ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ, УПРАВЛІННЯ, ПРАВА ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Пояснювальна записка

до кваліфікаційної роботи на здобуття ступеня вищої освіти магістр

на тему: «**Дослідження методів тестування програмного забезпечення «чорного ящика» (Black-Box Testing Techniques)»**

Виконав: здобувач вищої освіти
за освітньо-професійною програмою
Інформаційні управляючі системи та
технології спеціальності 126 Інформаційні
системи та технології ступеня вищої
освіти магістр
групи 126ІСТ_мд_21
Даниленко А.А.
Керівник: Флегантов Л.О.
Рецензент: Петраш Р.В.

Полтава – 2023 року

ВСТУП

Актуальність теми дослідження методів тестування програмного забезпечення ПЗ «чорного ящика» полягає у наступному: збільшення обсягів даних викликає потребу у тестуванні внутрішньої логіки програм та їх інтеграції; сучасні програми стають складнішими, що ускладнює процес тестування та збільшує актуальність методів «чорного ящика»; тестування «чорного ящика» відповідає вимогам користувачів щодо надійності та безпеки ПЗ; швидке розширення ринку та поява нових технологій вимагають адаптивних методів тестування; методи «чорного ящика» забезпечують швидше та ефективніше тестування без глибоких знань внутрішньої структури ПЗ; потреба в автоматизації через зростання обсягу та складності ПЗ; методи «чорного ящика» підтримують швидке та ефективне тестування в гнучких та DevOps середовищах; виклики тестування нових платформ вимагають гнучких методів тестування. Ці фактори підтверджують важливість та актуальність дослідження методів тестування ПЗ «чорного ящика» у сучасних умовах.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконана у відповідності до науково-дослідної ініціативної теми «Організаційно-методологічні аспекти впровадження інформаційно-комунікаційних систем і технологій в управлінні діяльністю сучасних організацій та підприємств за умов переходу до цифрової економіки» ДРН 0117U003099.

Мета роботи: оцінити ефективність та застосування методів тестування «чорного ящика» у сучасному контексті розробки програмного забезпечення.

Завдання роботи:

- дослідити еволюцію у методах тестування «чорного ящика»;

- оцінити сучасні методики тестування з урахуванням їх ефективності;

- розробити автоматизовані тестові сценарії з використанням методів «чорного ящика» для різних мов програмування (Java, Python, JavaScript) та їх комбінацій;

- провести порівняльний аналіз розроблених сценаріїв автоматизованого тестування на основі визначених метрик ефективності;

- сформулювати рекомендації щодо вдосконалення та оптимізації процесу тестування ПЗ з використанням методів «чорного ящика».

Об'єкт дослідження: методи тестування програмного забезпечення «чорного ящика».

Предмет дослідження: теоретичний аналіз та практичне застосування методів тестування програмного забезпечення «чорного ящика».

Методи дослідження:

- аналіз літературних джерел щодо методів тестування ПЗ «чорного ящика» з метою з'ясування основних понять, принципів та класифікації цих методів;

- порівняння існуючих технік та інструментів, що використовуються при тестуванні «чорного ящика»;

- аналіз прикладів використання методів тестування «чорного ящика» у реальних проєктах;

- порівняння різних методів тестування «чорного ящика» з точки зору їх особливостей, сфер застосування, переваг та недоліків;

- проведення експерименту з використанням різних методів тестування програмного забезпечення;

- аналіз отриманих результатів та формування висновків щодо ефективності та придатності методів тестування «чорного ящика» Розроблення рекомендацій для їх використання.

Інформаційна база: стандарти та вимоги, що перевіряються під час тестування: загальні категорії стандартів та вимог (законодавчі вимоги – закони та правила, які обмежують або визначають, як повинно функціонувати програмне забезпечення, наприклад, щодо конфіденційності даних, захисту споживачів тощо); стандарти безпеки (вимоги до забезпечення безпеки даних та інформаційних систем, ISO 27001); стандарти якості ПЗ (вимоги до якості та надійності ПЗ, ISO 9001); стандарти інтерфейсів та сумісності (вимоги до

сумісності ПЗ з іншими системами та інтерфейсами, стандарти W3C для вебсайтів); стандарти доступності (вимоги до доступності для людей з обмеженими можливостями, які регулюють, як ПЗ повинно бути доступним для всіх користувачів); стандарти відкритості та інтероперабельності (вимоги до відкритості та можливості ПЗ працювати разом з іншими продуктами та системами); галузеві стандарти (вимоги, специфічні для певних галузей); офіційна документація ПЗ, що використовується у тестуванні; науково-технічна література, статті, аналітичні джерела мережі інтернет з питань тестування ПЗ.

Елементи наукової новизни: узагальнено систему класифікації методів тестування «чорного ящика», що дозволяє ефективніше порівнювати та вибирати методики для конкретних проєктів; здійснено систематизацію метрик оцінки ефективності тестування, що дозволяє отримувати більш об'єктивну та всебічну картину результатів тестування програмного забезпечення.

Практичне значення: робота спрямована на підвищення якості та надійності програмного забезпечення, зменшення кількості помилок та невиявлених проблєм під час розробки програмного забезпечення; запропоновані рекомендації щодо вибору методів та інструментів для тестування щодо ефективного планування та виконання тестування.

Апробація результатів дослідження. За результатами проведеного дослідження опубліковано тези доповідей: Даниленко А.А. Програмне забезпечення тестування «чорного ящика». *Матеріали XX щорічного міждисциплінарного семінару «Студентські роботи за науковою тематикою кафедри інформаційних систем та технологій ННІ ЕУП та ІТ ПДАУ»*, 29 листопада 2023 року. Полтава, 2023. С. 64-65.

