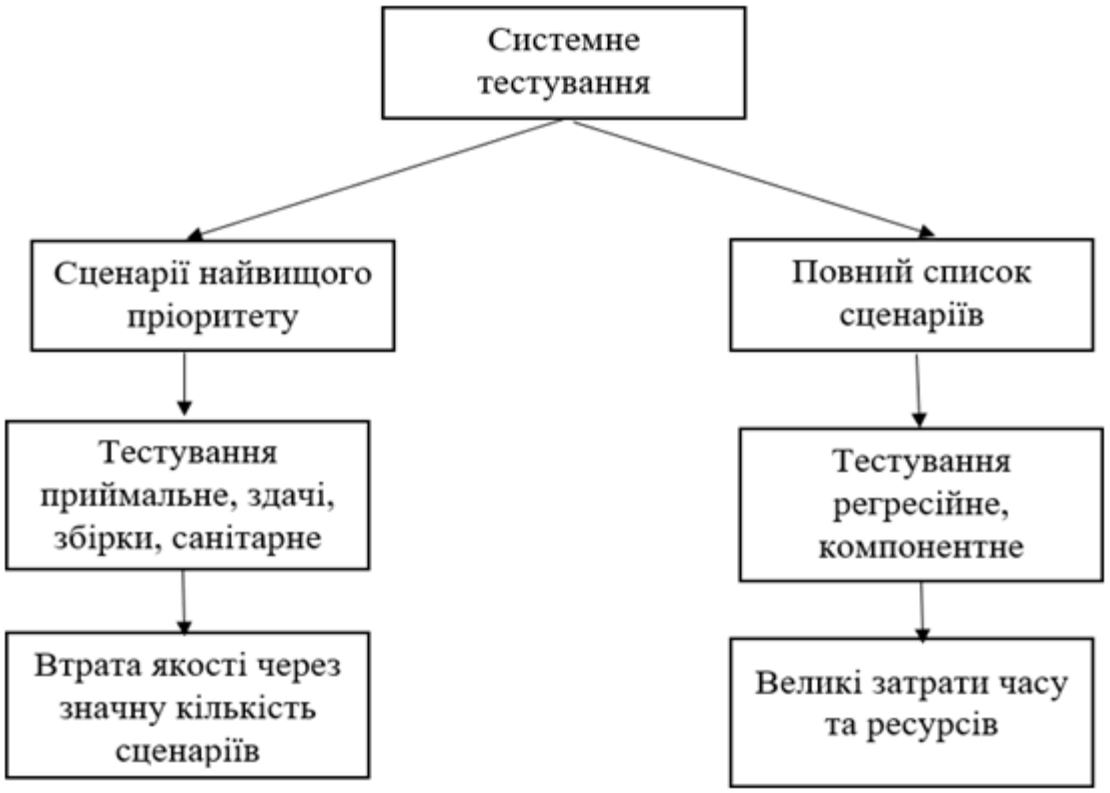

Рисунок 1.2 – Варіанти системного тестування

– регресійне тестування (Regression Testing). Щоразу, коли відбувається зміна в певній області коду програмного застосунка, це може вплинути й на інші області коду в програмі. Щоб перевірити, що виправлена помилка не призвела до порушення інших функціональних можливостей або правил, проводиться регресійне тестування. Його метою є забезпечити впевненість, що зміни, такі як виправлення помилок, не призвели до появи іншої помилки в застосунку. При регресійному тестуванні проводиться тестування зверху вниз по інтегрованим модулям (рис. 1.1);

– приймальне тестування (Acceptance Testing). Це найважливіший вид тестування, оскільки він проводиться командою із забезпечення якості, яка перевіряє, чи відповідає програма заявленим специфікаціям та чи відповідає вимогам клієнта. Команда QA матиме набір попередньо написаних сценаріїв

та тестових випадків, які будуть використані для тестування програми. Приймальне тестування призначене не лише для того, щоб вказати на прості орфографічні помилки, косметичні помилки або прогалини в інтерфейсі, але також вказати на будь-які помилки застосунку, які призведуть до збоїв у роботі системи або великих помилок у програмі. Виконуючи приймальні тести на системі, тестувальна група визначить, як буде працювати програма під час постійного використання. Існують також юридичні та договірні вимоги щодо прийняття системи в цілому. Схема приймального тестування наведена на рисунку 1.3;

Рисунок 1.3 – Схема приймального тестування ПЗ

– альфа-тестування. Цей тест є першим етапом тестування і проводитиметься серед команд (розробників та команд із забезпечення якості). Тестування модулів, тестування інтеграції та тестування системи у поєднанні відомі як альфа-тестування. Під час цієї фази в застосунку буде перевірено наступне: орфографічні помилки, несправні посилання, застосунок буде протестовано на машинах з найнижчою специфікацією для перевірки часу завантаження та будь-яких проблем із затримкою;

– бета-тестування. Цей тест виконується після успішного альфа-тестування. При бета-тестуванні частина ймовірних користувачів тестує застосунок. Бета-тестування також відоме як тестування перед випуском. Бета-тестові версії програмного забезпечення інколи розподіляються серед широкої аудиторії в Інтернеті, частково для того, щоб виконати тест в реальних умовах. Якщо розроблюване програмне забезпечення призначене для внутрішнього користування в певній компанії, то для тестування залучають співробітників замовника;

– нефункціональне тестування. Цей вид заснований на тестуванні програми на основі її нефункціональних атрибутів. Нефункціональне тестування програмного забезпечення передбачає тестування програмного застосунку на основі вимог, які не є функціональними за своєю суттю, але також важливі, такі як продуктивність, безпека, користувальницький інтерфейс тощо;

– тестування продуктивності. Тестування продуктивності здебільшого використовується для виявлення будь-яких вузьких місць або проблем з продуктивністю, а не для пошуку помилок у програмному забезпеченні. Існують різні причини, які сприяють зниженню продуктивності програмного забезпечення.

1.3.2 Розгляд співвідношення циклів розроблення та тестування вебсторінки

Цикл тестування вебсторінки складається з трьох етапів:

– вивчення та аналіз предмета тестування;

– планування тестування;

– виконання тестування.

На кожному з етапів може бути знайдений дефект, який повинен бути відлагоджений відповідальною персоною (наприклад, програмістом або

продюсером), і якість відновлення повинна бути сертифікована тестувальником.

Цикл розроблення ПЗ наведений на рисунку 1.4.

Узгодимо цикл тестування з циклом розроблення:

– вивчення та аналіз предмета тестування починаються перед затвердженням специфікації (на завершенні стадії «Проєктування системи»);

– планування тестування відбувається на стадії «Проєктування системи»;

– виконання тестування відбувається на стадії «Тестування системи на відповідність проєкту (верифікація)» . Наприклад, для системи штучного інтелекту верифікацією буде перевірка правильності виконання закладених функцій [21].

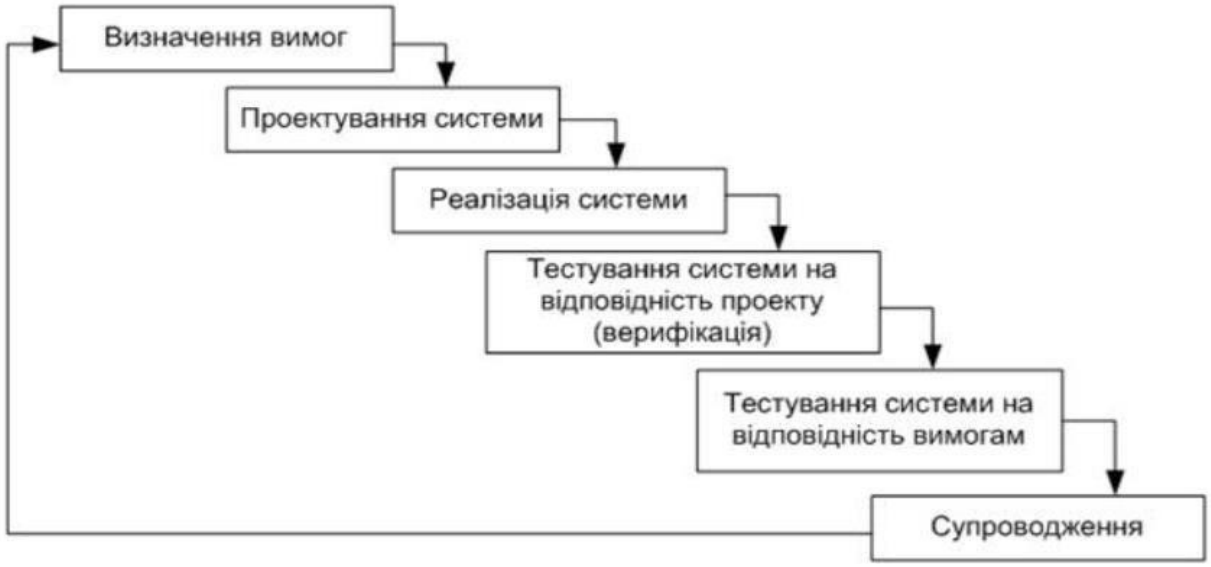

Рисунок 1.4 – Цикл розроблення ПЗ

Підсумуємо наведену вище інформацію, та визначимо список видів тестування за ознаками:

– за рівнем інтеграції: модульне, інтеграційне, системне;
– за типом: функціональне, нефункціональне;
– за видами аналізу: регресійне, приймальне, продуктивності;
– за рівнем готовності проєкту: альфа- та бета-тестування.

Загальна схема сучасних методів вебтестування подана на рисунку 1.5.

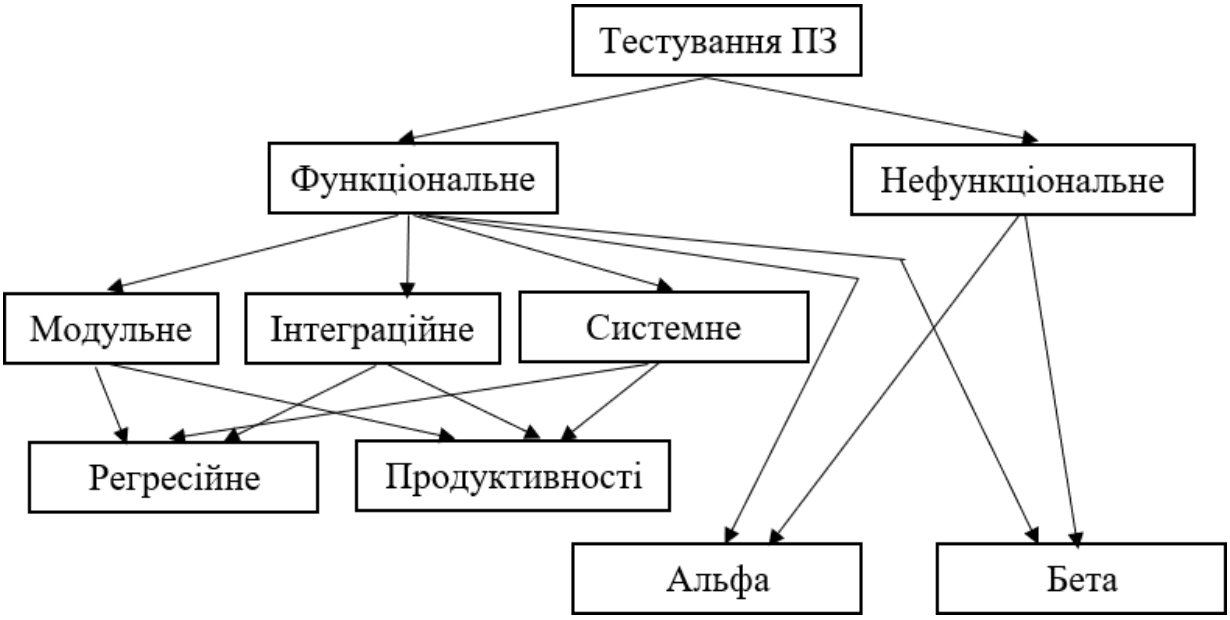

Рисунок 1.5 – Сучасні види вебтестування

1.4 Аналіз літературних джерел щодо апробації результатів застосування методів тестування вебзастосунків

У джерелі [22] досліджено необхідність проведення автоматизації процесу тестування вебзастосунків. Основною метою дослідження є вибір програмного продукту, яких допоможе підвищити ефективність та якість тестування вебплатформ, а також розроблення прототипів тест кейсів виконуваних програмою для аналізу отриманої користі від впровадження автоматизації. У процесі аналізу цього джерела можливо обрати програмний продукт для реалізації тест-кейсів, який буде використовуватися у практичних розділах даної роботи.

У джерелі [23] реалізовано систему для автоматичного тестування вебзастосунків. Завдяки цій системі можна буде легко підтримувати працездатність вебплатформ при внесенні змін у їх код. Система дозволяє зберігати, переглядати та покроково аналізувати результати тестувань.

В якості сховища даних використовується база даних PostgreSQL. Сама система написана на мові Java та TypeScript, з використанням фреймворків Spring та Angular. Доступ до системи можна отримати через браузер Google Chrome з будь якого пристрою. Це джерело може бути корисним як приклад реалізації системи автоматизованого вебтестування з метою підтримки працездатних вебсайтів.

У джерелі [24] запропоновано шаблони, які використовуються для генерації тестових сценаріїв і можуть швидко адаптуватися до будь-якого типового проєкту. Розроблено систему автоматичного тестування, що базується на сучасному фреймворку Selenium у поєднанні з мовою Python та бібліотекою тестування Python Unittest. Проведене дослідження показало практичну значимість нового методу генерації тестових сценаріїв завдяки скороченню витрат часу на складання тестів та проведення тестування. Шаблони з цього джерела можуть бути використані у практичних розділах даної роботи завдяки їх можливостям до швидкої адаптації до будь-якого проєкту.

У джерелі [25] проаналізовано основні підходи щодо проведення тестування на вразливості інформаційно-телекомунікаційних та комп’ютерних систем, зроблено вибір напряму тестування вебресурсів на проникнення. Згідно з методологією розроблено сценарії збору інформації, аналізу інформації та перевірки сайту. Визначено поширені помилки і уразливості. На основі статистичних даних основних небезпек вебсайту розроблено варіант методології тестування вебзастосунку. Зроблено висновок, що подальші розробки в цьому напрямі повинні вестися в площині класифікації, аналізу та управління окремими рівнями, розширення і врахування можливих кібератак на вебресурси. Дане джерело може бути використане як зразок, якщо у практичних розділах роботи будуть розглядатися питання з тестування систем на вразливість.

У джерелі [26] проаналізовані можливості для підвищення якості програмного забезпечення завдяки створенню системи, що поєднує у собі

кілька способів тестування програмного застосунку. Дана система повинна показувати більш високі показники надійності та достовірності у порівнянні з існуючими рішеннями. Дане джерело може бути корисним при аналізі прикладів мультикритеріального тестування у практичній частині даної роботи.

У джерелі [27] проведено аналіз стану проблеми тестування веб застосунків, аналітичний огляд літератури за темою атестаційної роботи, а ще проведено аналіз стану проблеми процесу находження оптимальної платформи тестування. Також було зроблено порівняльний аналіз існуючих програм тестування. Методом ієрархій була виявлена найбільш актуальна програма, на базі якої буде будуватися бібліотека. За результатами було сформовано критерії, які повинні бути присутні у бібліотеці тест-кейсів та розроблено рекомендацій щодо програм-бібліотек для збереження тест-кейсів. Проведено економічне обґрунтування науково дослідної роботи та розраховано економічну ефективність даного дослідження. Дане джерело може бути корисним як теоретичний посібник з проблем вебтестування.

У джерелі [28] наведено особливості тестування веб-додатків в контексті кожного етапу, а також розглянуто планування тестування, тест-дизайн, інтеграцію тестування в процес розробки, оцінку наявних коштів та трудовитрати. У даному джерелі доцільно розглянути теоретичну інформацію з тестування вебзастосунків.

У джерелі [29] проаналізовано існуючі методи для автоматизованого тестування вебзастосунків. Удосконалено метод роботи з елементами користувацького інтерфейсу, які з’являються у результаті асинхронноїклієнт-серверної взаємодії. Розроблено алгоритм та програмний засіб для його реалізації. Методи з даного джерела можуть бути використані при реалізації практичних частини даної роботи.

У джерелі [30] приділено увагу аналізу сучасних ефективних інструментів для автоматизованого тестування вебзастосунків. Інструменти з

цього джерела можуть бути використані при реалізації практичної частини даної роботи.

У джерелі [31] проведено огляд сучасних видів та методів тестування застосунків. Розглянуто засіб для тестування за допомогою емуляторів та реальних пристроїв. Наведено практичне його застосування та розглянуті переваги та недоліки його застосування. В практичній частині роботи протестовано вебзастосунок. Тестування основних функцій проведено з комп'ютерного браузера та мобільного, тому що сайт має адаптивний дизайн до різних розмірів екранів. Види та методи з цього джерела можуть бути використані у практичній частині даної роботи.

У джерелі [32] проведена апробація методики тестування безпеки вебзастосунків на основі інформації з відкритих джерел з урахуванням технологічного процесу їх розроблення. Дане джерело використане при аналізі апробації методологій вебтестування.

1.5 Постановка задачі дослідження

Актуальність даної роботи полягає у важливості проведення вебтестування з метою підвищення якості вебзастосунків та їх відповідності до очікувань користувачів.

Об'єктом роботи є процес тестування вебзастосунків.

Метою роботи є дослідження та порівняння вибраних методів вебтестування задля визначення їх ефективності під час тестування окремих класів задач.

Враховуючи мету роботи, необхідно вирішити такі завдання:

– здійснити класифікацію методів вебтестування;

– проаналізувати сучасні тенденції реалізації проєктів з вебтестування;

– змоделювати процес вебтестування за допомогою стандарту IDEF0;

– виявити особливості вибраних методів вебтестування;

– здійснити вибір інструментальних засобів для реалізації вибраних методів;

– програмно реалізувати вибрані методи вебтестування;

– застосувати методи вебтестування щодо вибраної предметної області;

– здійснити порівняльний аналіз досліджених методів вебтестування;

– визначити перспективи подальшої роботи.