Структура та обсяг кваліфікаційної роботи. Робота складається зі вступу, трьох розділів та висновків. Основний текст роботи викладений на 78 сторінках, містить 26 таблиці, 4 рисунки. Список використаних джерел налічує 69 найменувань.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ЗАСАДИ ТА МЕТОДОЛОГІЯ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Огляд методів тестування «чорного ящика»

Загальна теорія та методи тестування ПЗ, включно з методами тестуванням «чорного ящика» (Black-Box Testing Techniques), розглядаються у багатьох науково-технічних публікаціях. До найбільш авторитетних джерел належать праці IT-спеціалістів, які за даними пошукової системи Google є найбільш відомими в галузі тестування ПЗ, зокрема, це: I. Sommerville [1], C. Kaner, J. Falk, & H. Nguyen [2], B. Beizer [3], ,G. J. Myers, C. Sandler & T. Badgett [4], R. Binder [5], J. A. Whittaker, G. Thom [6], P. C. Jorgensen [7] та ін.

Різні аспекти теорії та практики тестування ПЗ описували: Gerald M. Weinberg [8], Cem Kaner, James Marcus Bach, Bret Pettichord [9], Cem Kaner, Sowmya Padmanabhan, Douglas Hoffman [10], Ron Patton [11], Eric S. Raymond [12], Rafael Aguayo [13], Lisa Crispin, Janet Gregory [14, 15], Scott Berkun [16], Tom Kelley, Jonathan Littman [17], Edwards Deming [18], Philip B. Crosby [19, 20], Tom Gilb [21], James A Whittaker, Jason Arbon, Jeff Carollo [22], Lee Copeland [23], Rex Black [24], Daniel Kahneman [25], Tim Riley, Adam Goucher [26], Elisabeth Hendrickson [27], Robert V. Binder [28], James A. Whittaker [29] та інші.

Кожне з розглянутих джерел пропонує свій погляд на тестування «чорного ящика», від загальної теорії до конкретних практичних порад і методів. Важливо, що всі вони підкреслюють важливість цього методу для забезпечення якості ПЗ, зокрема його здатність виявляти помилки та недоліки, які можуть не бути очевидним. Поняття про методи тестування «чорного ящика» у різних джерелах можна узагальнити наступним чином: тестування «чорного ящика» зосереджене на зовнішній поведінці програми, не звертаючись до деталей її внутрішньої структури чи реалізації; тестувальники аналізують вхідні дані та очікувані результати, не враховуючи внутрішній механізм програми; мета застосування –

перевірити, чи відповідають вихідні дані програми очікуваним результатам, що важливо для забезпечення її коректної роботи.

У табл. 1.1 узагальнена інформація про існуючі підходи до визначення поняття про методи тестування ПЗ «чорного ящика».

Таблиця 1.1 – Визначення методу тестування «чорного ящика» у різних джерелах та їх особливості

Джерело	Визначення методу тестування «чорного ящика»	Особливості
[1]	Тестування на основі зовнішньої видимої поведінки програми без знання внутрішньої структури	Порівняння з тестуванням «білого ящика», акцент на функціональності
[2]	Тестування, де програма розглядається як «чорний ящик», зосередження на вхідних даних та виходах	Приклади методів тестування, акцент на взаємодії з програмою та функціональних вимогах
[3]	Фокус на функціональному аспекті програми без знання її внутрішньої структури	Практичне застосування, детальний розгляд технік тестування
[4]	Тестування з акцентом на зовнішній поведінці програми	Важливість відповідності програми специфікаціям та очікуванням користувачів
[5]	Тестування об’єктно-орієнтованих систем, зосередження на зовнішній поведінці без внутрішніх деталей	Застосування методів «чорного ящика» для тестування функціональності об’єктно-орієнтованих систем
[6]	Тестування програми як «чорного ящика» з оцінкою зовнішньої функціональності	Виявлення дефектів, поради щодо використання методів тестування для забезпечення якості
[7]	Фокус на зовнішньому видимому поведінці програми	Глибокий аналіз методів тестування, значення «чорного ящика» для виявлення помилок

Опис методів тестування «чорного ящика» у різних джерелах має свої особливості, а саме: порівняння тестування «чорного» та «білого» ящика, акцент на функціональності та відповідності специфікаціям [1]; опис тестування «чорного ящика» як способу взаємодії з програмою на основі її поведінки, приклади конкретних методів [2]; підкреслення практичного застосування методів «чорного ящика», акцент на важливості переконання в заданому

функціоналі [3]; акцент на аналізі зовнішньої поведінки програми, важливість відповідності програми специфікаціям [4]; зосередження на об'єктно-орієнтованих системах, використання «чорного ящика» для перевірки їх функціональності [5]; підхід до тестування як оцінки зовнішньої функціональності, важливість для виявлення дефектів ПЗ [6]; глибокий аналіз методів «чорного ящика», підхід для забезпечення якості ПЗ [7].

Про актуальність методів тестування ПЗ свідчить динаміка популярності пошукових запитів «Testing Techniques» за даними вебсервісу Google Trends (рис. 1.1) [30].

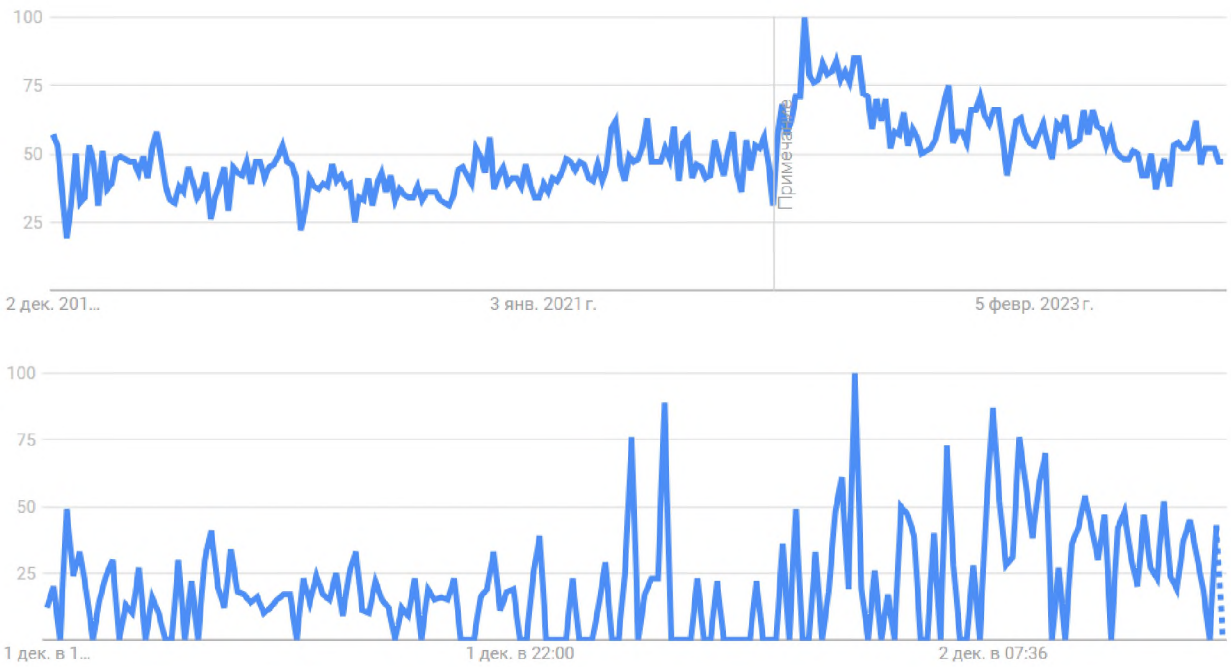

Рисунок 1.1 – Динаміка популярності пошукового запиту «Testing Techniques» у світі за останні 5 років та на початок грудня 2023 року

Динаміка популярності запитів «Testing Techniques» та «тестування ПЗ» в Україні за останні 5 років представлена на графіках (рис. 1.2).

Числа на графіках (рис. 1.1, 1.2) означають рівень інтересу до теми запиту: 100 балів означають найвищий рівень популярності запиту, 50 – рівень популярності, удвічі менший у порівнянні з першим випадком. 0 балів означає

регіон, за яким недостатньо даних про запит, що розглядається. Графіки показують, що в Україні та в усьому світі спостерігається сталий інтерес до методів тестування ПЗ.

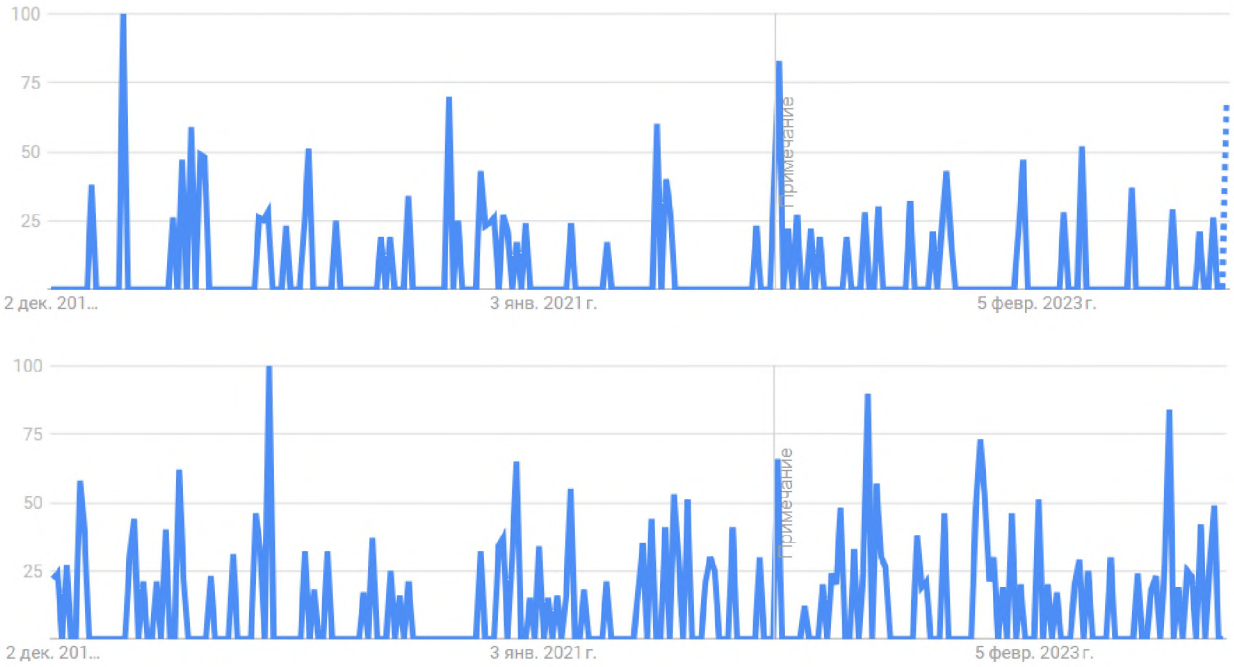

Рисунок 1.2 – Динаміка популярності запитів «Testing Techniques» та «тестування ПЗ» в Україні за останні 5 років

Показовими також є дані Google Trends щодо географічного розподілу популярності пошукового запиту Testing Techniques станом на кінець листопада – початок грудня 2023 року (рис. 1.3).

Популярність запиту Testing Techniques по регіонах світу (рис. 1.3) показує, де найчастіше здійснювався пошук за цим запитом за певний період відносно загальної кількості пошукових запитів. Згідно методики Google Trends, запитам надаються бали від 0 до 100, де 100 балів отримує регіон з найбільшою часткою популярності запиту, 50 балів – регіон, де рівень популярності запиту вдвічі нижчий, ніж у першому. 0 балів означає регіон, за яким недостатньо даних про запит, що розглядається.