2 ДОСЛІДЖЕННЯ ВИБРАНИХ МЕТОДІВ ВЕБТЕСТУВАННЯ

2.1 Дослідження найпоширеніших видів вебтестування

Тестування чорної скриньки або поведінкове тестування – стратегія (метод) тестування функціонального поведінки об’єкта (програми, системи) з точки зору зовнішнього світу, при якому не використовується знання про внутрішній устрій тестованого об’єкта. Під стратегією розуміються систематичні методи відбору і створення тестів для тестового набору. Стратегія поведінкового тесту виходить з технічних вимог та специфікацій.

В даний час відомі два види «чорних» ящиків. До першого виду відносять будь-який «чорний» ящик, який може розглядатися як автомат, званий кінцевим або нескінченним. Поведінка таких «чорних» ящиків відомо. До другого виду відносяться такі «чорні» ящики, поведінка яких може бути виявленою тільки в експерименті.

У такому випадку в явній чи неявній формі висловлюється гіпотеза про передбачуваність поведінки «чорного» ящика в імовірнісному сенсі. Без попередньої гіпотези неможливо будь-яке узагальнення, або, як кажуть, неможливо зробити індуктивний висновок на основі експериментів з «чорним» ящиком. Для позначення моделі «чорного» ящика було запропоновано поняття «білого» ящика. «Білий» ящик складається з відомих компонентів, тобто відомих X, Y, δ, λ. Його вміст спеціально підбирається для реалізації тієї ж залежності виходу від входу, що і у відповідного «чорного» ящика. У процесі проведених досліджень і при узагальненнях, висуванні гіпотез і встановлення закономірностей виникає необхідність коректування організації «білого» ящика і зміни моделей. У зв’язку з цим, при моделюванні дослідник повинен обов’язково звертатися до схеми співвідношення «чорний» – «білий» ящик [20]. Існує кілька ознак, за якими прийнято робити класифікацію видів тестування (табл. 2.1).

Таблиця 2.1 – Ознаки класифікації тестування

Вид	**Опис**
За об’єктом тестування	функціональне тестування
	тестування продуктивності
	навантажувальне тестування
	стрес-тестування
	тестування стабільності
	тестування зручності використання
	тестування інтерфейсу користувача
	тестування безпеки
	тестування локалізації
	тестування сумісності
За знанням системи	тестування чорного ящика
	тестування білого ящика
	тестування сірого ящика
За ступенем автоматизації	ручне тестування
	автоматизоване тестування
	напівавтоматизоване тестування
За ступенем ізольованості компонентів	компонентне (модульне) тестування
	інтеграційне тестування
	системне тестування
За часом проведення тестування	альфа-тестування
	тестування при прийманні
	тестування нової функціональності
	регресійне тестування
	тестування при здачі
За ознакою позитивності сценаріїв	позитивне тестування
	негативне тестування
За ступенем підготовленості до тестування	тестування за документацією
	інтуїтивне тестування

Функціональні тести базуються на функціях та особливостях, а також на взаємодії з іншими системами, і можуть бути представлені на всіх рівнях тестування: компонентному або модульному, інтеграційному, системному і приймальному. Функціональні види тестування розглядають зовнішню поведінку системи.

Нефункціональне тестування описує тести, необхідні для визначення характеристик програмного забезпечення, які можуть бути виміряні різними величинами. У цілому, це тестування того, як система працює.

Одні з найпоширеніших видів функціональних тестів:

– тестування безпеки (Security and Access Control Testing) – стратегія тестування, що використовується для перевірки безпеки системи, а також для аналізу ризиків, пов’язаних із забезпеченням цілісного підходу до захисту програми, атак хакерів, вірусів, несанкціонованого доступу до конфіденційних даних. Тестування безпеки може виконуватися як автоматизовано так і в ручну, включаючи перевірку як позитивних, так і негативних тестових випадків;

– тестування взаємодії (Interoperability Testing) – це функціональне тестування, що перевіряє здатність програми взаємодіяти з одним і більше компонентами або системами і включає в себе тестування сумісності та інтеграційне тестування.

Основні види нефункціональних тестів:

– тестування навантаження (Performance and Load Testing) – визначення масштабованості застосунків під навантаженням, при цьому відбувається: вимір часу виконання вибраних операцій за певних інтенсивностей виконання цих операцій; визначення кількості користувачів, що одночасно працюють з застосунком; визначення меж прийнятної продуктивності при збільшенні навантаження (при збільшенні інтенсивності виконання цих операцій); дослідження продуктивності при високих, граничних, стресових навантаженнях;

– стресове тестування (Stress Testing) дозволяє перевірити наскільки застосунок і система в цілому працездатні в умовах стресу і також оцінити здатність системи до регенерації, тобто до повернення до нормального стану після припинення впливу стресу. Стресом у даному контексті може бути підвищення інтенсивності виконання операцій до дуже високих значень або аварійна зміна конфігурації сервера. Також одним із завдань при стресовому тестуванні може бути оцінка деградації продуктивності, таким чином цілі стресового тестування можуть перетинатися з цілями тестування продуктивності;

– тестування стабільності або надійності (Stability / Reliability Testing) – перевірка працездатності програми при тривалому тестуванні з середнім рівнем навантаження. Час виконання операцій може грати в даному виді тестування другорядну роль. При цьому на перше місце виходить відсутність витоків пам’яті, перезапусків серверів під навантаженням й інші аспекти, які впливають саме на стабільність роботи;

– об’ємне тестування (Volume Testing) – отримання оцінки продуктивності при збільшенні обсягів даних у базі даних програми, при цьому відбувається: вимір часу виконання вибраних операцій за певних інтенсивностей виконання цих операцій; може проводитися визначення кількості користувачів, що одночасно працюють з застосунком;

– тестування установки (Installation testing) спрямоване на перевірку успішної інсталяції та настройки, а також оновлення або видалення програмного забезпечення. На даний момент найбільш поширена установка ПЗ за допомогою інсталяторів (спеціальних програм, які самі по собі так само потребують належного тестування). У реальних умовах інсталяторів може не бути. У цьому випадку доведеться самостійно виконувати установку програмного забезпечення, використовуючи документацію у вигляді інструкцій або readme-файлів, де крок за кроком описано всі необхідні дії та перевірки;

– тестування зручності користування (Usability Testing) – це метод тестування, спрямований на встановлення ступеня зручності використання, навченості, зрозумілості та привабливості для користувачів розроблюваного продукту в контексті заданих умов.

Тестування зручності користування дає оцінку рівня зручності використання програми за наступними пунктами:

– продуктивність, ефективність (efficiency) – скільки часу і кроків знадобиться користувачеві для завершення основних завдань програми, наприклад, розміщення новини, реєстрації, покупки тощо (менше – краще);

– правильність (accuracy) – скільки помилок зробив користувач під час роботи с застосунком (менше – краще);

– активізація в пам’яті (recall) – як дефакто користувач пам’ятає про роботу програми після припинення роботи з нею на тривалий період часу (повторне виконання операцій після перерви має проходити швидше ніж у нового користувача);

– емоційна реакція (emotional response) – як користувач почувається після завершення завдання – розгублений, знаходиться у стані стресу? Чи порекомендує користувач систему своїм друзям (позитивна реакція – краще)?

Тестування на відмову і відновлення (Failover and Recovery Testing) перевіряє тестований продукт з точки зору здатності протистояти й успішно відновлюватися після можливих збоїв, що виникли у зв’язку з помилками програмного забезпечення, відмовами обладнання або проблемами зв’язку (наприклад, відмова мережі). Метою даного виду тестування є перевірка систем відновлення (або дублюючих основний функціонал систем), які, у разі виникнення збоїв, забезпечать збереження і цілісність даних тестованого продукту.

Конфігураційне тестування (Configuration Testing) – ще один вид традиційного тестування продуктивності. У цьому випадку замість того, щоб тестувати продуктивність системи з точки зору навантаження, тестується ефект впливу на продуктивність змін у конфігурації. Прикладом такого

тестування можуть бути експерименти з різними методами балансування навантаження. Конфігураційне тестування також може бути поєднане з навантажувальним, стрес- або тестуванням стабільності.

Після проведення необхідних змін, таких як виправлення дефекту, програмне забезпечення повинне бути повторно протестовано для підтвердження того факту, що проблему було дійсно вирішено. Нижче перераховані види тестування, які необхідно проводити після установки програмного забезпечення, для підтвердження працездатності програми або правильності здійсненого виправлення дефекту:

– димове тестування (Smoke Testing). Поняття димове тестування пішло з інженерного середовища. При введенні в експлуатацію нового ПЗ вважалося, що тестування пройшло вдало, якщо з установки не пішов дим. В області тестування програмного забезпечення, воно спрямоване на поверхневу перевірку всіх модулів програми на предмет працездатності та наявність швидко встановлюваних критичних і блокуючих дефектів. За результатами димового тестування робиться висновок про те, приймається чи ні встановлена версія програмного забезпечення на тестування, експлуатацію або на постачання замовнику;

– регресійне тестування (Regression Testing) – вид тестування спрямований на перевірку змін, зроблених у застосунку або навколишньому середовищі (лагодження дефекту, злиття коду, міграція на іншу операційну систему, базу даних, вебсервер або сервер додатка), для підтвердження того факту, що існуюча раніше функціональність працює як і раніше;

– санітарне тестування або перевірка узгодженості / справності (Sanity Testing) – вузьконаправлене тестування достатнє для доказу того, що конкретна функція працює згідно заявлених у специфікації вимог;

– модульне тестування (юніт-тестування) – тестується мінімально можливий для тестування компонент, наприклад, окремий клас або функція. Часто модульне тестування здійснюється розробниками ПЗ;

– тестування збірки (Build Verification Test) спрямоване на визначення відповідності випущеної версії критеріям якості для початку тестування. За своєю метою є аналогом димового тестування, спрямованого на приймання нової версії в подальше тестування або експлуатацію. Вглиб воно може проникати далі, в залежності від вимог до якості випущеної версії;

– інтеграційне тестування – тестуються інтерфейси між компонентами, підсистемами. За наявності резерву часу на даній стадії тестування ведеться ітераційно, з поступовим підключенням наступних підсистем.

Загальна схема інтеграції програмних модулів була показана на рисунку 1.1.

Рівні інтеграційного тестування:

– компонентний інтеграційний рівень (Component Integration Testing). Перевіряється взаємодія між компонентами системи після проведення компонентного тестування;

– системний інтеграційний рівень (System Integration Testing). Перевіряється взаємодія між різними системами після проведення системного тестування.

Підходи до інтеграційного тестування:

– знизу вгору (Bottom Up Integration). Усі низькорівневі модулі, процедури або функції збираються воєдино і потім тестуються. Після чого збирається наступний рівень модулів для проведення інтеграційного тестування. Даний підхід вважається корисним, якщо всі або практично всі модулі розроблюваного рівня готові. Також даний підхід допомагає визначити за результатами тестування рівень готовності застосунків;

– зверху вниз (Top Down Integration). У першу чергу тестуються компоненти верхнього рівня ієрархії об’єктів з використанням заглушок замість компонентів більш низького рівня;

– великий вибух («Big Bang» Integration). Усі або практично усі розроблені модулі збираються разом у вигляді закінченої системи або її основної частини, і потім проводиться інтеграційне тестування. Такий підхід

дуже хороший для збереження часу. Проте, якщо тест кейси та їх результати записані не вірно, то сам процес інтеграції дуже ускладниться, що стане перепоною для команди тестування при досягненні основної мети інтеграційного тестування.

Системне тестування передбачає тестування інтегрованої системи на її відповідність вимогам та має наступні види:

– альфа-тестування – імітація реальної роботи з системою штатними розробниками, або реальна робота з системою потенційними користувачами / замовником. Найчастіше альфа-тестування проводиться на ранній стадії розроблення продукту, але у деяких випадках може застосовуватися для закінченого продукту в якості внутрішнього приймального тестування. Іноді альфа-тестування виконується під відлагоджувачем або з використанням оточення, яке допомагає швидко виявляти знайдені помилки. Виявлені помилки можуть бути передані тестувальникам для додаткового дослідження в оточенні, подібному тому, в якому буде використовуватися програма;

– бета-тестування – у деяких випадках виконується поширення версії з обмеженнями (за функціональністю або часом роботи) для певної групи осіб, з тим щоб переконатися, що продукт містить достатньо мало помилок. Іноді бета-тестування виконується для того, щоб отримати зворотній зв’язок про продукт від його майбутніх користувачів.

Часто для вільного/відкритого ПЗ стадія альфа-тестування характеризує функціональне наповнення коду, а бета-тестування – стадію виправлення помилок. При цьому, як правило, на кожному етапі розроблення проміжні результати роботи доступні кінцевим користувачам.

Тестування «білого ящика» і «чорного ящика».

Схематичний аналіз тестування методом «білого ящика» наведений на рисунку 2.1 [17].

Рисунок 2.1 – Схема тестування методом «білого ящика»

Тестування чорного ящика в даній ситуації буде відрізнятися відсутністю вихідного коду ПЗ як вхідного компоненту.

У термінології професіоналів-тестувальників фрази «тестування білого ящика» і «тестування чорного ящика» відносяться до того, чи має розробник тестів доступ до вихідного коду ПЗ, що тестується, тестування виконується через інтерфейс користувача чи прикладний програмний інтерфейс, наданий модулем, що тестується.

Під час тестування білого ящика (White-Box Testing) розробник тесту має доступ до вихідного коду програм і може писати код, який пов’язаний з бібліотеками ПЗ, що тестується. Це типово для юніт-тестування (Unit-Testing), при якому тестуються тільки окремі частини системи. Воно забезпечує те, що компоненти конструкції працездатні й стійкі до певного ступеня. Під час тестування білого ящика використовуються метрики покриття коду.

При тестуванні чорного ящика тестер має доступ до ПЗ тільки через ті ж інтерфейси, що й замовник або користувач, або через зовнішні інтерфейси, що дозволяють іншому комп’ютеру або іншому процесу підключитися до системи для тестування.

Наприклад, модуль, що тестується, може віртуально натискати клавіші або кнопки миші за допомогою механізму взаємодії процесів, із упевненістю в тому, чи все йде правильно, що ці події викликають той же відгук, що й реальні натискання клавіш і кнопок миші. Як правило, тестування чорного ящика ведеться з використанням специфікацій чи інших документів, що описують вимоги до системи. В даному виді тестування критерій покриття складається з покриття структури вхідних даних, покриття вимог і покриття моделі (у тестуванні на основі моделей).

При тестуванні сірого ящика розробник тесту має доступ до вихідного коду, але при безпосередньому виконанні тестів доступ до коду, як правило, не потрібний.

Можливості статичного тестування ПЗ наведені на рисунку 2.2.

Рисунок 2.2 – Можливості статичного тестування ПЗ

Функції динамічного тестування ПЗ наведені на рисунку 2.3.

Рисунок 2.3 – Функції динамічного тестування ПЗ

При статичному тестуванні програмний код не виконується – аналіз програми відбувається на основі вихідного коду, який вираховується вручну, або аналізується спеціальними інструментами. У деяких випадках, аналізується не вихідний, а проміжний код (такий як байт-код або код на MSIL).

До статичного тестування також відносять тестування вимог, специфікацій, документації.

Тестувальники використовують тестові скрипти на різних рівнях: як у модульному, так і в інтеграційному і системному тестуванні. Тестові скрипти, як правило, пишуться для перевірки компонентів, в яких найбільш висока ймовірність появи відмов або вчасно не знайдена помилка може бути високовартісною.

Покриття коду, за своєю суттю, є тестуванням методом білого ящика. Протестоване ПЗ збирається зі спеціальними настройками або бібліотеками та/або запускається в особливому оточенні, в результаті чого для кожної використовуваної (виконуваної) функції програми визначається місцезнаходження цієї функції у вихідному коді. Цей процес дозволяє розробникам і фахівцям із забезпечення якості визначити частини системи, які, при нормальній роботі, використовуються дуже рідко або ніколи не використовуються (такі як код обробки помилок тощо). Це дозволяє зорієнтувати тестувальників на тестування найбільш важливих режимів.

Як правило, інструменти й бібліотеки, які використовуються для отримання покриття коду, вимагають значних витрат продуктивності та/або пам’яті, неприпустимих при нормальному функціонуванні ПЗ. Тому вони можуть використовуватися тільки в лабораторних умовах.

Приймальне тестування (Acceptance Testing) – формальний процес тестування, який перевіряє відповідність системи вимогам і проводиться з метою: визначення чи задовольняє система приймальним критеріям; винесення рішення замовником або іншою уповноваженою особою приймається застосунок чи ні.

Схема приймального тестування ПЗ наведена на рисунку 1.3.

Приймальне тестування виконується на підставі набору типових тестових випадків і сценаріїв, розроблених на підставі вимог до даного застосунку.

Рішення про проведення приймального тестування приймається тоді, коли: продукт досяг необхідного рівня якості; замовник ознайомлений з планом приймальних робіт (Product Acceptance Plan) або іншим документом, де описаний набір дій, пов'язаних з проведенням приймального тестування, дата проведення, відповідальні тощо.

Фаза приймального тестування триває до тих пір, поки замовник не виносить рішення про відправлення програми на доопрацювання або видачі застосунка.

2.2 Моделювання процесу вебтестування за допомогою стандарту IDEF0

У будь-якій галузі виникає питання щодо ефективності реалізації будь-яких процесів. Аналіз показників ефективності допомагає виявити загальну картину діяльності, але для більш широкого дослідження доцільно використовувати моделювання процесів.

Найбільш широко використовується така методологія моделювання бізнес-процесів (Business Process Modeling): стандарт IDEF0. Моделі в нотації IDEF0 призначені для високорівневого опису реалізації процесів в функціональному аспекті.

IDEF – методологія моделювання і стандарт документування процесів, що відбуваються в системі. Метод документування технологічних процесів є механізмом документування та збору інформації про процеси. IDEF показує причинно-наслідкові зв'язки між ситуаціями і подіями в зрозумілій експерту

формі, використовуючи структурний метод вираження знань про те, як функціонує система, процес або підприємство [33].