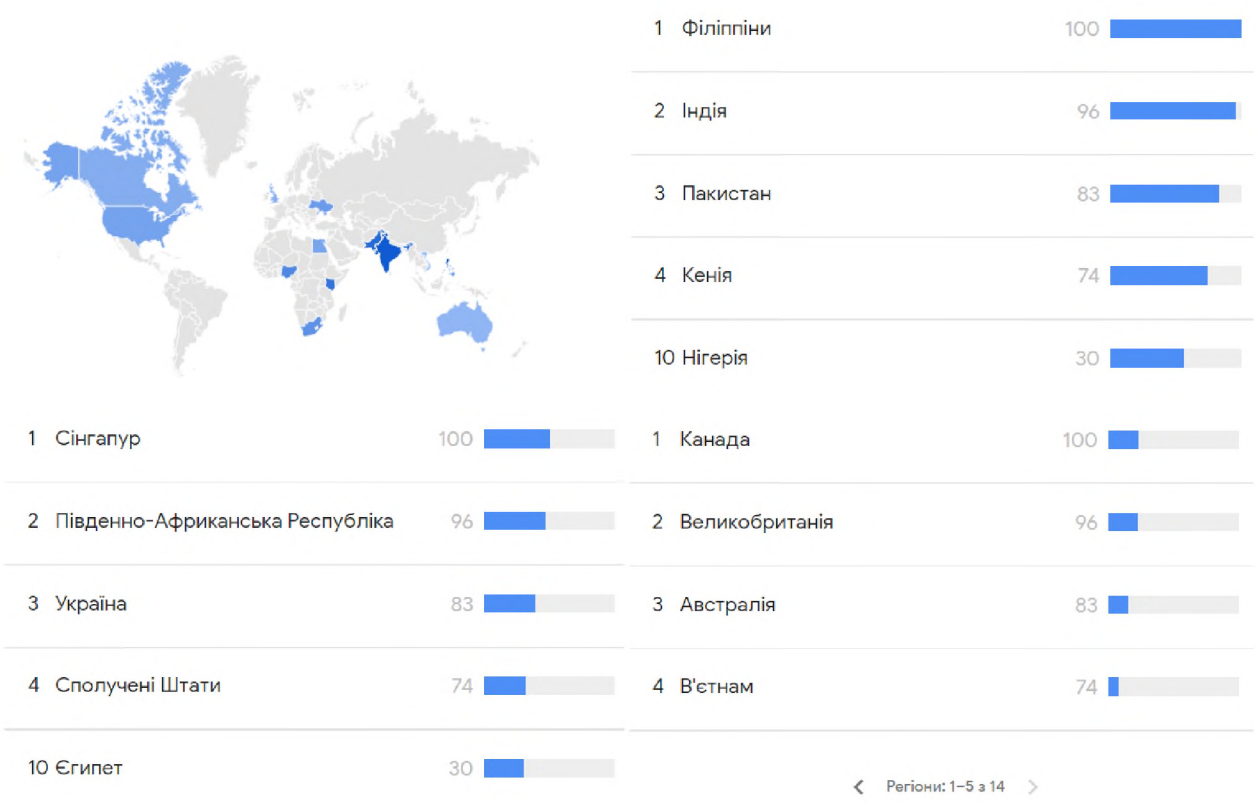

Рисунок 1.3 – Популярність пошукового запиту Testing Techniques за регіонами світу на початок грудня 2023 року за даними Google Trends

Чим більше балів, тим вища частка відповідних запитів від загальної кількості усіх запитів, а не їхня абсолютна кількість. Тому, наприклад, маленькій країні, де запити Testing Techniques становлять 80% від усіх запитів, буде надано вдвічі більше балів, ніж великий, де лише 40% усіх запитів містять це слово.

1.2 Етапи та види тестування

Базовим поняттям у тестуванні ПЗ є «життєвий цикл тестування програмного забезпечення» (Software Testing Life Cycle, STLC) [31], представлений на діаграмі (рис. 1.4).

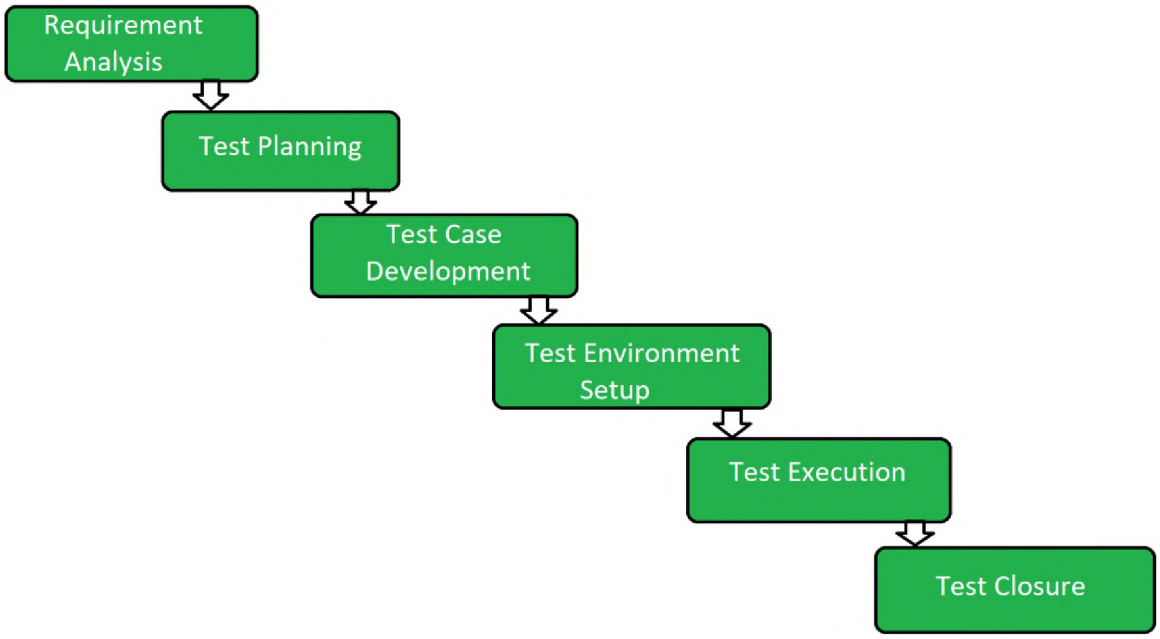

Рисунок 1.4 – Життєвий цикл тестування програмного забезпечення [32]

Розглянемо зміст основних етапів STLC [33]:

1. Аналіз вимог – дослідження та аналіз вимог для ідентифікації тестових сценаріїв;

2. Планування тестування – визначення стратегії, ресурсів, обмежень та графіку тестування;

3. Розробка тестових випадків – створення та перевірка тестових випадків і сценаріїв;

4. Налаштування тестового середовища – підготовка необхідного програмного та апаратного забезпечення;

5. Виконання тестування – проведення тестів згідно з планом та тестовими випадками;

6. Закриття тестового циклу – аналіз результатів, підготовка звітності та висновків.

На основі аналізу використаних джерел можна виокремити основні види тестування ПЗ, окреслити їх відмінності, та основні етапи.

Докладний огляд методів тестування представлений на діаграмі [34].

Функціональне тестування – вид тестування ПЗ, що оцінює функціональність програми та переконується, що вона виконує свої функції відповідно до вимог. Його основні етапи:

- визначення функціональних вимог до програми;
- створення тестових сценаріїв, що перевіряють кожну функцію;
- виконання тестів та порівняння результатів з очікуваними.

Тестування інтерфейсу користувача (UI тестування) – перевіряє, як користувачі взаємодіють з програмою через графічний інтерфейс. Включає такі етапи:

- тестування навігації та взаємодії з елементами інтерфейсу;
- перевірка вигляду та розміщення елементів;
- введення та перевірка коректності введених даних.

Тестування продуктивності – визначає, як програма працює під навантаженням та чи задовольняє вимогам щодо продуктивності. Основні етапи:

- створення сценаріїв, що моделюють навантаження на програму;
- вимірювання часу відгуку та швидкості програми під час виконання сценаріїв;
- аналіз результатів та порівняння з вимогами.

Тестування безпеки – перевіряє програму на вразливості та забезпечує захист від атак та несанкціонованого доступу. Основні етапи:

- виявлення потенційних загроз та вразливостей;
- проведення атак на програму для виявлення слабких місць;
- виправлення та валідація виявлених проблєм.

Тестування відмовостійкості – перевіряє, як програма поводиться в умовах негативних впливів, таких як відмови обладнання чи програмні помилки. Основні етапи:

- створення сценаріїв з відмовами для тестування реакції програми;
- перевірка відновлення та стійкості програми під час відмов

Ці види тестування є лише деякими з можливих підходів та методів, які використовуються для забезпечення якості ПЗ. Кожен х них має свої власні

підходи та процеси, які використовуються для виявлення помилок та вдосконалення програмного продукту.

Дещо інший підхід до видів тестування викладений у [35]. Він використовує ознаки, за якими класифікуються види тестування ПЗ [36, 37].

За об'єктом тестування виділяють наступні види: функціональне тестування (functional testing); дослідницьке тестування (exploratory testing); тестування продуктивності (performance testing); навантажувальне тестування (load testing); димне тестування (smoke testing); стрес-тестування (stress testing); тестування стабільності (stability / endurance / soak testing); тестування зручності використання (usability testing); тестування інтерфейсу користувача (ui testing); тестування безпеки (security testing); тестування локалізації (localization testing); тестування сумісності (compatibility testing).

За цілями тестування виділяють: функціональне тестування (functional); нефункціональне тестування (non-functional); тестування пов'язане зі змінами.

За знанням щодо системи виділяють: тестування чорної скриньки (black box); тестування білої скриньки (white box); тестування сірої скриньки (gray box).

За ступенем автоматизації виділяють: ручне тестування (manual testing); автоматизоване тестування (automated testing); напівавтоматизоване тестування (semiautomated testing).

За ступенем ізольованості компонентів виділяють: компонентне (модульне) тестування (component / unit testing); інтеграційне тестування (integration testing); системне тестування (system / end-to-end testing).

За часом проведення тестування виділяють: альфа-тестування (alpha testing); тестування нової функціональності (new feature testing); регресійне тестування (regression testing); тестування при здачі (acceptance testing); бета-тестування (beta testing).

За ознакою позитивності сценаріїв: позитивне тестування (positive testing); негативне тестування (negative testing).

За ступенем підготовленості до тестування: тестування за документацією (formal testing); Ad Hoc Testing (інтуїтивне) тестування (ad hoc testing).

Наочне уявлення про таку класифікацію надає діаграма (mind-map) [34]. Методи тестування «чорного ящика» мають відповідність усім методам тестування, що були розглянуті вище.

Функціональне тестування. Цей метод тестування «чорного ящика» зосереджений на перевірці функціональних можливостей програми, не розглядаючи її внутрішню структуру Функціональні тестові сценарії виконуються на основі вимог та очікуваних результатів, які також визначаються без врахування деталей реалізації програми.

Тестування інтерфейсу користувача (UI тестування) також є методом тестування «чорного ящика», оскільки воно спрямоване на перевірку того, як користувачі взаємодіють з програмою через інтерфейс, не вдаючись до внутрішнього коду. Тестування продуктивності. Хоча тестування продуктивності зазвичай оцінює швидкість та відмовостійкість програми під навантаженням, воно також може включати тестування функціональності під навантаженням, що відповідає концепції «чорного ящика».

Тестування безпеки також може бути розглянуте як метод тестування «чорного ящика», оскільки воно здійснюється з точки зору зовнішнього (стороннього) користувача, не вдаючись до внутрішніх деталей безпеки програми.

Тестування відмовостійкості перевіряє, як програма поводиться під час відмов або негативних впливів, і це також може бути частиною загального тестування «чорного ящика», якщо воно включає тестування зовнішнього впливу на програму. Усі ці методи тестування «чорного ящика» базуються на тестуванні програми з точки зору зовнішнього користувача або зовнішнього спостерігача, і вони дозволяють перевіряти функціональність, продуктивність, безпеку та інші аспекти ПЗ, не зважаючи на її внутрішню реалізацію.

У табл. 1.2 узагальнено класифікацію видів тестування за різними ознаками (категоріями).