Цілі моделювання бізнес-процесів, зазвичай, формулюються таким чином:

– забезпечити розуміння структури організації та динаміки, що відбувається в ній;

– забезпечити розуміння поточних проблем організації та можливостей їх вирішення;

– переконатися, що замовники, користувачі та розробники однаково розуміють цілі та завдання організації;

– створити базу для формування вимог до ПЗ, автоматизувати бізнес-процеси організації (вимоги до ПО формуються на основі бізнес-моделі).

Важливим елементом моделі бізнес-процесів є бізнес-правила або правила предметної області. Типовими бізнес-правилами є корпоративна політика і державні закони. Бізнес-правила, зазвичай, формулюються в спеціальному документі і можуть відбиватися в моделях. Декомпозиція в загальному сенсі – це метод, що дозволяє замінити рішення однієї великої задачі рішенням серії менших завдань, розщеплення об’єкта на складові частини за встановленим критерієм. Практично декомпозиція застосовується для деталізації бізнес-моделей [33].

Використання моделювання у процесі аналізу вебтестування може допомогти тестувальникам або проєктним менеджерам чи замовникам краще розуміти загальну структуру проєкту та проаналізувати можливості для розвитку чи реорганізації процесу тестування або навіть знайти кейси можливостей для покращення вебзастосунку з точки зору розроблення.

Стандарт IDEF0 допоможе найкращим чином проаналізувати картину взаємодії вебзастосунку та процесу вебтестування.

Перше, що створюється у процесі моделювання у стандарті IDEF0, – контекстна діаграма.

Контекстна діаграма – це діаграма самого верхнього рівня, що представляє систему загалом, у вигляді «чорного ящика», і зв’язує її із зовнішнім світом за допомогою інтерфейсних дуг. Контекстна діаграма складається з одного функціонального блоку, будь-якої кількості стрілок, мети моделювання та певної точки зору [34].

Контекстна діаграма процесу вебтестування наведена на рисунку 2.4.

На цьому етапі спеціалісти-тестувальники зі свого боку, розробники ПЗ та замовники процесу тестування зі свого визначають загальну картину того, як та у які строки буде проводитися процес тестування, які методи будуть використані при тестуванні і т.д.

Діаграма першого рівня декомпозиції A0, а також всі наступні діаграми декомпозиції, надають інтерфейсні обмеження (контекст) для дочірніх діаграм.

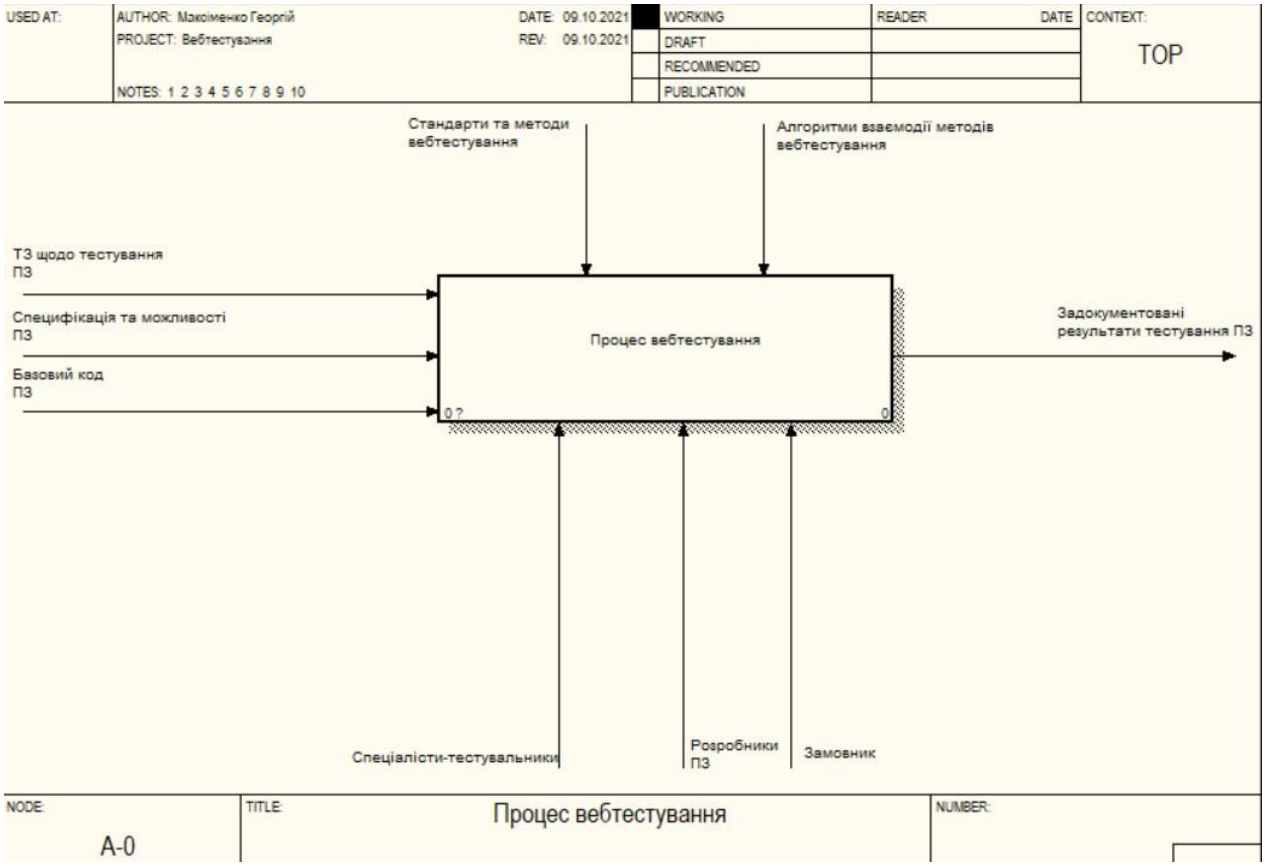

Рисунок 2.4 – Контекстна діаграма процесу вебтестування

Процес подальшого проєктування є більш формалізованим і необхідним ступенем деталізації, що досягається виконанням наступного рекурсивного процесу:

– вибір блоку діаграми. Декомпозицію рекомендується починати з самого змістовного блоку з точки зору домінування, функціональної складності і впливу на декомпозицію інших блоків цієї діаграми;

– розгляд об’єкта, визначеного цим блоком;

– створення нової діаграми;

– виявлення недоліків нової діаграми;

– створення альтернативних декомпозицій;

– коригування нової діаграми;

– коригування всіх пов’язаних з нею діаграм та визначення логіки їх взаємодії [35].

Діаграма декомпозиції першого рівня наведена на рисунку 2.5.

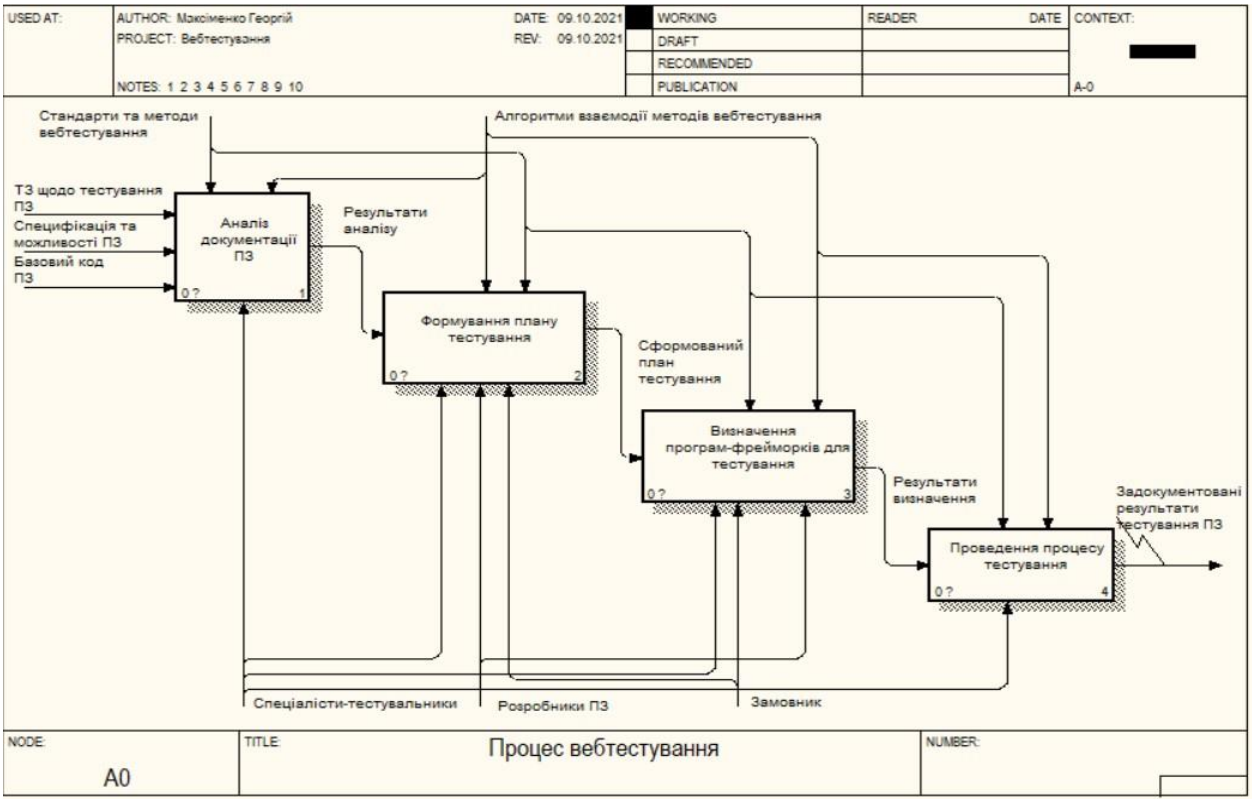

Рисунок 2.5 – Діаграма декомпозиції першого рівня процесу вебтестування

На цьому етапі відбувається більш конкретний огляд майбутнього процесу вебтестування, формується план тестування, та набір застосунків, які будуть використані, як інструменти тестування.

На етапі аналізу документації щодо ПЗ спеціалісти-тестувальники отримують загальне уявлення про ПЗ, його очікувану поведінку та можливі слабкі місця, очікування замовника щодо термінів, плану та тестування.

Діаграма декомпозиції процесу аналізу документації щодо ПЗ наведена на рисунку 2.6.

Результатом цього етапу є сформоване теоретичне та практичне розуміння спеціалістами тестувальниками того, що можливо очікувати від ПЗ, якими є побажання замовників та який первинний план тестування може бути сформований.

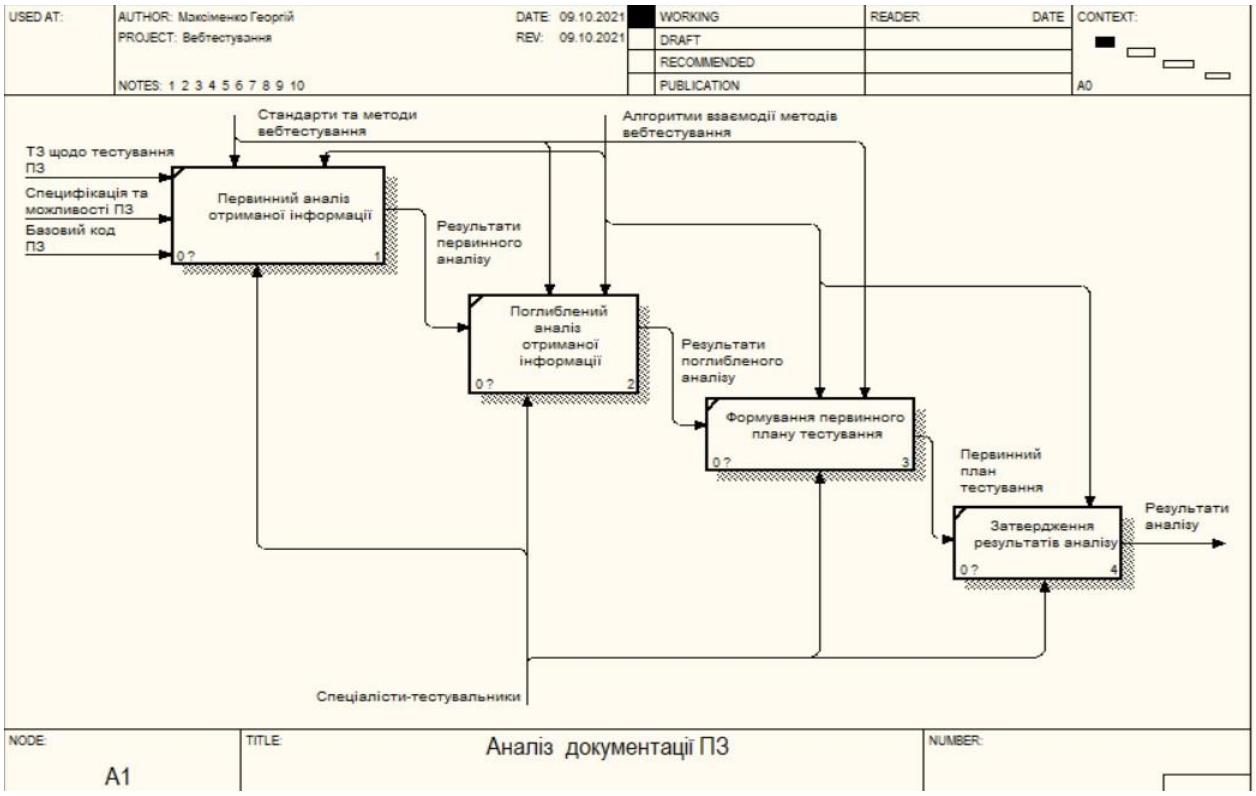

Рисунок 2.6 – Діаграма декомпозиції другого рівня процесу аналізу документації ПЗ

Наступним етапом є формування плану тестування. Одним з елементів вхідної інформації для цього етапу є первинний план тестування, який має зазнати покращень у процесі виконання етапу формування остаточно плану.

Діаграма декомпозиції процесу формування плану тестування ПЗ наведена на рисунку 2.7.

Результатом цього етапу є сформований план тестування ПЗ з точки зору термінів тестування та побажань замовника.

Наступним етапом є визначення інструментів тестування (рис. 2.8).

Результатом цього етап є вибрані інструменти вебтестування відповідно до затвердженого раніше плану тестування.

Останнім етапом є проведення тестування, аналіз та документування його результатів (рис. 2.9).

За результатами фінального етапу замовники отримують задокументовані результати тестування вебзастосунку.

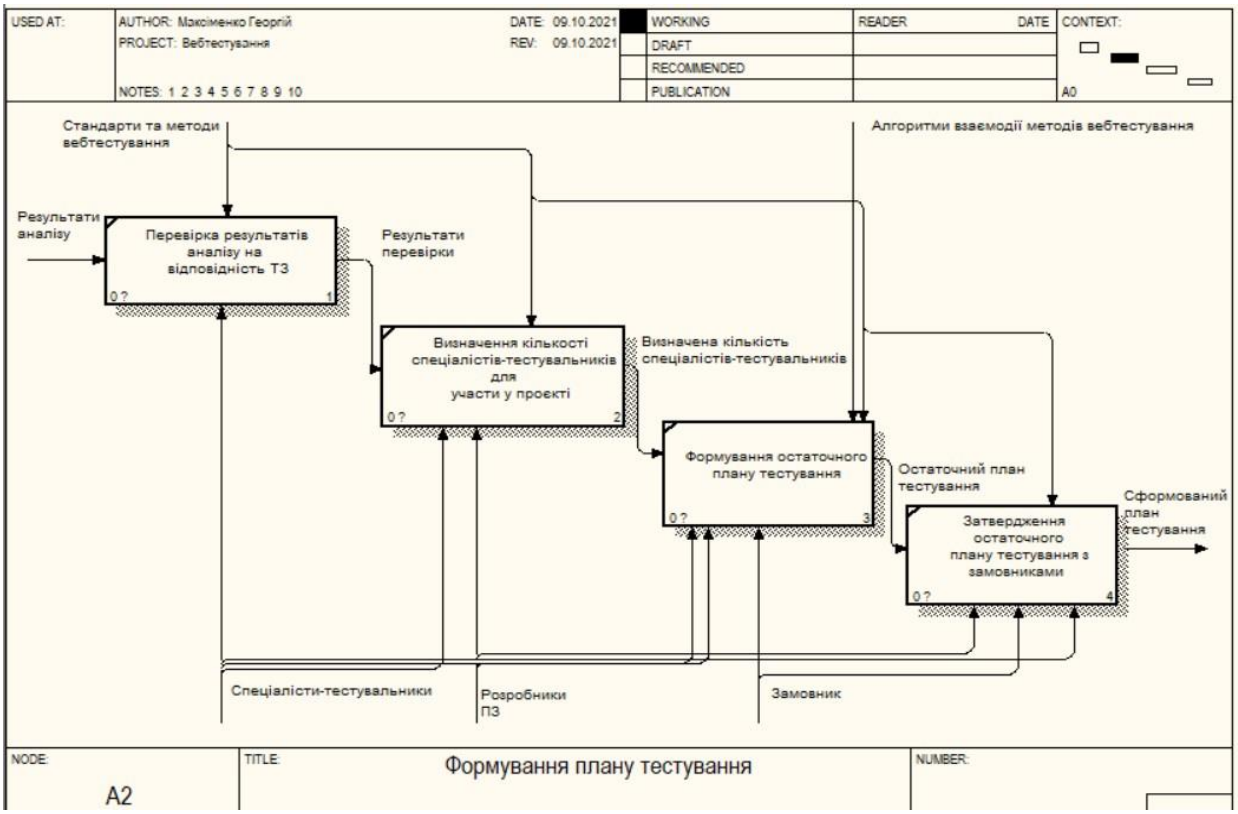

Рисунок 2.7 – Діаграма декомпозиції другого рівня процесу формування плану тестування