Таблиця 1.2 – Класифікація видів тестування

Категорія тестування	Види тестування	Опис
За об'єктом тестування	Функціональне тестування, дослідницьке тестування, тестування продуктивності, навантажувальне тестування, димне тестування, стрес-тестування, тестування стабільності, тестування зручності використання, тестування інтерфейсу користувача, тестування безпеки, тестування локалізації, тестування сумісності.	Різні аспекти та функції ПЗ, що вимагають перевірки.
За цілями тестування	Функціональне тестування, нефункціональне тестування, тестування, пов'язане зі змінами.	Фокус на конкретних аспектах функціональності або характеристиках ПЗ.
За знанням щодо системи	Тестування чорної скриньки, тестування білої скриньки, тестування сірої скриньки.	Різні рівні знань про внутрішню структуру ПЗ для тестування.
За ступенем автоматизації	Ручне тестування, автоматизоване тестування, напівавтоматизоване тестування.	Використання ручних або автоматизованих методів тестування.
За ступенем ізольованості компонентів	Компонентне (модульне) тестування, інтеграційне тестування, системне тестування.	Тестування окремих частин ПЗ або цілісних систем.
За часом проведення тестування	Альфа-тестування, бета-тестування.	Різні етапи розробки ПЗ, від внутрішнього тестування до публічного.
За ознакою позитивності сценаріїв	Позитивне тестування, негативне тестування.	Тестування на відповідність або невідповідність очікуванням
За ступенем підготовленості до тестування	Тестування за документацією, Ad Hoc (інтуїтивне) тестування.	Від строго формалізованого до спонтанного тестування.

У табл. 1.3 відображені ключові етапи та дії основних видів тестування: функціонального, UI, продуктивності, безпеки та відмовостійкості.

Таблиця 1.3 – Етапи основних видів тестування

Етап тестування	Функціональне тестування	Тестування інтерфейсу користувача (UI)	Тестування продуктивності	Тестування безпеки	Тестування відмовостійкості
Вибір об’єкта тестування	Визначення функціоналу для тестування	Вибір елементів UI для тестування	Визначення сторінки для тесту (наприклад, каталог)	Вибір частин вебзастосунку для тесту на проникнення	Визначення сценаріїв відмов
Створення сценарію тестування	Розробка тест-кейсів для перевірки функціоналу	Розробка сценаріїв взаємодії з UI	Розробка сценарію поведінки користувача на сайті	Вибір інструментів для тестування на проникнення	Підготовка тестового середовища
Підготовка тестового навантаження	-	-	Визначення кількості користувачів (наприклад, 100)	Запуск тестування на проникнення	Запуск сценаріїв відмов
Запуск тесту	Виконання тест-кейсів	Виконання тестів взаємодії з UI	Виконання тесту з визначеним навантаженням	Сканування на вразливості	Моніторинг та аналіз відновлення
Моніторинг та збір даних	Збір результатів тестування	Збір даних про взаємодію з UI	Спостереження за метриками продуктивності	Експлуатація знайдених вразливостей	Збір та аналіз результатів відмов
Аналіз результатів	Оцінка відповідності функціоналу вимогам	Оцінка зручності та ефективності UI	Оцінка продуктивності сайту	Аналіз результатів тесту	Оцінка продуктивності після відмов
Повторення тесту	Повторення тесту після внесення змін	Повторення тестів UI після внесення змін	Повторення тесту після оптимізації сайту	Виправлення вразливостей та повторне тестування	Оптимізація та повторення тестування

1.3 Стандарти і вимоги тестування

Стандарти та вимоги, які перевіряються під час тестування, можуть змінюватись в залежності від конкретного проєкту, галузі та регіону. Однак, існують загальні категорії стандартів та вимог, які перевіряються під час тестування.

У табл. 1.4 відображені основні категорії стандартів та вимог, які перевіряються під час тестування ПЗ. Відображено основні категорії стандартів та вимог, які важливо враховувати при тестуванні ПЗ, згідно з конкретним проєктом, галуззю, та регіоном світу.

Таблиця 1.4 – основні категорії стандартів та вимог у тестуванні

Категорія стандартів та вимог	Опис	Приклади стандартів
Законодавчі вимоги	Вимоги, засновані на законах та правилах, що регулюють функціонування ПЗ в певній галузі, особливо в частині конфіденційності даних та захисту споживачів.	Вимоги GDPR (General Data Protection Regulation), закони про захист споживачів.
Стандарти безпеки	Визначають вимоги до забезпечення безпеки даних та інформаційних систем.	ISO 27001, стандарти захисту інформації.
Стандарти якості ПЗ	Встановлюють вимоги до якості та надійності ПЗ.	ISO 9001, стандарти якості в різних галузях.
Стандарти інтерфейсів та сумісності	Визначають вимоги до сумісності ПЗ з іншими системами та інтерфейсами.	Стандарти W3C для вебсайтів, протоколи взаємодії.
Стандарти доступності	Містять вимоги до доступності ПЗ для людей з обмеженими можливостями.	WCAG (Web Content Accessibility Guidelines), стандарти доступності ADA (Americans with Disabilities Act).
Стандарти відкритості та інтероперабельності	Визначають вимоги до відкритості та можливості ПЗ працювати разом з іншими продуктами та системами.	Протоколи відкритого обміну даними, стандарти API.
Галузеві стандарти	Визначають вимоги, специфічні для певних галузей.	HIPAA (Health Insurance Portability and Accountability Act) у медицині, стандарти FAA (Federal Aviation Administration) в авіації.

Законодавчі вимоги – закони та правила, які обмежують або визначають, як повинно функціонувати ПЗ в певній галузі, наприклад, щодо конфіденційності даних [38], захисту споживачів [39] тощо.

Стандарти безпеки – визначають вимоги до забезпечення безпеки даних та інформаційних систем, такі як ISO 27001 [40], які визначають набір заходів для захисту даних.

Стандарти якості ПЗ – встановлюють вмоги до якості та надійності ПЗ, такі як ISO 9001 [41] або стандарти якості в інших галузях.

Стандарти інтерфейсів та сумісності – визначають вимоги до сумісності ПЗ з іншими системами та інтерфейсами, наприклад, стандарти W3C для вебсайтів [42].

Стандарти доступності – містять вимоги до доступності для людей з обмеженими можливостями, які регулюють, як ПЗ повинно бути доступним для всіх користувачів.

Стандарти відкритості та інтероперабельності – визначають вимоги до відкритості та можливості ПЗ працювати разом з іншими продуктами та системами.

Галузеві стандарти – визначають вимоги, специфічні для певних галузей, таких як медицина, фінанси, авіація тощо.

Конкретний перелік стандартів та вимог залежить від характеру проєкту і галузі, у якій він ді. Під час тестування важливо переконатися, що ПЗ відповідає усім відповідним стандартам і вимогам, що застосовуються до конкретного випадку.