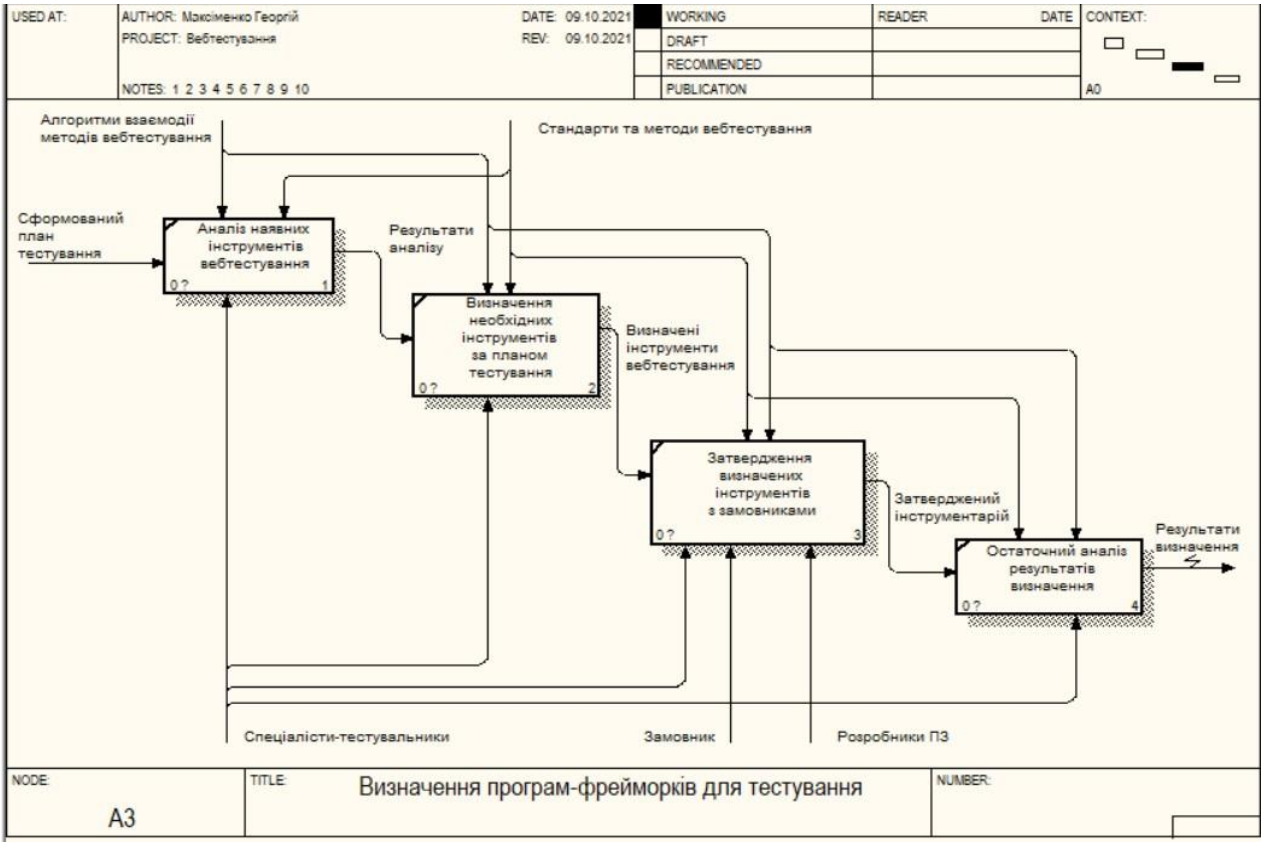

Рисунок 2.8 – Діаграма декомпозиції другого рівня процесу визначення програм-фреймворків для тестування

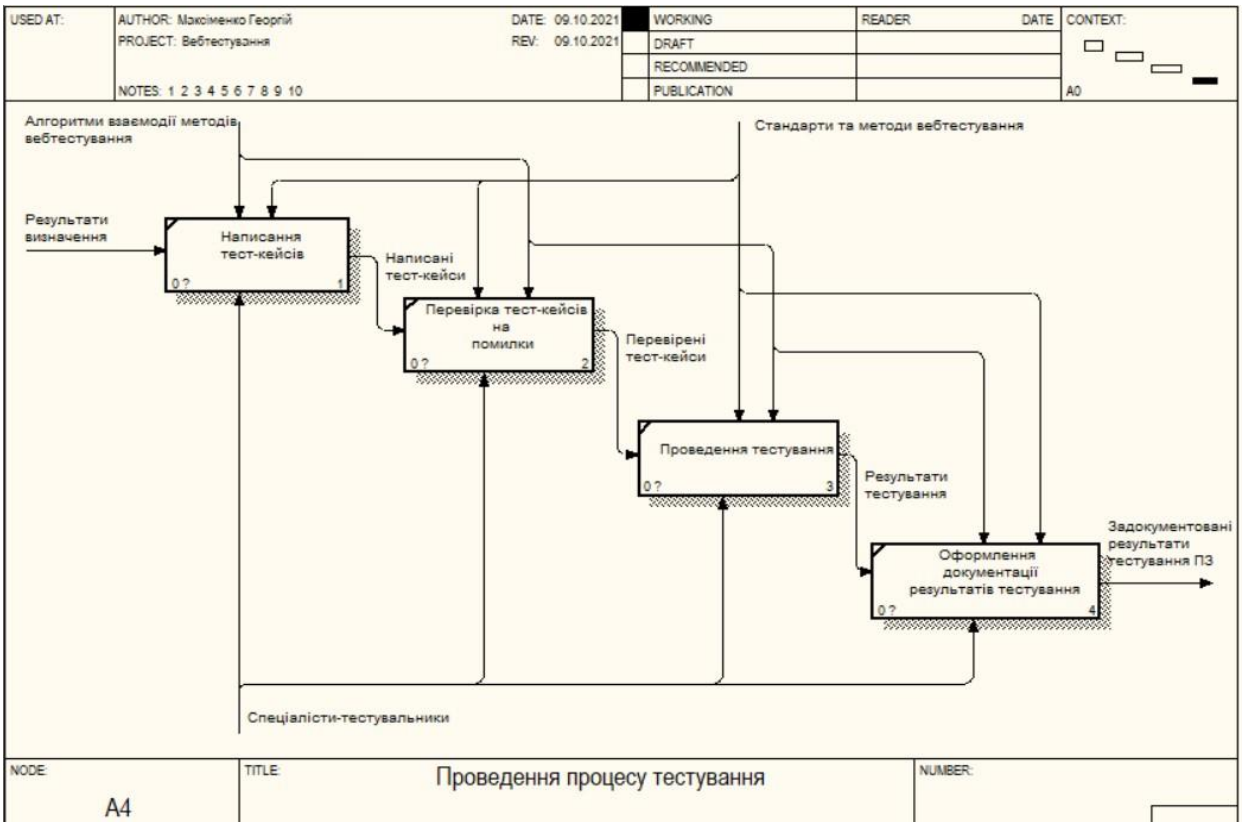

Рисунок 2.9 – Діаграма декомпозиції другого рівня процесу проведення тестування

За результатами тестування вебзастосунок може бути змінений (з метою покращення) або є одразу відкритим для користування.

2.3 Дослідження методу тестування «Регресійне тестування»

Регресійне тестування (Regression Testing) – це вид тестування, що спрямований на перевірку змін, зроблених в застосунку або програмному середовищі (виправлення дефекту, злиття коду, міграція на іншу операційну систему, базу даних, вебсервер або серверні застосунки), для підтвердження того факту, що існуюча раніше функціональність працює як і раніше. Регресійними можуть бути як функціональні, так і нефункціональні тести. Як правило, для регресійного тестування використовуються тест-кейси, написані на ранніх стадіях розроблення і тестування. Це дає гарантію того, що зміни в новій версії програми не пошкодили вже існуючу функціональність.

Рекомендується робити автоматизацію регресійних тестів, для прискорення подальшого процесу тестування і виявлення дефектів на ранніх стадіях розроблення програмного забезпечення [36].

Головним завданням етапу регресійного тестування є реалізація систематичного процесу оброблення змін у коді. Після кожної модифікації програми необхідно упевнитися, що на функціональність програми не зробив впливу модифікований код. Якщо такий вплив виявлено, говорять про регресійний дефект. Під час регресійного тестування функціональні можливості, зміна яких не планувалася, використовують раніше розроблені тести. Одна з цілей регресійного тестування полягає в тому, щоб, відповідно до використаного критерію покриття коду (наприклад, критерію покриття потоку операторів або потоку даних), гарантувати той же рівень покриття, що і при повному повторному тестуванні програми. Для цього необхідно запускати тести, що відносяться до змінених областей коду або функціональних можливостей, та реалізовувати перевірку методів інтелектуального аналізу [37].

Інша мета регресійного тестування полягає в тому, щоб упевнитися, що програма функціонує у відповідності зі своєю специфікацією, і що зміни не призвели до внесення нових помилок у раніше протестований код. Ця мета завжди може бути досягнута повторним виконанням всіх тестів регресійного набору, але більш перспективно відсівати тести, на яких вихідні дані модифікованої і старої програми не відрізняються [37].

Оскільки регресійне тестування є повторним проведенням циклу звичайного тестування, види регресійного тестування збігаються з видами звичайного тестування. Можна говорити, наприклад, про модульне регресійне тестування або про функціональне регресійне тестування. Інший спосіб класифікації видів регресійного тестування пов’язує їх з типами супроводу, які, в свою чергу, визначаються типами модифікацій. Виділяють три типи супроводу:

– коригуючий супровід, званий, зазвичай, виправленням помилок, виконується у відповідь на виявлення помилки, що не вимагає зміни специфікації вимог. При коригуючому супроводі проводиться діагностика і коректування дефектів у програмному забезпеченні з метою підтримки системи в працездатному стані;

– адаптивний супровід здійснюється у відповідь на вимоги зміни даних або середовища виконання. Воно застосовується, коли існуюча система поліпшується або розширюється, а специфікація вимог змінюється з метою реалізації нових функцій;

– вдосконалюючий (прогресивний) супровід включає будь-яке оброблення з метою підвищення ефективності роботи системи або ефективності її супроводу [37].

Процес реалізації регресійного тестування у вигляді блок-схеми наведений на рисунку 2.10.

Рисунок 2.10 – Блок-схема процесу реалізації регресійного тестування

Розглянемо приклад регресійного тестування, коли до вебсторінки було додано дві нових кнопки з назвами (buttonName), як елементи кнопок, за допомогою мови програмування JS та фреймворку Cypress (застосунок, який відповідає за реалізацію тест-кейсів) необхідно перевірити чи видно їх та чи можна на них натиснути (лістинг 2.1).

Перевірка приймає як параметр buttonName і для кожного з двох варіантів перевіряє чи існує кнопка з даним, як параметр, текстом як ім’я на вебсторінці та чи можна на неї натиснути.

Конструкція «this.button» шукає кнопку за унікальним для кнопки HTML-селектором на вебсторінці, конструкція «break» дозволяє виходити з циклу перевірки кожного параметра та переходити до наступного.

Лістинг 2.1 Скрипт регресійного тестування за допомогою програмного коду:

```
switch (buttonName) {
  case ‘buttonName1’:
    cy.get(this.button).contains(buttonName1).should(‘be.visible’).click()
    break;
  case ‘buttonName2’:
    cy.get(this.button).contains(buttonName2).should(‘be.visible’).click()
    break;
  default:
    console.log(‘No buttons available’)
    break;
}
```

Якщо жодна з перших двох умов не буде виконана, то буде реалізована базова умова з текстом «Немає даних кнопок», що свідчить про те, що регресійна перевірка є невдалою та, як наслідок, розробники вебзастосунку мають перевірити правильність внесених змін.

2.4 Дослідження методу тестування «Модульне тестування»

Модульне тестування – тестування, мета якого перевірити працездатність окремих модулів (функції або класу) [38]. Модульне тестування є найпоширенішим видом тестування при тестуванні невеликих електронних систем.

Традиційне визначення поняття модуля з точки зору тестування: «модуль-компонент мінімального розміру, який може бути незалежно протестований в процесі верифікації системи». Наприклад у системі розпізнавання обличь на фото модулем буде інтелектуальна система розпізнавання [39].

Головна ціль модульного тестування – точно впевнитися у відповідності висунутим вимогам для окремого модуля програмної системи перед тим, як проводити інтеграцію до складу всієї системи. При цьому під час модульного тестування вирішуються основні чотири завдання [39].

Модулі, які піддаються тестуванню, зазвичай, невеликі за розміром, тому модульне тестування вважається найпростішим, але дуже трудомістким етапом тестування всієї системи.

Проте, незважаючи на простоту у виконанні, з модульним тестуванням пов’язано дві основні проблеми:

– відмінності в трактуванні поняття модульного тестування;

– не існує єдиних принципів для визначення окремого модуля.

Отже, як наслідок, модульне тестування допомагає спеціалісту-тестувальнику без складнощів виконувати зміни на кожному з етапів тестування та у будь-який момент проаналізувати кожен з модулів на функціональність.

Схема процесу реалізації модульного тестування у вигляді блок-схеми наведена на рисунку 2.11.

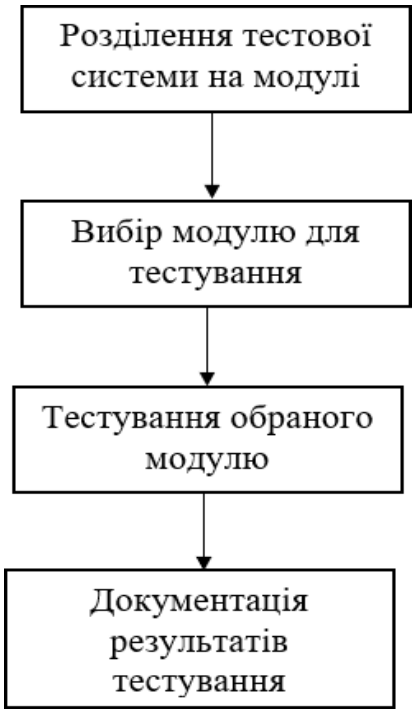

Рисунок 2.11 – Схема процесу реалізації модульного тестування

Для прикладу реалізації модульного тестування у програмному коді розглянемо кейс, при якому автоматизований тест має перевірити можливість введення тексту в текстове поле та перевірити його відображення у полі, потім очистити поле та перевірити те, що текст був видалений, потім ввести у поле цифри та перевірити, що вони не відобразилися (поле може приймати лише літери).

Програмний код реалізований за допомогою мови програмування JavaScript та фреймворку Cypress.

Лістинг 2.2 Скрипт модульного тестування за допомогою програмного коду:

```
function fillTextInput () {
cy.get(this.textInput). fill (‘Hello, world’);
cy.get(this.textInput).contains(‘Hello, world’).should(‘be.visible’);
cy.get(this.textInput).clear();
cy.get(this.textInput). fill (‘12345’);
cy.get(this.textInput). contains (‘12345’).should(‘not.be.visible’)
}
```

За допомогою цього програмного коду реалізується модульна перевірка на здатність текстового поля приймати допустимі символи та не відображати недопустимі, що є повноцінною перевіркою текстового поля як модуля.

2.5 Дослідження методу тестування «Інтеграційне тестування»

Інтеграційне тестування як вид вебтестування використовується для перевірки взаємодії між окремими модулями систем, а також перевірки цілісної системи як результату взаємодії окремих модулів.

При інтеграційному тестуванні існує декілька підходів:

– знизу вверх. Спочатку збираються компоненти та/або модулі самих низьких рівнів, поступово піднімаючись по ієрархії, доходячи до цілої системи. Для використання цього підходу необхідно, щоб всі компоненти були реалізовані і готові до тестування;

– зверху вниз. Цей підхід починає тестування з самого високого рівня системи і йде вниз до самих мінімальних компонентів. У процесі тестування цим методом у кожен компонент вищого рівня замість підстановки справжніх компонентів нижчих рівнів підставляються їх заглушки, що дозволяє тестувати логіку тільки поточного компонента, не спираючись на його залежності;

– великий вибух. Усі компоненти збираються в єдину повну систему і тестуються на функціях системи [40].

Схема процесу реалізації інтеграційного тестування наведена на рисунку 2.12.

Рисунок 2.12 – Схема процесу реалізації інтеграційного тестування

Для прикладу реалізації інтеграційного тестування розглянемо скрипт, написаний на мові програмування JavaScript та реалізований за допомогою фреймворку Cypress, який перевіряє те, що при натисканні на поле для вводу тексту з’являється випадаючий список елементів, кожен з яких може бути результатом пошуку по введеному у поле тексту.

Лістинг 2.3 Скрипт інтеграційного тестування за допомогою програмного коду:

```
function checkDropdownMenuVisible () {
cy.get(this.input).click();
cy.get(this.dropdownMenu).should(‘be.visible’}
}
```

За допомогою даного скрипту реалізується інтеграційне тестування випадаючого списку у поле для введення тексту.

2.6 Аналіз можливих непередбачуваних ситуацій при реалізації обраних видів тестування

У сучасному світі вебзастосунки займають велику частину життя і можуть допомагати у полегшенні процесу отримання медичної допомоги, нових знань та інше.

Наприклад, в останній час у розвитку освіти з’явилася додаткова проблема: знання старіють кожні три-п’ять років, а технологічні знання кожні два-три роки. Обсяг знань випускників університетів подвоюється кожні три-чотири роки. Якщо це не змінить освітніх технологій, то якість підготовки спеціалістів буде відставати на ринку праці. Засвоєння знань за допомогою сучасних інформаційних технологій відбувається швидше, ніж використання звичайних технологій [41].

Сучасні комп’ютерні технології можуть забезпечити передачу знань та доступ до різноманітної навчальної інформації ефективніше, ніж традиційні засоби навчання. Крім того, використання нових технологій підвищує зацікавленість до процесу навчання.

Слід зазначити, що системи дистанційного навчання не мають особливих вимог до комп’ютерної підтримки, але налаштовані таким чином, що хтось автоматично оновлює навчально-методичні матеріали за допомогою Інтернету [41].