1.3 Алгоритми тестування

Кожен з видів тестування виконується за відповідним алгоритмом. У табл. 1.5 представлені алгоритми основних видів тестування ПЗ.

Таблиця 1.5 – Алгоритми основних видів тестування

Вид тестування	Алгоритм тестування	Приклад
Функціональне тестування	1. Визначення функціоналу для тестування. 2. Розробка тест-кейсів для перевірки функціоналу. 3. Виконання тест-кейсів. 4. Збір результатів тестування. 5. Оцінка відповідності функціоналу вимогам. 6. Повторення тесту після внесення змін.	Мета: перевірка операції додавання у калькуляторі. Сценарій: введіть «5 + 3», натисніть кнопку «=», переконайтеся, що відображається «8», повторіть тест з різними даними.
UI тестування	1. Вибір елементів UI для тестування. 2. Розробка сценаріїв взаємодії з UI. 3. Виконання тестів взаємодії з UI. 4. Збір даних про взаємодію з UI. 5. Оцінка зручності та ефективності UI. 6. Повторення тестів UI після внесення змін.	Відкрийте веббраузер, перейдіть на сторінку входу, введіть «testuser» та «testpassword», натисніть «Увійти», переконайтеся в переході на особистий кабінет.
Тестування продуктивності	1. Визначення сторінки для тесту (наприклад, каталог). 2. Розробка сценарію поведінки користувача на сайті. 3. Визначення кількості користувачів для тестування. 4. Виконання тесту з визначеним навантаженням. 5. Спостереження за метриками продуктивності. 6. Оцінка продуктивності сайту.	Мета: перевірка сайту на можливість обслуговування 100 одночасних користувачів. Сценарій: користувачі відкривають сторінку каталогу, переглядають товари і додають їх до кошика.
Тестування безпеки	1. Вибір частин вебзастосунку для тесту на проникнення. 2. Вибір інструментів для тестування на проникнення. 3. Сканування на вразливості. 4. Експлуатація знайдених вразливостей. 5. Аналіз результатів тесту.	Мета: перевірка на вразливості SQL-ін’єкції. Інструменти: Burp Suite або SQLMap. Сканування вхідних полів форм на наявність SQL-ін’єкцій.
Тестування відмовостійкості	1. Визначення сценаріїв відмов. 2. Підготовка тестового середовища. 3. Запуск сценаріїв відмов. 4. Моніторинг та аналіз відновлення. 5. Збір та аналіз результатів.	Мета: перевірка відновлення сайту після втрати зв’язку з БД. Сценарій: вимкнення сервера БД та спостереження за реакцією сайту.

Розглянемо алгоритми окремих видів тестування на прикладах. Функціональне тестування: перевірка функціональності операції додавання в калькуляторі. Алгоритм:

1. Підготовка тестового середовища: запустити калькулятор на комп'ютері або іншому пристрої;

2. Визначення тестового випадку: обрати операцію додавання на інтерфейсі калькулятора;

3. Введення тестових вхідних даних:

- на клавіатурі ввести перше число, наприклад, «5»;
- використовуючи інтерфейс калькулятора, ввести знак «+»;
- на клавіатурі ввести друге число, наприклад, «3»;

4. Виконання операції: клацнути «=« або аналогічну кнопку в інтерфейсі програми;

5. Перевірка результату: перевірити, що на екрані калькулятора відображається результат операції, в цьому випадку «8»;

6. Повторення тесту: повторити тест з різними вхідними даними, наприклад, «2 + 2» і «10 + 7», щоб переконатися, що операція додавання працює коректно для різних вхідних значень;

7. Завершення тесту: закрити калькулятор або перейти до іншої функціональності для проведення інших тестів.

Приклад (функціональне тестування):

- мета – перевірити операцію додавання у калькуляторі;
- введіть «5 + 3»;
- натисніть кнопку «=»;
- перевірите, що на екрані відображається «8»;
- повторити тест з різними вхідними даними;
- завершити тестування або перейти до тестування іншої функціональності.

Тестування інтерфейсу користувача (UI тестування): перевірка інтерфейсу користувача вебсторінки для входу на сайт. Алгоритм:

1. Запуск тестового середовища: відкрийте веббраузер та перейдіть на сторінку входу на сайт;

2. Введення даних:

- знайдіть поле для введення імені користувача;
- введіть ім’я користувача, наприклад, «testuser»;

3. Введення паролю:

- знайдіть поле для введення паролю;
- введіть пароль, наприклад, «testpassword»;

4. Натискання кнопки «Увійти»:

- знайдіть кнопку для входу на сайт, яка зазвичай позначена як «Увійти» або «Увійти в систему»;
- натисніть на цю кнопку;

5. Перевірка результату:

- переконайтеся, що ви переходите в особистий кабінет або на головну сторінку сайту після входу;
- перевірте, чи відображається ваше ім’я користувача або інша інформація, що підтверджує успішний вхід;

6. Повторення тесту:

- повторіть тест з різними комбінаціями імен користувача та паролів, щоб переконатися, що інтерфейс користувача взаємодіє правильно;

7. Тестування на різних браузерах та пристроях:

- перевірте роботу інтерфейсу на різних веббраузерах, таких як Chrome, Firefox, та на різних пристроях (комп’ютер, смартфон, планшет);

8 Завершення тесту:

- вийдіть з облікового запису, закрийте веббраузер та перейдіть до інших тестів;

Приклад (UI тестування):

- відкрийте веббраузер і перейдіть на сторінку входу на сайт;
- введіть ім’я користувача «testuser» та пароль «testpassword»;
- натисніть кнопку «Увійти»;

- переконайтеся, що ви переходите на особистий кабінет, і ваше ім'я користувача відображається.