Ще однією важливою перевагою технологій дистанційного навчання є вартість навчання. Наприклад, керівники Microsoft вважають, що вартість онлайн-навчання може становити щонайменше половину вартості традиційного навчання, оскільки викладач може проводити заняття будь-де. Таким чином, роль викладача має консультативне значення, звільняючи час викладача для наукової діяльності [41].

Важливим елементом процесу реалізації вебтестування є автоматизація за допомогою програмного коду, вдала реалізації якої неможлива без вбудованої технології прийняття рішень (порівняння вхідної інформації від програмного коду з вихідною з об’єкта тестування).

Розвиток інформаційних технологій дозволив створити інформаційні системи – складні програмні продукти, які реалізують інформаційні технології, призначені для комп’ютерного моделювання різноманітних процесів з метою вирішення широкого кола задач.

Технології прийняття рішень в інформаційних системах мають широкі можливості інтеграції і сумісного аналізу різнорідних даних та є незамінним інструментом для вирішення задач управління. Інформаційні системи та технології прийняття рішень застосовуються під час автоматизації оброблення інформації про об’єкти будь-якого походження [42].

Важливим етапом розвитку інформаційних технологій є верифікація взаємодії окремих модулів як системи, тобто інтеграційне тестування.

Результатом реалізації інтеграційного тестування є тестування коректності взаємодії декількох модулів, об’єднаних в єдине ціле. Таке тестування називають інтеграційним. Його мета – упевнитися в коректності спільної роботи компонент системи. Наприклад, інтеграційне тестування геоінформаційної системи має перевірити можливість взаємодії навігаційних та UI-модулів [43].

У результаті проведення інтеграційного тестування та усунення всіх виявлених дефектів виходить погоджена та цілісна архітектура програмної системи, тобто можна вважати, що інтеграційне тестування – це тестування архітектури та низькорівневих функціональних вимог.

Інтеграційне тестування, як правило, є ітеративним процесом, при якому перевіряється можливість модулів до об’єднання у єдину систему [43].

Переваги інтеграційного тестування:

– дуже зручно реалізовувати, якщо системи невеликі. Часу для цього підходу необхідно більше, тому великі системи можуть призвести до більшого споживання часу;

– виявлення несправностей за цим методом є дуже легким процесом, якщо аналізують невеликі системи.

Недоліки інтеграційного тестування:

– оскільки всі модулі з’єднані, тому, якщо в системах виникає якась несправність, то важко помітити її;

– деякі модулі дуже важливі і їх потрібно перевірити. Вони повинні бути протестовані перед використанням у системі. Але під час інтеграційного тестування ці модулі можуть не бути перевірені ефективно, оскільки вони з’єднані між собою;

– час, який забирає вся програмна система, набагато більший, ніж інші підходи до інтеграційного тестування;

– з’єднання модулів може зайняти певний час, що може вплинути на загальний час роботи програмної системи;

– час для цього підходу більший, оскільки багато модулів з’єднані разом, тестування кожного модуля займає більше часу.

Інтеграційне тестування може бути реалізоване також для більш специфічних засобів, таких як застосунки на базі апарату нечіткої логіки.

Об’єкти спеціального призначення функціонують у апріорній невизначеності, що характеризується нечітким простором станів, потребує нових інтелектуальних підходів для підвищення надійності прийнятих рішень, характеризується функціональним та територіальним розподілом, а також складною ієрархією взаємодіючих процесів.

Необхідно передбачити певні вимоги до математичного апарату, методів об’єктно-орієнтованого моделювання та аналізу взаємодіючих процесів складної системи. Існуючі підходи до аналізу, моделювання та побудови складних систем управління та оброблення даних неефективні через їхні функціональні обмеження.

Слід зазначити, що перспективним інструментом для побудови складних систем, а також для аналізу та моделювання взаємодіючих процесів складних систем є використання нечіткої логіки Лотфі Заде [44].

Отже, використання інтеграційного тестування щодо вебзастосунків, пов’язаних із навчанням чи медициною, прийняттям рішень, застосунків на базі апарату нечіткої логіки дозволяє визначити такі непередбачувані результати:

– для вебзастосунків щодо сфери навчання або медицини непередбачуваним небажаним результатом може бути помилка в інтеграції модулів у більш значному масштабі, ніж очкувалося, наприклад, при натисканні на поле вводу тексту випадаючий список не тільки не з’являється, але й вся сторінка перестає працювати. Непередбачуваним, але позитивним, результатом, наприклад, може бути менший час на появу випадаючого списку, ніж очікувалося;

– для застосунків щодо прийняття рішень чи на базі апарату нечіткої логіки небажаним непередбачуваним результатом будуть помилки у обробленні вхідної інформації і, як наслідок, помилки у видачі результату. Позитивним непередбачуваним результатом буде менший час на аналіз вхідної інформації і видачу результату.

Модульне тестування може проводитися на багатьох вебзастосунках:

– геоінформаційних вебзастосунках. Геоінформаційна система є новою системою орієнтування у часі і просторі, вона включає в себе сучасні методи оброблення інформації і, у той же час, є доступною для більшості людей. Застосування геоінформаційних систем дозволяє на якісно новому рівні забезпечити інформаційною базою практично всі служби та на цій основі вирішити технічні, економічні та інші завдання. Вона використовується у розвитку та управлінні об’єктами нерухомості [45];

– вебзастосунках, пов’язаних із розпізнаванням зображень. Проблема розпізнавання структурних зображень є актуальною, це перспективний спосіб оцінити ступінь подібності об’єктів. Такий підхід забезпечує простоту конструкції та високу надійність прийняття рішень. Основною проблемою ефективного опису характерних ознак є спотворення фрагментів аналізованих об’єктів. Причинами зміни вхідних даних можуть бути дії геометричних перетворень, вплив фону або перешкод. Забезпечення якісного розпізнавання вимагає впровадження ефективних засобів оброблення зображень. Використовуються методи статистичного моделювання, гранулювання даних та нечітких множин, виявлення та порівняння ключових точок на зображенні, класифікація та групування даних, а також імітаційне моделювання [46];

– вебзастосунках, пов’язаних з моделюванням складних об’єктів на основі інтерпретації розробленої моделі із застосуванням інструментального засобу моделювання та аналізу взаємодіючих процесів [47].

Переваги модульного тестування:

– розбиття системи на невеликі частини, що сприяє більш глибокому структурному аналізу;

– у процесі модульного тестування можуть бути знайдені недоліки у інтеграції модулів і внесені структурні зміни;

– невеликий час на тестування окремого модулю.

Недоліки модульного тестування:

– велика ймовірність тестування складного коду;

– очікуваний результат тестування є тільки приблизним;

– помилки в інтеграції ускладнюють процес виділення модулів як окремих частин.

Непередбачувані результати при проведенні модульного тестування можуть бути наступними:

– бажаним непередбачуваним результатом може бути знайдена можливість для покращення інтеграційної логіки з метою збільшення зручності у реалізації цілісної системи. Також позитивним непередбачуваним результатом можуть бути знайдені моменти, де необхідно додати чи видалити певні модулі з метою покращення діяльності системи;

– небажаним непередбачуваним результатом можуть бути нереалістичні очікування від тестування модулів, які призводять до формування нереалістичних тест-кейсів. Також поганим результатом може бути складна інтеграційна логіка, яка призводить до неможливості виокремлення окремих модулів.

Для прикладу реалізації регресійного тестування можливо розглянути наступні вебзастосунки:

– застосунки, що займаються побудовою моделей, здатних приймати правильні рішення в умовах неповної і нечіткої інформації та використання їх у комп’ютерних системах [48];

– застосунки, що реалізують інформаційну підготовку до прийняття рішень щодо будівництва навчальних закладів у зоні високого попиту та розвиненої інфраструктури, надають можливість визначити оптимальне та раціональне місцезнаходження необхідних просторових об’єктів [49];

– застосунки комп’ютерного зору, що реалізують моделі оцінки статистичних даних під час визначення та класифікації релевантності зображення, впроваджують інтегровані критерії релевантності даних на основі цих розрахункових значень [50].

Переваги регресійного тестування:

– можливість відстежувати короткострокові зміни у застосунку завдяки регресійним тестам;

– можливість за порівняно невеликий проміжок часу покрити тестами значну частину функціоналу;

– можливість знайти виправлення, що призводять до помилок в інших місцях системи.

Недоліки регресійного тестування:

– необхідність запускати повний набір тест-кейсів після внесення змін у застосунок;

– необхідність постійного оновлення тест-кейсів відповідно до незначних змін у функціоналі застосунку;

– можливість появи тест-залежних один від одного тест-кейсів.

До бажаних непередбачуваних результатів модульного тестування можливо віднести знайдені невеликі зміни у застосунку, які у майбутньому, при додаванні нового функціоналу можуть призвести до значних помилок. У такому випадку модульне тестування допомагає швидко виявити та усунути майбутні складнощі у функціонуванні системи.

До небажаних непередбачуваних результатів можливо віднести неможливість адаптації набору тест-кейсів до значних змін у функціонуванні застосунку, що призведе до необхідності написання нового програмного коду.

3 ЗАСТОСУВАННЯ МЕТОДІВ ВЕБТЕСТУВАННЯ ЩОДО ВИБРАНОЇ ПРЕДМЕТНОЇ ОБЛАСТІ

3.1 Вибір інструментальних засобів для реалізації вибраних методів

Для реалізації автоматизованого тестування вебзастосунків перш за все необхідно мати середовище розроблення (IDE) тест-кейсів, написаних за допомогою мови програмування. Для практичної реалізації тест-кейсів була обрана IDE WebStorm.

WebStorm забезпечує автодоповнення, аналіз коду «на льоту», навігацію по коду, рефакторинг, зневадження та інтеграцію з системами управління версіями. Важливою перевагою інтегрованого середовища розробки WebStorm є робота з проєктами (у тому числі, рефакторинг коду JavaScript, що міститься в різних файлах і теках проєкту, а також вкладеного в HTML). Підтримується множинна вкладеність (коли в документ на HTML вкладений скрипт на JavaScript, в який вкладено інший код HTML, всередині якого вкладений JavaScript) – у таких конструкціях підтримується коректний рефакторинг [51].

Наступним кроком після вибору IDE є вибір мови програмування для реалізації програмних тест-кейсів. Для реалізації була обрана базова мова програмування IDE WebStorm – JavaScript.

JavaScript – динамічна, об’єктно-орієнтована прототипна мова програмування, реалізація стандарту ECMAScript. Використовується для створення сценаріїв вебсторінок, що надає можливість на боці клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд вебсторінки.

JavaScript класифікують як прототипну (підмножина об’єктно-орієнтованої), скриптову мову програмування з динамічною типізацією.

Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування (імперативну та частково функціональну) і деякі відповідні архітектурні властивості, зокрема: динамічна та слабка типізація, автоматичне керування пам’яттю, прототипне наслідування, функції як об’єкти першого класу [52].

Сучасні тенденції у програмуванні сприяють програмній реалізації Behavior Driven Development (BDD).

BDD фокусується на наступних питаннях:

– з чого починається процес;

– що потрібно тестувати, а що ні;

– скільки перевірок повинно бути вчинено за один раз;

– що можна назвати перевіркою;

– як зрозуміти, чому тест не пройшов.

Виходячи з цих питань, BDD вимагає, щоб імена тестів були цілими реченнями, які починаються з дієслова в умовному способі та досягали бізнес-цілі. Опис приймальних тестів повинен вестися на гнучкій мові (наприклад, англійською мовою) [53].

Для запуску написаних програмних тест-кейсів доцільно використати фреймворк Cypress.

Переваги Cypress:

– вбудований набір інструментів для тестування;

– вбудований механізм автоматичного очікування;

– можливість повертатися на будь-який крок тесту;

– вичерпна документація з великим набором прикладів;

– можливість написання в тому числі і unit-тестів [54].

Останнім етапом є вибір інструментарію для формування тест-кейсів та створення діаграм формування планів з тестування за допомогою застосунку ERwin.

Інструментальний засіб ERwin є програмним продуктом, що надає необхідні для розробника функціональні можливості по моделюванню та

документуванню моделей бази даних з подальшою трансформацією фізичної моделі в базу даних [55].

3.2 Етапи програмної реалізації вибраних методів вебтестування

Програмна реалізація вибраних методів вебтестування передбачає виконання наступних етапів.

Етап 1. Визначення вихідних даних для практичної реалізації.

Вихідними даними для реалізації обраних видів тестування є:

– змодельований процес тестування за допомогою застосунку ERwin;

– мова програмування JavaScript;

– IDE для написання програмних тест-кейсів WebStorm;

– фреймворк для запуску готових тест-кейсів Cypress;

– вебзастосунок для тестування Demoblaze (предметна область – онлайн торгівля).

Етап 2. Визначення особливостей предметної області для застосування методів.

У якості тестової платформи був обраний тестовий вебзастосунок Demoblaze, який симулює діяльність e-commerce магазину з продажу смартфонів, ноутбуків та іншого.

Задачами реалізації вебтестування на тестовій платформі є такі:

– перевірити можливість відкриття сторінки з товаром;

– перевірити можливість переходу між вкладками з товаром;

– перевірити вірність відображення зображень з товаром;

– перевірити можливість додавання товару до корзини;

– перевірити валідність полів заповнення замовлення.

Відповідно до обраних раніше методів тестування тестовою задачею аналізу предметної області буде можливість безперебійного користування e-commerce застосунком.

Етап 3. Класифікація задач предметної області.

Обрані задачі можливо класифікувати наступним чином:

– тестування можливості входу на сторінку;

– тестування навігації;

– тестування вірності відображення інформації про товар;

– тестування безперебійності процесу створення замовлення.

Етап 4. Програмна реалізація перевірки відкриття сторінки з товаром за допомогою вибраних методів тестування.

Програмна реалізація перевірки відкриття сторінки з товаром за допомогою модульного тестування наведена у лістингу 3.1.

Лістинг 3.1 Реалізація перевірки відкриття сторінки з товаром за допомогою модульного тестування:

```
it ('check link clickability', () => {
cy.get('[class="hrefch"]').eq(0).should('not.be.disabled')
})
```

Даний скрипт перевіряє те, що посилання, за яким відкривається сторінка з товаром, не є деактивованим.

Програмна реалізація перевірки відкриття сторінки з товаром за допомогою інтеграційного тестування наведена у лістингу 3.2.

Лістинг 3.2 Реалізації перевірки відкриття сторінки з товаром за допомогою інтеграційного тестування:

```
it('check link position', () => {
cy.get('[class="card-block"]').eq(0).find('[class="hrefch"]').should('be.visible')
})
```

Даний скрипт перевіряє те, що посилання на товар находиться саме у середині елементу з описом товару.

Програмна реалізація перевірки відкриття сторінки за допомогою регресійного тестування наведена у лістингу 3.3.

Лістинг 3.3 Реалізація перевірки відкриття сторінки з товаром за допомогою регресійного тестування:

```
it ('click the link', () => {
cy.get('[class="cardblock"]').eq(0).find('[class="hrefch"]').should('be.visibl
e').click();
 cy.get('[class="hrefch"]').should('not.exist')
})
```

Даний скрипт перевіряє те, що посилання знаходиться у елементі з описом товару, натискає на нього і очікує на те, що елемент з посиланням не буде видимий, це свідчить про перехід на нову сторінку.

Етап 5. Програмна реалізація можливості переходу між вкладками за допомогою вибраних методів тестування.

Програмна реалізація можливості переходу між вкладками за допомогою модульного тестування наведена у лістингу 3.4.

Лістинг 3.4 Реалізації перевірки можливості переходу між категоріями за допомогою модульного тестування:

```
it('check navigation button visibility', () => {
cy.get('[onclick="byCat(\'notebook\')"]').should('be.visible');
})
```

Даний скрипт приймає у якості параметру тип товару та перевіряє наявність модулів з різними типами товару.

Програмна реалізація можливості переходу між вкладками за допомогою інтеграційного тестування наведена у лістингу 3.5.

Лістинг 3.5 Реалізація перевірки можливості переходу між категоріями за допомогою інтеграційного тестування:

```
it('check button position', () => {cy.get('#itemc').parent('[class="list-group"]')})
```

Даний скрипт перевіряє наявність елементу навігаційної кнопки з товаром у єдиному блоці навігаційних елементів.

Програмна реалізація можливості переходу між вкладками за допомогою регресійного тестування наведена на лістингу 3.6.

Лістинг 3.6 Реалізація перевірки можливості переходу між категоріями за допомогою регресійного тестування:

```
it ('open different page', () => {
cy.get('[class="hrefch"]').eq(0).should('contain.text', 'Samsung galaxy s6')
cy.get('[class="listgroup"]').should('be.visible').find('[onclick="byCat(\'notebook\'
)"]').wait(2000).click({force:true})
cy.wait(3000).get('[class="hrefch"]').eq(0).should('contain.text', 'Sony vaio i5')
cy.get('[class="hrefch"]').eq(0).should('not.contain.text', 'Samsung galaxy s6')
})
```

Даний скрипт перевіряє видимість усіх необхідних для тесту елементів, натискає на навігаційну кнопку та перевіряє відкриття необхідної сторінки.

Етап 6. Програмна реалізація можливості відображення зображення фото з товаром за допомогою вибраних методів тестування.

Перевірка вірності відображення зображення фото з товаром необхідна для стабільної роботи магазину (від вірного зображення товару залежить легкість пошуку необхідного товару).

Програмна реалізація вірності відображення фото за допомогою модульного тестування наведена у лістингу 3.7.

Лістинг 3.7 Реалізація перевірки вірності відображення фото з товаром за допомогою модульного тестування:

```
it ('check only one photo visible', () => {
cy.get('[class="card h-100"]').eq(0).find('img').eq(0).should('be.visible')
cy.get('[class="card h-100"]').eq(0).find('img').eq(1).should('not.exist')
})
```

Даний скрипт перевіряє те, що у одному блоці товару знаходиться тільки одне фото з товаром.

Програмна реалізація вірності відображення фото за допомогою інтеграційного тестування наведена у лістингу 3.8.

Лістинг 3.8 Реалізація перевірки вірності відображення фото з товаром за допомогою інтеграційного тестування:

```
it('check right photo displayed', () => {
cy.get('[class="card h-
100"]').eq(0).find('[class="hrefch"]').should('contain.text','Samsung galaxy s6')
cy.get('[class="card h-
100"]').eq(0).find('[src="imgs/galaxy_s6.jpg"]').should('be.visible')
})
```

Даний скрипт порівнює назву фото та назву товару у посиланні, що свідчить про відображення вірного фото для певного товару.

Програмна реалізація вірності відображення фото за допомогою регресійного тестування наведена у лістингу 3.9.

Лістинг 3.9 Реалізація перевірки вірності відображення фото з товаром за допомогою регресійного тестування:

```
it ('check all product set visibility', () => {
cy.get('[class="card h-100"]').eq(0).should('not.be.empty')
})
```

Даний скрипт перевіряє видимість усіх необхідних для тесту елементів.

Етап 7. Програмна реалізація можливості додавання товару у корзину за допомогою вибраних методів тестування.

Перевірка можливості додавання товару у корзину є однією з двох найважливіших перевірок цього тесту, бо без вірної роботи цього функціоналу діяльність магазину неможлива.

Програмна реалізація можливості додавання товару до корзини за допомогою модульного тестування наведена у лістингу 3.10.

Лістинг 3.10 Реалізація перевірки можливості додавання товару до корзини за допомогою модульного тестування:

```
it ('check add to cart button state', () => {
cy.get('[class="card h-100"]').eq(0).find('[class="hrefch"]').click()
cy.get('.btn-success').should('be.visible')
cy.get('.btn-success').should('not.be.disabled')
})
```

Даний скрипт перевіряє те, що кнопка додавання до корзини видна, не є відключеною та натискає на неї.

Програмна реалізація можливості додавання товару до корзини за допомогою інтеграційного тестування наведена у лістингу 3.11.

Лістинг 3.11 Реалізація перевірки можливості додавання товару до корзини за допомогою інтеграційного тестування:

```
it ('check add to cart button position', () => {
cy.get('[class="card h-100"]').eq(0).find('[class="hrefch"]').click()
cy.get('#tbodyid').find('.btn-success').trigger('mouseenter')
})
```

Даний скрипт перевіряє те, що кнопка додавання до корзини знаходиться у середині блоку з описом товару.

Програмна реалізація можливості додавання товару до корзини за допомогою регресійного тестування наведена у лістингу 3.12.

Лістинг 3.12 Реалізація перевірки можливості додавання товару до корзини за допомогою регресійного тестування:

```
it('check add product to card', () => {
cy.get('[class="card h-100"]').eq(0).find('[class="hrefch"]').click()
cy.get('#tbodyid').find('.btn-
success').trigger('mouseenter').wait(2000).click({force:true}).wait(2000)
cy.reload() cy.wait(3000).scrollTo('top').get('[class="container"]').
find('[class="nav-link"]').contains('Cart').click()
cy.wait(2000).get('[class="success"]').should('be.visible') })
```

Даний скрипт додає товар до корзини та перевіряє те, що він там з’явився.

Етап 8. Програмна реалізація перевірки валідності полів заповнення замовлення за допомогою вибраних методів тестування.

Останньою перевіркою є перевірка валідності полів заповнення замовлення, ця перевірка є не менш важливою під час створення замовлення.

Програмна реалізація перевірки валідності полів заповнення замовлення за допомогою модульного тестування наведена у лістингу 3.13.

Лістинг 3.13 Перевірка валідності полів заповнення замовлення за допомогою модульного тестування:

```
it ('check order fields visibility', () => {
cy.get('[class="card h-100"]').eq(0).find('[class="hrefch"]').click()
cy.get('#tbodyid').find('.btn-
success').trigger('mouseenter').wait(2000).click({force:true}).wait(2000)
cy.reload() cy.wait(3000).scrollTo('top').get('[class="container"]').
find('[class="nav-link"]').contains('Cart').click()
```

```
cy.wait(2000).get('[class="success"]').should('be.visible') cy.get('[data-
target="#orderModal"]').click() cy.get('#name').should('be.visible')
cy.get('#country').should('be.visible') cy.get('#city').should('be.visible')
cy.get('#card').scrollIntoView().should('be.visible')
cy.get('#month').scrollIntoView().should('be.visible')
cy.get('#year').scrollIntoView().should('be.visible')
})
```

Даний скрипт перевіряє видимість полів заповнення після натискання на кнопку «place order».

Даний скрипт є прикладом економії місця та часу. Конструкція ‘*this.checkAddedProductVisibility()’* викликає метод, що створює замовлення, таким чином, немає необхідності у повторенні написаного раніше коду. Конструкція ‘*${fieldName}*’ знайде та перевірить кожне з необхідних полів за співпадаючим селектором, що також усуває необхідність зайвого коду.

Реалізація перевірки валідності полів заповнення замовлення за допомогою інтеграційного тестування наведена у лістингу 3.14.

Лістинг 3.14 Перевірка валідності полів заповнення замовлення за допомогою інтеграційного тестування:

```
it('check order fields position', () => {
cy.get('[class="card h-100"]').eq(0).find('[class="hrefch"]').click()
cy.get('#tbodyid').find('.btn-
success').trigger('mouseenter').wait(2000).click({force:true}).wait(2000)
cy.reload() cy.wait(3000).scrollTo('top').get('[class="container"]').
find('[class="nav-link"]').contains('Cart').click()
cy.wait(2000).get('[class="success"]').should('be.visible')
cy.get('[data-target="#orderModal"]').click().wait(1000)
cy.get('.modal-body').find('#name').should('be.visible') cy.get('.modal-
body').find('#country').should('be.visible')
cy.get('.modal-body').find('#city').should('be.visible')
```

```
cy.get('.modal-body').find('#card').should('be.visible')
cy.get('.modal-body').find('#month').should('be.visible')
cy.get('.modal-body').find('#year').scrollIntoView().should('be.visible'
})
```

Даний скрипт перевіряє видимість полів у середині діалогового вікна з замовленням.

Програмна реалізація перевірки валідності полів заповнення замовлення за допомогою регресійного тестування наведена у лістингу 3.15.

Лістинг 3.15 Перевірка валідності полів заповнення замовлення за допомогою регресійного тестування:

```
it ('fill order fields', () => { cy.get('[class="card h-
100"]').eq(0).find('[class="hrefch"]').click() cy.get('#tbodyid').find('.btn-
success').trigger('mouseenter').wait(2000).click({force:true}).wait(2000)
cy.reload()
cy.wait(3000).scrollTo('top').get('[class="container"]') .find('[class="nav-
link"]').contains('Cart').click()
 cy.wait(2000).get('[class="success"]').should('be.visible') cy.get('[data-
target="#orderModal"]').click()
cy.get('#name').type('Test') cy.get('#country').type('Test')
cy.get('#city').type('Test') cy.get('#card').scrollIntoView().type('111222333')
cy.get('#month').scrollIntoView().type('August')
cy.get('#year').scrollIntoView().type('2021')
})
```

Даний скрипт заповнює кожне поле. Зазначені параметри допомагають зробити скрипт більш гнучким та використати його під різні дані.

3.3 Застосування методів вебтестування щодо вибраної предметної області

Для запуску тест-кейсів через Cypress можливо використання багатьох команд, базовими з яких є «Cypress open» – команда, що відкриває тест-кейси з використанням UI (тест-кейси проходять з візуалом роботи тестової бази) та «Cypress run» – команда, що запускає тести у так званому headless-режимі (тести реалізуються у консолі WebStorm).

Запуск тестів проводився на одному персональному комп'ютеру зі швидкістю інтернету в межах 75 мб/с.

Команда запуску наведена на рисунку 3.1.

У даному випадку UI запуск тестів буде більш вдалим рішенням, він дозволить наглядно продемонструвати реалізацію тест-кейсів.

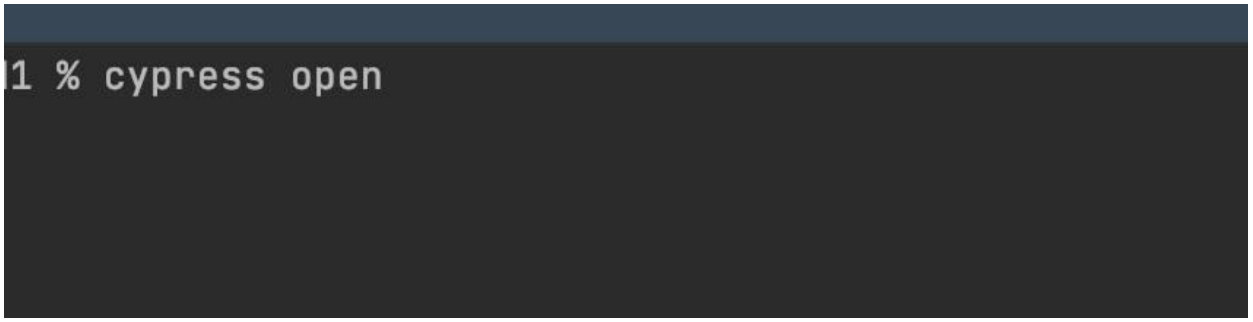

Рисунок 3.1 – Команда запуску тестів з використанням UI елементів

Після реалізації команди «Cypress open» відкривається діалогове вікно зі списком наявних тест-кейсів (рис. 3.2).

На даному вікні є можливість пошуку тест-кейсів за назвою, запускати по одному чи усі, що наявні, одразу або запускати тести у CI інтеграції.

Базовим браузером Cypress є Chrome, базовою заміною є браузер Electron, також можливо додати велику кількість інших відомих браузерів.

Команда запуску тестових сценаріїв у headless-режимі наведена на рисунку 3.3.

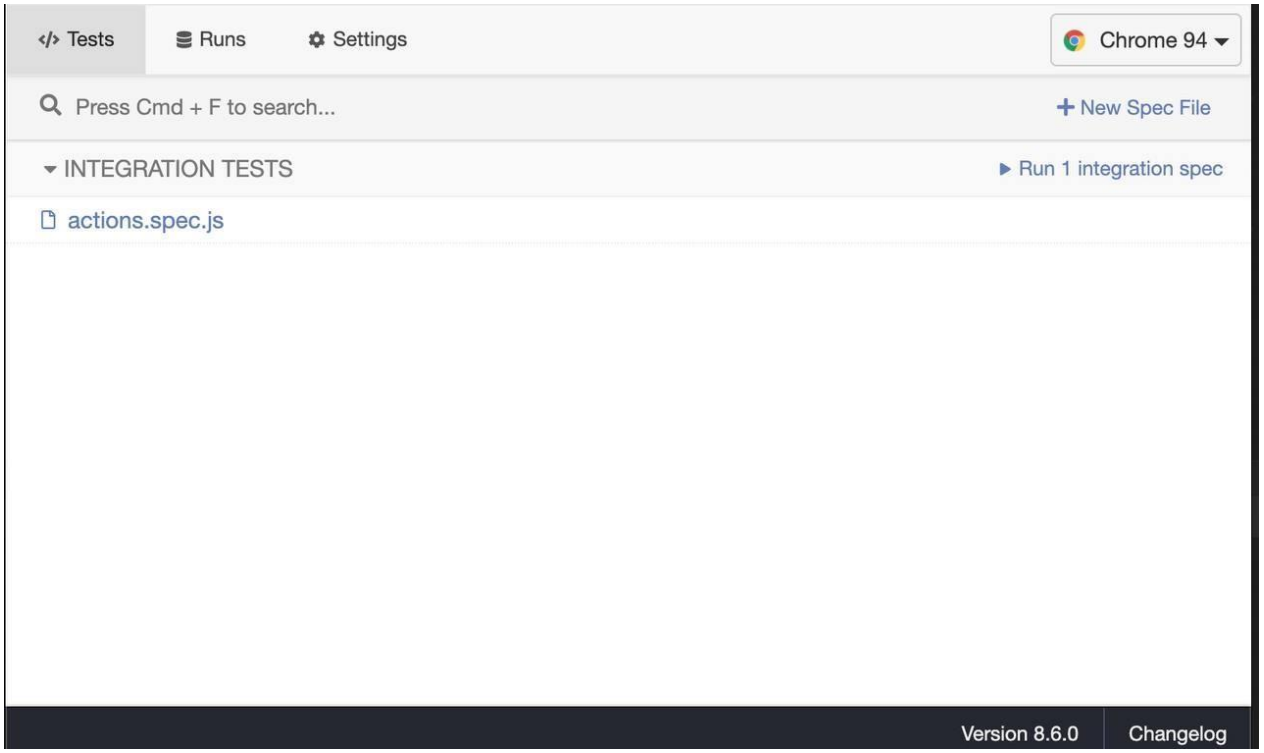

Рисунок 3.2 – Базове вікно Cypress

```
% cypress run
```

Рисунок 3.3 – Команда запуску тестових сценаріїв у headless-режимі

Приклад реалізації тестового сценарію у headless-режимі наведений на рисунку 3.4.

```
Test suite for Demoblaze
  ✓ check link clickability (2284ms)


1 passing (4s)
```

Рисунок 3.4 – Приклад реалізації тестового сценарію у headless-режимі

У процесі реалізації тестових сценаріїв у програмному коді перш за все визначається параметр beforeEach (базова дія Cypress, що виконується перед кожним тестовим сценарієм).

Для даної тестової бази базовою дією буде вхід на сайт (рис. 3.5).

```
beforeEach( fn: () => {
  cy.visit('https://www.demoblaze.com/')
})
```

Рисунок 3.5 – Приклад застосування beforeEach дії

Після входу на тестову базу починається реалізація першого наявного тест-сценарію. Реалізація перевірки відкривання сторінки з товаром за допомогою модульного тестування наведена на рисунку 3.6.

```
it( title: 'check link clickability', fn: () => {
  cy.get('[class="hrefch"]').eq( index: 0).should( chainer: 'not.be.disabled')
})
```

Рисунок 3.6 – Приклад реалізації тестового сценарію щодо перевірки посилання на можливість натиснути у програмному коді

Час проходження сценарію у UI-режимі склав 4 секунди.

Час проходження сценарію у headless-режимі склав 2 секунди.

Даний код бере CSS селектор першого посилання та за ним перевіряє, що дане посилання не є відключеним. CSS селектор формально описує обраний елемент з точки зору DOM-дерева. DOM-дерево допомагає програмам та скриптам отримувати доступ до HTML-елементів тестової бази. Приклад DOM-дерева з найденим на ньому селектором першого тестового сценарію наведений на рисунку 3.7.

Як видно, на рисунку 3.7 знайдено дев’ять результатів пошуку за селектором, що свідчить про знаходження дев’яти посилань на протестованій сторінці. Використання індексу 0 (у мові програмування JavaScript під 0 йде перший елемент) дозволяє Cypress працювати саме з першим елементом, що нівелює не проходження сценарію через велику кількість співпадань (CSS селектор має бути унікальним).

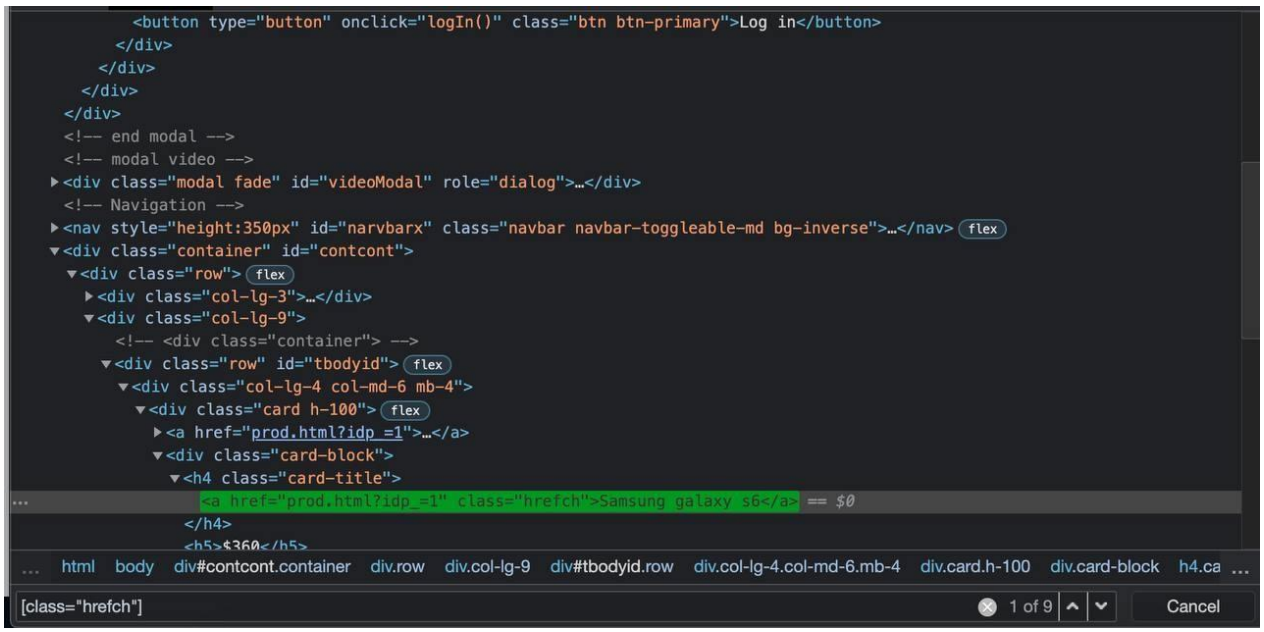

Рисунок 3.7 – Приклад DOM-дерева з найденим селектором

Розглянемо реалізацію першого сценарію у Cypress, як приклад проходження тестових сценаріїв (рис. 3.8).

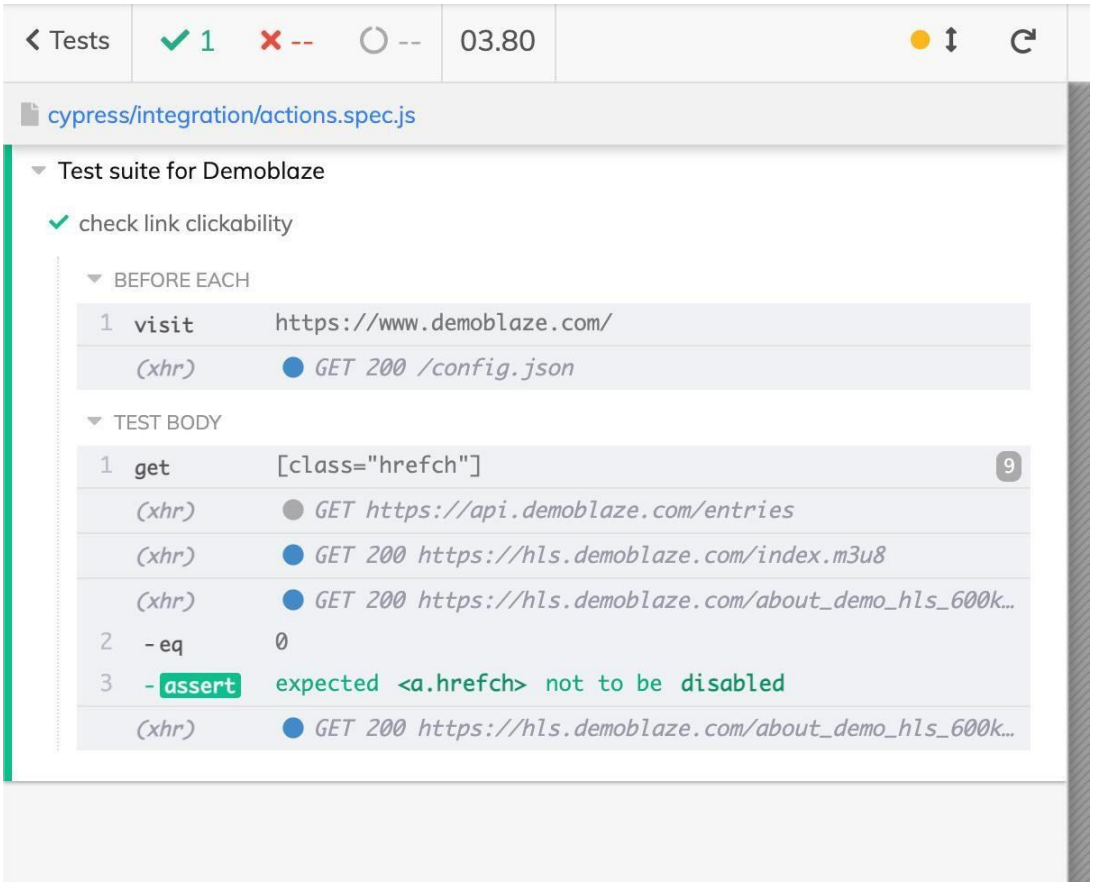

Рисунок 3.8 – Вікно проходження сценарію у Cypress

Вікно сценарію у Cypress є максимально інформативним, відображає час проходження тесту, його статус покроково (рядок assert), кількість пройдених та непройдених сценаріїв.

Також вікно Cypress відображає саму тестову базу (рис. 3.9).

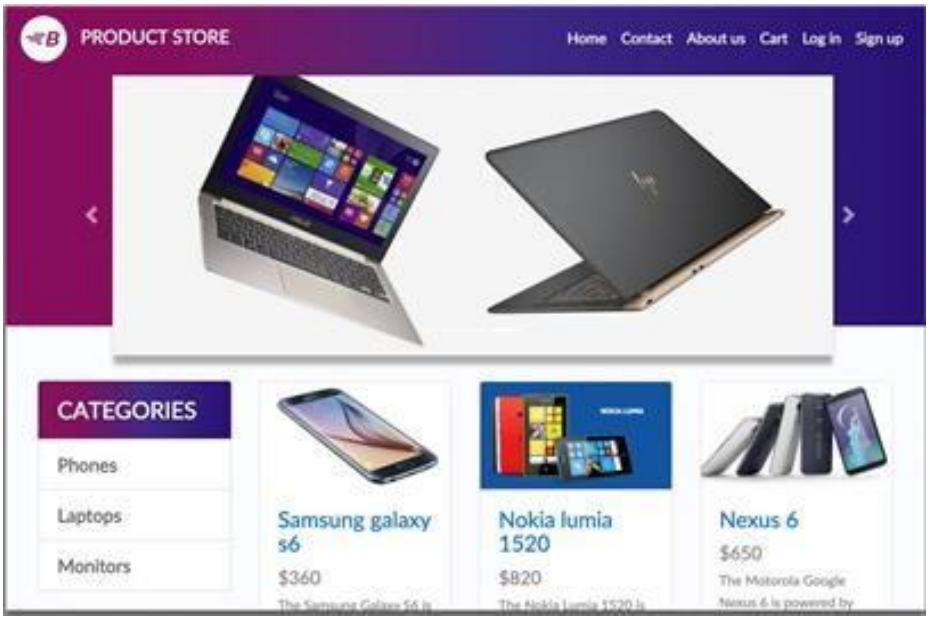

Рисунок 3.9 – Приклад відображення тестової бази у Cypress

Реалізація сценарію перевірки можливості відкриття сторінки з товаром за допомогою інтеграційного тестування наведена на рисунку 3.10.

```
it( title: 'check link position',  fn: () => {
  cy.get('[class="card-block"]').eq( index: 0).find( selector: '[class="hrefch"]').should( chainer: 'be.visible')
})
```

Рисунок 3.10 – Приклад реалізації тестового сценарію перевірки позиції посилання у програмному коді

Час проходження сценарію у UI-режимі склав 3 секунди.

Час проходження сценарію у headless-режимі склав 3 секунди.

Сценарій працює з першим блоком товару та шукає посилання саме усередині нього.

Наступним тестовим сценарієм є перевірка можливості відкриття сторінки з товаром за допомогою регресійного тестування (рис. 3.11).

```
it( title: 'click the link', fn: () => {
  cy.get('[class="card-block"]').eq( index: 0).find( selector: '[class="hrefch"]').should( chainer: 'be.visible').click();
  cy.get('[class="hrefch"]').should( chainer: 'not.exist')
})
```

Рисунок 3.11 – Приклад програмної реалізації тестового сценарію

Даний сценарій знаходить посилання, натискає на нього та перевіряє, що посилання вже не існує у DOM-дереві, що свідчить про вдалий перехід на нову сторінку.

Час проходження сценарію у UI-режимі склав 4 секунди.

Час проходження сценарію у headless-режимі склав 3 секунди.

Наступним тестовим сценарієм є перевірка можливості переходу між вкладками з товаром за допомогою модульного тестування (рис. 3.12).

```
it( title: 'check navigation button visibility', fn: () => {
  cy.get('[onclick="byCat(\'notebook\')"]').should( chainer: 'be.visible')
})
```

Рисунок 3.12 – Приклад програмної реалізації тестового сценарію перевірки на наявність навігаційних кнопок

Час проходження сценарію у UI-режимі склав 3 секунди.

Час проходження сценарію у headless-режимі склав 2 секунди.

Виділення помаранчевим кольором для даного сценарію використовується для роботи з лапками.

Наступний тестовий сценарій реалізує перевірку можливості переходу між категоріями з товаром за допомогою інтеграційного тестування, а саме перевіряє, що кнопка знаходиться усередині блоку з кнопками.

Метод «parent» перевіряє, що вищім ніж елемент кнопки у DOM-дереві є елемент блоку з кнопками (рис. 3.13).

Час проходження сценарію у UI-режимі склав 3 секунди.

Час проходження сценарію у headless-режимі склав 2 секунди.

Елемент «#» у даному методі відповідає за id елемента.

```
it( title: 'check button position', fn: () => {
  cy.get('#itemc').parent( selector: '[class="list-group"]')
})
```

Рисунок 3.13 – Приклад програмної реалізації тестового сценарію перевірки позиції навігаційної кнопки

Наступним сценарієм є перевірка можливості переходу між категоріями з товаром за допомогою регресійного тестування, а саме перевірка відкриття іншої сторінки з товаром (рис. 3.14).

```
it( title: 'open different page', fn: () => {
  cy.get('[class="hrefch"]').eq( index: 0).should( chainer: 'contain.text', value: 'Samsung galaxy s6')
  cy.get('[class="list-group"]').should( chainer: 'be.visible').find( selector: '[onclick="byCat(\'notebook\')"]').wait( ms: 2000).click( options: {force:true})
  cy.wait(3000).get( selector: '[class="hrefch"]').eq( index: 0).should( chainer: 'contain.text', value: 'Sony vaio i5')
  cy.get('[class="hrefch"]').eq( index: 0).should( chainer: 'not.contain.text', value: 'Samsung galaxy s6')
```

Рисунок 3.14 – Приклад програмної реалізації тестового сценарію перевірки відкриття іншої сторінки

Час проходження сценарію у UI-режимі склав 8 секунд.

Час проходження сценарію у headless-режимі склав 8 секунд.

Дана реалізація знаходить ім’я товару у першому блоці на базовій сторінці та після переходу на іншу перевіряє, що у першому блоці знаходиться інше ім’я.

Елементи wait та force допомагають вирішувати проблему не до кінця видимих елементів на сторінці.

Наступний тестовий сценарій перевіряє те, що тільки один фото-елемент товару у одному блоці є видимим, тобто реалізує перевірку правильності відображення фото з товаром за допомогою модульного тестування (рис. 3.15).

```
it( title: 'check only one photo visible', fn: () => {
  cy.get('[class="card h-100"]').eq( index: 0).find( selector: 'img').eq( index: 0).should( chainer: 'be.visible')
  cy.get('[class="card h-100"]').eq( index: 0).find( selector: 'img').eq( index: 1).should( chainer: 'not.exist')
})
```

Рисунок 3.15 – Приклад програмної реалізації перевірки кількості фото елементів у одному блоку з товаром

Даний скрипт перевіряє, що у першому блоку з товаром видимий тільки один фото-елемент, а другий навіть не існує у DOM-дереві.

Час проходження сценарію у UI-режимі склав 3 секунди.

Час проходження сценарію у headless-режимі склав 3 секунди.

Тестовий сценарій перевіряє те, що посилання з товаром відображає фото певного товару, тобто реалізує перевірку правильності відображення фото з товаром за допомогою інтеграційного тестування (рис. 3.16).

Рисунок 3.16 – Приклад програмної реалізації перевірки відповідності посилання та фото

Час проходження сценарію у UI-режимі склав 3 секунди.

Час проходження сценарію у headless-режимі склав 3 секунди.

Даний скрипт знаходить текст з іменем товару у першому блоці з товаром та порівнює його з іменем товару у CSS-селекторі першого блоку з товаром.

Наступний тестовий сценарій реалізує перевірку правильності відображення фото з товаром за допомогою регресійного тестування, а саме перевіряє те, що перший елемент з товаром відображається та не є пустим (рис. 3.17).

Рисунок 3.17 – Приклад програмної реалізації перевірки наповнення блоку з товаром

Час проходження сценарію у UI-режимі склав 3 секунди.

Час проходження сценарію у headless-режимі склав 2 секунди.

Наступний тестовий сценарій заходить на посилання з товаром та перевіряє, що кнопка додавання товару у корзину є видимою та може бути натиснутою і, таким чином, реалізує перевірку можливості додавання товару до корзини за допомогою модульного тестування (рис. 3.18).

```
it( title: 'check add to cart button state', fn: () => {
  cy.get('[class="card h-100"]').eq( index: 0).find( selector: '[class="hrefch"]').click()
  cy.get('.btn-success').should( chainer: 'be.visible')
  cy.get('.btn-success').should( chainer: 'not.be.disabled')
})
```

Рисунок 3.18 – Приклад програмної реалізації перевірки стану кнопки додавання товару до корзини

Час проходження сценарію у UI-режимі склав 5 секунд.

Час проходження сценарію у headless-режимі склав 4 секунди.

Наступний тестовий сценарій перевіряє, що кнопка додавання товару знаходиться усередині елемента з описом товару і таким чином реалізує перевірку можливості додавання товару до корзини за допомогою інтеграційного тестування (рис. 3.19).

```
it( title: 'check add to cart button position', fn: () => {
  cy.get('[class="card h-100"]').eq( index: 0).find( selector: '[class="hrefch"]').click()
  cy.get('#tbodyid').find( selector: '.btn-success').trigger( eventName: 'mouseenter')
})
```

Рисунок 3.19 – Приклад програмної реалізації перевірки позиції кнопки додавання товару до корзини

Час проходження сценарію у UI-режимі склав 4 секунди.

Час проходження сценарію у headless-режимі склав 4 секунди.

Метод «trigger» дозволяє навести курсор на знайдену кнопку.

Наступний тестовий сценарій перевіряє можливість додавання товару до корзини, натискаючи на кнопку додавання, потім перезавантажує сторінку та переходить до корзини і перевіряє наявність товару у ній (рис. 3.20).

Рисунок 3.20 – Приклад програмної реалізації перевірки можливості додавання товару до корзини

За допомогою даного сценарію реалізується перевірка можливості додавання товару до корзини через регресійне тестування.

Час проходження сценарію у UI-режимі склав 14 секунд.

Час проходження сценарію у headless-режимі склав 15 секунд.

Метод «scrollTo» наводить курсор на самий верх тестової сторінки.

Наступний тестовий сценарій перевіряє видимість усіх текстових полів заповнення замовлення на товар. Деякі з полів знаходяться за межами видимості при натисканні на кнопку замовлення, для виведення їх на екран використовується метод «scrollIntoView» (рис. 3.21).

Рисунок 3.21 – Приклад програмної реалізації тестового сценарію з перевірки видимості полів заповнення замовлення

Даний сценарій реалізує перевірку полів заповнення замовлення за допомогою модульного тестування.

Час проходження сценарію у UI-режимі склав 15 секунд.

Час проходження сценарію у headless-режимі склав 14 секунд.

Сценарій (рис. 3.22) перевіряє, що поля заповнення замовлення не тільки є видимими, але й знаходяться усередині одного діалогового вікна, тобто реалізується перевірка за допомогою інтеграційного тестування.

Рисунок 3.22 – Приклад програмної реалізації тестового сценарію з перевірки позиції полів заповнення замовлення

Час проходження сценарію у UI-режимі склав 18 секунд.

Час проходження сценарію у headless-режимі склав 15 секунд.

Логічним наступним тестовим сценарієм є перевірка можливості введення тексту у поля для заповнення замовлення. Для цього у Cypress використовується метод «type» (його аналогом є метод «fill», який використовується при необхідності вводу великого тексту за малий проміжок часу). Даний сценарій реалізує перевірку валідності полів заповнення замовлення за допомогою регресійного тестування.

Програмна реалізація даного методу наведена на рисунку 3.23.

Рисунок 3.23 – Приклад програмної реалізації тестового сценарію з перевірки введення тексту у поля для заповнення замовлення

Час проходження сценарію у UI-режимі склав 16 секунд.

Час проходження сценарію у headless-режимі склав 16 секунд.

Останнім тестовим сценарієм є перевірка того, що після вводу даних замовлення може бути відправлене та клієнт може отримати повідомлення про вдалу відправку, даний сценарій є ще одним прикладом реалізації регресійного тестування. (рис. 3.24).

```
it( title: 'click purchase button', fn: () => {
  cy.get('[class="card h-100"]').eq( index: 0).find( selector: '[class="hrefch"]').click()
  cy.get('#tbodyid').find( selector: '.btn-success').trigger( eventName: 'mouseenter').wait( ms: 2000).click( options: {force:true}).wait( ms: 2000)
  cy.reload()
  cy.wait(3000).scrollTo( position: 'top').get( selector: '[class="container"]') .find( selector: '[class="nav-link"]').contains( content: 'Cart').click()
  cy.wait(2000).get( selector: '[class="success"]').should( chainer: 'be.visible')
  cy.get('[data-target="#orderModal"]').click()
  cy.get('#name').type( text: 'Test')
  cy.get('#country').type( text: 'Test')
  cy.get('#city').type( text: 'Test')
  cy.get('#card').scrollIntoView().type( text: '111222333')
  cy.get('#month').scrollIntoView().type( text: 'August')
  cy.get('#year').scrollIntoView().type( text: '2021')
  cy.get('[onclick="purchaseOrder()"]').scrollIntoView().click()
  cy.get('[class="sa-icon sa-success animate"]').should( chainer: 'be.visible')
})
```

Рисунок 3.24 – Приклад програмної реалізації тестового сценарію з відправки замовлення та перевірки наявності повідомлення про успішну відправку

Час проходження сценарію у UI-режимі склав 16 секунд.

Час проходження сценарію у headless-режимі склав 15 секунд.

На рисунку 3.25 наведені результати проходження усіх наведених вище тестових сценаріїв у Cypress за командою «cypress open».

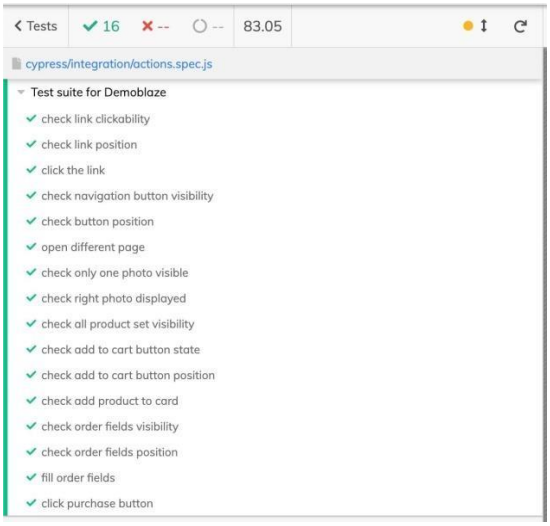

Рисунок 3.25 – Результати проходження усіх тестових сценаріїв у Cypress за командою «cypress open»

Час проходження сценаріїв у цьому режимі – 1 хвилина та 23 секунди. Порівняємо ці результати з запуском за командою «cypress run» (рис. 3.26).

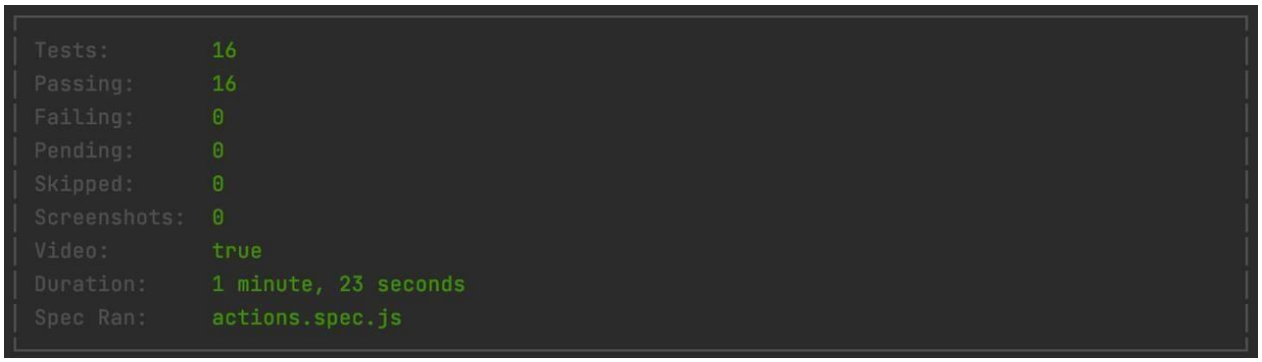

Рисунок 3.26 – Результати проходження усіх тестових сценаріїв у Cypress за командою «cypress run»

Час проходження є абсолютно однаковим, але проходження за допомогою команди «cypress open» є значно більш зручним з точки зору виправлення проблем у тестових сценаріях, бо дозволяє наглядно побачити, що відбувається на кожному кроці тесту.

3.4 Порівняльний аналіз досліджених методів вебтестування

Незважаючи на те, що обрані три методи вебтестування (модульного, регресійного та інтеграційного) мають мету виявляти та запобігати недоліків у тестовій базі, під час реалізації даних методів виділяють такі відмінності:

– модульне тестування є найкомпактніше, воно перевіряє тестований елемент на базовому рівні, як окремий модуль, що не пов’язаний з іншими;

– інтеграційне тестування проводить аналіз на рівень вище, воно перевіряє взаємодію модулів;

– регресійне тестування є найбільш значним, воно перевіряє тестовий елемент, як єдину систему згори донизу.

Для аналізу отриманих результатів стосовно проведення вебтестування обраними методами необхідно застосувати оцінювання за такими, наприклад, критеріями:

– час проходження сценарію;

– складність реалізації сценарію у програмному коді;

– ступінь універсальності реалізованого програмного коду для вирішення подібних прикладних задач.

Під час аналізу отриманих результатів за критерієм часу проходження тестування аналізується час проходження у UI-режимі, як більш популярному при реалізації тестових сценаріїв за допомогою Cypress.

Порівняльний аналіз досліджених методів вебтестування стосовно критерію часу (у секундах) наведений у таблиці 3.1.

Порівняльний аналіз досліджених методів вебтестування стосовно критерію складності реалізації наведений у таблиці 3.2.

Порівняльний аналіз досліджених методів вебтестування стосовно можливості повторного використання наведений у таблиці 3.3.

Таблиця 3.1 – Порівняний аналіз результатів проходження тестових сценаріїв щодо критерію час

	Модульне	**Інтеграційне**	**Регресійне**
Перевірка відкриття сторінки з товаром	4	3	4
Перевірка можливості переходу між категоріями з товаром	3	3	8
Перевірка вірності відображення фото	3	3	3
Перевірка можливості додавання товару до корзини	5	4	14
Перевірка полів заповнення замовлення	15	18	16

Таблиця 3.2 – Порівняний аналіз складності реалізації тестових сценаріїв у програмному коді

	Модульне	**Інтеграційне**	**Регресійне**
Перевірка відкриття сторінки з товаром	Низька	Низька	Середня
Перевірка можливості переходу між категоріями з товаром	Низька	Середня	Висока
Перевірка вірності відображення фото	Середня	Середня	Середня
Перевірка можливості додавання товару до корзини	Середня	Висока	Висока
Перевірка полів заповнення замовлення	Висока	Висока	Висока

Таблиця 3.3 – Порівняний аналіз можливості повторного використання реалізованих у програмному коді методів

	Модульне	**Інтеграційне**	**Регресійне**
Перевірка відкриття сторінки з товаром	Висока	Середня	Середня
Перевірка можливості переходу між категоріями з товаром	Висока	Висока	Низька
Перевірка вірності відображення фото	Середня	Середня	Висока
Перевірка можливості додавання товару до корзини	Висока	Низька	Низька
Перевірка полів заповнення замовлення	Низька	Низька	Низька

Відповідно до наведеного порівняльного аналізу можливо побачити, що тестові сценарії за регресійним методом, є найскладнішими у створенні, найдовшими у проходженні та з найменшою ймовірністю повторного

використання. Це пов’язано з особливостями реалізації тестових сценаріїв за допомогою регресійного методу.

Модульне та інтеграційне тестування реалізує тестову дію на рівні модулів та їх інтеграції відповідно, що є досить гнучким рішення з точки зору усіх трьох елементів критеріальної оцінки, регресійне тестування працює зі системою, як набором інтеграцій в цілому, тоді як кожна система унікальна, що свідчить про підвищену складність реалізації регресійного тестування та його велику популярність як методу, що дозволяє аналізувати систему в цілому.

Для остаточного аналізу отриманих результатів доцільно ввести тестову шкалу за кожним з критеріїв «від кращого до гіршого».

Для критерію проходження сценарію за часом бали для оцінки є такі:

– для модульного тестування від 2 секунд до 4 секунд – 1 бал, 5 секунд та більше – 0,5 балів;

– для інтеграційного тестування 3 секунди або 4 секунди – 1 бал, 5 секунд та більше – 0,5 балів;

– для регресійного тестування від 3 секунд до 10 секунд – 1 бал, 11 секунд та більше – 0,5 балів.

Для критерію складності реалізації тестових сценаріїв у програмному коді бали для оцінки є такі:

– для модульного та інтеграційного тестувань: низька – 2 бали, середня – 1 бал, висока – 0 балів;

– для регресійного тестування: низька – 3 бали, середня – 2 бали, висока – 1 бал;

Для критерію можливості повторного використання реалізованих тестових сценаріїв у інших проєктах бали для оцінки є такі:

– для модульного та інтеграційного тестувань: висока – 2 бали, середня – 1 бал, низька – 0 балів;

– для регресійного тестування: висока – 3 бали, середня – 2 бали, низька – 0,5 балів.

Бали для оцінювання результатів наведені відповідно до особливостей реалізації кожного з видів тестування.

Результати оцінювання за балами наведені у таблиці 3.4.

Таблиця 3.4 – Результати оцінювання тестових результатів за обраними критеріями

	Модульне	**Інтеграційне**	**Регресійне**
Перевірка відкриття сторінки з товаром	1+2+2 = = 5 балів	1+2+1 = = 4 бали	1+2+2 = = 5 балів
Перевірка можливості переходу між категоріями з товаром	1+2+2 = = 5 балів	1+1+2 = = 4 бали	1+1+0,5 = = 2,5 бали
Перевірка вірності відображення фото	1+1+1 = = 3 бали	1+1+1 = = 3 бали	1+2+3 = = 6 балів
Перевірка можливості додавання товару до корзини	0,5+1+2 = = 3,5 балів	1+0+0 = = 1 бал	0+1+0,5 = = 1,5 бали
Перевірка полів заповнення замовлення	0,5+0+0 = = 0,5 балів	0,5+0+0 = = 0,5 балів	0+1+0,5 = = 1,5 бали

Відповідно до наведених результатів оцінювання найвдалішими методами з точки зору комбінації усіх трьох критеріїв для тестування обраних задач були наступні:

– перевірка відкриття сторінки з товаром – модульне та регресійне тестування;

– перевірка можливості переходу між категоріями з товаром – модульне тестування;

– перевірка вірності відображення фото – регресійне тестування;

– перевірка можливості додавання товару до корзини – модульне тестування;

– перевірка полів заповнення замовлення – регресійне тестування.

Отже, найперспективнішими для продовження процесу тестування на обраній тестовій базі є комбінація модульного та регресійного тестувань, вона дозволить розглянути систему на мікро- та макрорівнях.

3.5 Перспективи подальшої роботи

Автоматизоване тестування програмних і апаратних застосунків та їх сумісності зменшує витрати на розроблення і оновлення системи за допомогою швидкості виконання, можливості повторного використання, охоплення різноманітних кейсів, високої точності без людської фактору [56].

Проте, на даний момент, є проблема залучення спеціалістів ручного тестування у сферу підтримки автоматизованих тестів. Створюються спеціальні допоміжні інструменти – фреймворки автоматизованого тестування, що дають можливість зосередитись на виконанні конкретної задачі без розроблення допоміжного та службового інструментарію [57].

Отже, на даний час розвиток технології йде дуже швидкими темпами, а так, як будь-яка технологія потребує тестування, то подальша робота з розвитку даної теми завжди є та буде актуальною.

Вивчення та впровадження видів автоматизованого тестування, розвиток мережі фреймворків для тестування, підвищення компетентності спеціалістів-тестувальників, насамперед, щодо автоматизованого тестування, буде сприяти подальшому розвитку технологій у вебдіяльності та у будь-якій сфері людського життя.

ВИСНОВКИ

У даній кваліфікаційній роботі розроблено проєкт автоматизованого тестування онлайн-магазину.

У процесі аналізу теоретичних питань визначенні ключові моменти вебтестування, до яких слід віднести:

– визначення якості програмного забезпечення, а саме визначення того, наскільки програмне забезпечення, що тестується, здатне відповідати вимогам остаточних користувачів;

– верифікація, а саме процес визначення відповідності розроблення програмного забезпечення до планів розробки;

– валідація, а саме визначення відповідності можливостей програмного забезпечення до очікувань остаточних користувачів.

Визначено найперспективніший вид вебтестування – автоматизоване тестування, розглянуто його переваги та недоліки, а також допоміжні застосунки для його реалізації (GIT, який відповідає за колективну працю програмістів над одним проєктом).

Розглянуто сучасні методи вебтестування (модульне, інтеграційне, регресійне та ін.) та визначено можливі складнощі у їх реалізації.

Після аналізу сучасних видів вебтестування, які можуть бути використані у подальшій практичній діяльності, проведено моделювання процесу вебтестування за допомогою стандарту IDEF0. Процес моделювання методологією IDEF0 є самим вдалим вирішенням для формування наглядної взаємодії не тільки між зацікавленими сторонами проєкту, але й між етапами реалізації самого процесу вебтестування.

Для програмної реалізації та подальшого дослідження обрано три види тестування (інтеграційне, регресійне та модульне), проаналізовано кожний з них щодо доцільності їх реалізації у контексті вебтестування.

Аналіз непередбачуваних ситуацій при реалізації кожного з обраних методів вебтестування показав, що у процесі впровадження можуть бути виявлені не тільки слабкі, але й непомічені раніше можливості для розвитку застосунку, що свідчить про те, що при формування тест-кейсів потрібно концентруватися не тільки на негативному тестуванні (процесі пошуку слабких місць), але й на пошуку нових можливостей для покращення діяльності застосунку.

Як предметну область для дослідження обраних методів вебтестування взято тестовий проєкт на базі онлайн-магазину Demoblaze. Онлайн торгівля є одним з найпопулярніших видів вебзастосунків у сучасному світі, отже, реалізація тестового проєкту саме на цій тестовій базі є доцільною.

Як інструментарій для реалізації тестового проєкту обрано базову мову для розроблення та тестування вебзастосунків – JavaScript (у комплекті з базовим для неї середовищем розроблення WebStorm) та фреймворк для запуску тестів Cypress. Даний фреймворк (допоміжний застосунок) виділяється простим програмним кодом та широкими можливостями для виправлення помилок у тест-кейсах за допомогою вікна з етапами проходження тест-кейсу.

Отримані результати порівняно між собою за допомогою шкал від кращого до гіршого значення критеріїв.

Зроблено висновок щодо доцільності окремих методів вебтестування для вирішення конкретних прикладних задач предметної області:

– перевірка відкриття сторінки з товаром – модульне та регресійне тестування;

– перевірка можливості переходу між категоріями з товаром – модульне тестування;

– перевірка вірності відображення фото – регресійне тестування;

– перевірка можливості додавання товару до корзини – модульне тестування;

– перевірка полів заповнення замовлення – регресійне тестування.

Найперспективнішими для продовження процесу тестування на обраній тестовій базі є комбінація модульного та регресійного тестувань.

Визначено перспективи подальшої роботи.

Результати роботи апробовано у вигляді 2 тез доповідей під час Міжнародних науково-практичних конференцій «PROBLEMS OF MODERN SCIENCE AND PRACTICE» [58] та «SCIENCE, THEORY AND PRACTICE» [59].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. 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.

2. 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.

3. Yousef Ibrahim Daradkeh, and Iryna Tvoroshenko (2020) Application of an Improved Formal Model of the Hybrid Development of Ontologies in Complex Information Systems, *Applied Sciences*, 10(19). p. 6777.

4. Информационная технология. Оценка программной продукции. Характеристики качества и руководства по их применению. (ISO/IEC 9126:2001) ГОСТ Р ИСО/МЭК 9126-93 [Чинний від 1993-11-28] М.: ИПК Издательство стандартов, 2004. 12 с.

5. IEEE Standard for Software and System Test Documentation. IEEE 82-2008 [Чинний від 2008-03-08] NY.: The Institute of Electrical and Electronics Engineers, 2008. 132 p.

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

7. Daradkeh, Y.I., Gorokhovatskyi, V., Tvoroshenko, I., Gadetska, S., and Al-Dhaifallah, M. (2021) Methods of Classification of Images on the Basis of the Values of Statistical Distributions for the Composition of Structural Description Components, *IEEE Access*, 9, pp. 92964-92973.

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

9. 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.

10. Ромашин Р.І. (2015) Веб безпека.

11. Sankar R. Burpsuite – A Beginner’s Guide For Web Application Security or Penetration Testing. URL: https://kalilinuxtutorials.com/ burpsuite/ (дата звернення 03.10.2021).

12. Tvoroshenko Irina, Ahmad M. Ayaz, Mustafa Syed Khalid, Lyashenko Vyacheslav, and Alharbi Adel R. (2020) Modification of Models Intensive Development Ontologies by Fuzzy Logic, *International Journal of Emerging Trends in Engineering Research*, 8(3), pp. 939-944.

13. M. Ayaz Ahmad, Irina Tvoroshenko, Jalal Hasan Baker, Liubov Kochura, Vyacheslav Lyashenko (2020) Interactive Geoinformation Three-Dimensional Model of a Landscape Park Using Geoinformatics Tools, *International Journal on Advanced Science, Engineering and Information Technology*, 10(5), pp. 2005-2013.

14. M. Ayaz Ahmad, Irina Tvoroshenko, Jalal Hasan Baker, and Vyacheslav Lyashenko (2019) Modeling the Structure of Intellectual Means of Decision-Making Using a System-Oriented NFO Approach, *International Journal of Emerging Trends in Engineering Research*, 7(11), pp. 460-465.

15. ORDINATUS. 5 инструментов непрерывной интеграции. URL: http://ordinatus.ru/5-instrumentov-nepreryvnoj-integracii (дата звернення 15.09.2021).

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

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

18. Automated GUI Testing: A Step by Step Guide. URL: https://www.ukad-group.com/blog/automated-gui-testing-a-step-by-step-guide/ (дата звернення 03.10.2021).

19. Тестування на проникнення. URL: https://uk.myservername.com/beginners-guide-web-application-penetration-testing (дата звернення 03.10.2021).

20. Gorokhovatskyi, V., Rusakova, N., and Tvoroshenko, I. (2020) The application of image analysis methods and predicate logic in applied problems of magnetic monitoring, *Telecommunications and Radio Engineering*, 79(20), pp. 1801-1811.

21. 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.

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

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

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

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

26. Tvoroshenko, I., & Koriakin, I. (2021). Analysis of methods for detecting and classifying the likeness of human features.

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

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

29. Слободян Р (2020) Автоматизоване тестування веб-додатків шляхом взаємодії з користувацьким інтерфейсом.

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

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

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

33. Дубовік Я.С. (2019) Необхідність використання сучасних технологій моделювання бізнес-процесів.

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

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

36. Регресійне тестування. URL:https://qalearning.com.ua/theory/lectures/material/regression-testing-crossbrowser/ (дата звернення 02.10.2021).

37. Гороховатський В.О., Творошенко І.С. (2021). Методи інтелектуального аналізу та оброблення даних: навч. посібник.

38. Модульне тестування. URL:https://msn.khnu.km.ua/pluginfile.php/208290/mod_resource/content/2/%D0%9B%D0%A0%20%E2%84%964.pdf. (дата звернення 10.09.2021).

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

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

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

42. Творошенко І.С. (2021). Технології прийняття рішень в інформаційних системах: навч. посібник. *Харків: ХНУРЕ*.

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

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

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

46. 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.

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

48. 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.

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

50. 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.

51. WebStorm. URL: https://uk.wikipedia.org/wiki/WebStorm (дата звернення 19.09.2021).

52. JavaScript. URL: https://uk.wikipedia.org/wiki/JavaScript (дата звернення 20.09.2021).

53. BDD. URL: https://ru.wikipedia.org/wiki/BDD (дата звернення 21.09.2021).

54. Cypress. URL: https://bizzapps.ru/p/cypress/ (дата звернення 25.09.2021).

55. CA Erwin Process Modeller. URL: https://stud.com.ua/77235/informatika/erwin_data_modeler_erwin (дата звернення 27.09.2021).

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

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

58. Tvoroshenko I., and Maksimenko H. (2021) To the question of analysis of existing mechanisms of web application testing, *Abstracts of I International scientific- practical conference «Problems of modern science and practice» (September 21-24, 2021). Boston, USA*, pp. 403-409.

59. Tvoroshenko I., and Maksimenko H. (2021) Research of regression and modular testing of web applications, *Abstracts of IV International scientific-practical conference «Science, theory and practice» (October 12-15, 2021). Tokyo, Japan*, pp. 406-411.