Тестування продуктивності: перевірити продуктивність вебсайту під навантаженням і визначити, чи він здатний обслуговувати багато користувачів одночасно. Алгоритм:

1. Визначення цілей тестування: визначте, яку частину функціональності сайту ви будете тестувати щодо продуктивності. Наприклад, це може бути сторінка домашнього каталогу або сторінка оформлення замовлення;

2. Створення сценарію тестування: розробіть сценарій, який відображає поведінку користувача на сайті. Наприклад, це може бути відвідування сторінки каталогу, перегляд товарів і додавання товарів до кошика;

3. Підготовка тестового навантаження:

- визначте, скільки одночасних користувачів ви хочете моделювати під час тестування, наприклад, 100 користувачів;

- використовуйте інструменти для навантаження, такі як Apache JMeter [35] або Gatling [36], для створення навантаження, яке імітує роботу користувачів;

4. Запуск тесту: запустіть тест із зазначеними параметрами навантаження;

5. Моніторинг та збір даних:

- під час тестування слідкуйте за метриками продуктивності, такими як час відповіді сервера, завантаження сервера, швидкість завантаження сторінок тощо;

- збирайте дані про завантаження сервера та відгуки від користувачів під час тестування;

6. Аналіз результатів:

- аналізуйте зібрані дані та визначайте, чи виконує сайт вимоги до продуктивності;

- визначте проблєми та можливі шляхи їх вирішення, такі як оптимізація коду або збільшення ресурсів сервера;

7. Повторення тесту та оптимізація:

- повторіть тест після внесення змін до сайту або сервера;

- продовжуйте тестування та оптимізацію, доки сайт не досягне бажаного рівня продуктивності.

Приклад (тестування продуктивності):

- мета тестування – перевірити, чи вебсайт здатний обслуговувати 100 одночасних користувачів на сторінці каталогу;

- сценарій тестування: користувачі відкривають сторінку каталогу, переглядають товари та додають їх до кошика;

- запуск тесту: за допомогою інструменту JMeter запустіть тест з 100 одночасними користувачами, які виконують заданий сценарій;

- аналіз результатів: перевірте, чи сайт відповідає за прийнятними метриками продуктивності

Алгоритм тестування безпеки (security testing): перевірити безпеку вебзастосунку за допомогою тестування на проникнення (penetration testing):

1 Вибір цілей тестування: визначити, які частини вебзастосунку ви хочете протестувати на проникнення. Нариклад, це може бути вхід до системи, сторінки адміністратора, база даних тощо;

2 Вибір інструментів тестування: вибрати інструменти для тестування на проникнення, як Burp Suite [37], OWASP ZAP [38] або Metasploit [39];

3 Запуск тесту: використовуйте вибрані інструменти для проведення тестування на проникнення;

4 Сканування на вразливості: скануйте вебзастосунок на наявність вразливостей, таких як SQL-ін’єкції, міжсайтові атаки (XSS), невірно налаштовані сервери та інші потенційні проблєми;

5 Експлуатація вразливостей: якщо під час сканування знайдено вразливості, спробуйте їх експлуатувати, щоб переконатися, чи є це реальною загрозою для системи;

6 Аналіз результатів: аналізуйте результати тестування, визначте, чи є серйозні вразливості, і складіть звіт про результати;

7 Подання звіту та виправлення:

- звіт з результатами тестування розробникам та адміністраторам системи;

- виправте виявлені вразливості та знову протестуйте безпеку вебзастосунку;

8 Повторне тестування: повторюйте тестування на проникнення після виправлення вразливостей, щоб переконатися, що система залишилася безпечною.

Приклад (тестування безпеки):

- мета тесту – перевірити вебзастосунок на вразливості SQL-ін'єкції;
- вибрані інструменти – Burp Suite або SQLMap [40];
- запуск тесту – використовуйте обраний інструмент для сканування вхідних полів форм на вебсайті на наявність SQL-ін'єкцій;
- сканування на вразливості – інструмент може виявити наявність вразливостей;
- експлуатація вразливостей – спробуйте виконати SQL-ін'єкції для отримання доступу до бази даних або інших конфіденційних даних;
- аналіз результатів – звіт містить інформацію про знайдені вразливості та можливі наслідки їх експлуатації

Тестування відмовостійкості (resilience testing): перевіряє, як система або додаток поводиться під час виникнення відмов або помилок і чи здатний він відновлюватися безперебійно. Алгоритм:

1. Визначити сценарії відмов, які потрібно перевірити, наприкалад, втрата зв'язку з базою даних, втрата доступу до інтернету, або помилки в роботі сторонніх служб;

2. Підготувати тестове середовище, на якому будуть моделюватись відмови. Наприклад, можна вимкнути сервер бази даних або від'єднати доступ до мережі;

3. Запустити попередньо визначені сценарії відмов у тестовому середовищі. Наприклад, вимкнути сервер бази даних, а потім спостерігати, як система реагує;

4. Моніторинг та аналіз відновлення – слідкуйте за тим, як система реагує на відмови. Оцінюйте, чи вона може автоматично відновлюватися безперебійно, або чи потрібна втручання оператора;

5. Збір і аналіз результатів – зберіть результати тестування і визначте, як відмінно система або додаток відновлюються від відмов;

6. Оцініть продуктивність системи або додатку під час відмов та після відновлення;

7. Оптимізація та вдосконалення – виправте виявлені проблєми та удоскональте процес відновлення;

8. Повторюйте тестування відмовостійкості, щоб переконатися, що система може витримати відмови та відновлюватися ефективно.

1.4 Метрики тестування

Для оцінки результатів різних видів тестування використовуються спеціальні показники – метрики, що є специфічними для кожного з видів тестування [43].

Метрики оцінки результатів функціонального тестування допомагають визначити якість тестування та функціональну придатність ПЗ.

Покриття тестами (Test Coverage) [44], включає:

- покриття коду – вимірює, яка частина вихідного коду була протестована; включає такі підметрики, як покриття рядків коду, гілок (branch coverage), рішень (decision coverage) і умов (condition coverage);
- покриття функціональності – метрика вказує, скільки функціональних вимог було протестовано; це допомагає визначити, які частини системи залишилися без тестування.

Кількість виявлених дефектів (Defect Metrics):

- кількість виявлених дефектів на одиницю часу – вимірює, як ефективно тестування виявляє помилки в програмному продукті;

- кількість виявлених дефектів за пріоритетами – допомагає визначити, які дефекти є найбільш критичними для виправлення.

Табл. 1.6 узагальнює відомості про основні метрики, що використовуються для оцінки результатів різних видів тестування ПЗ.

Таблиця 1.6 – Основні метрики оцінки результатів тестування

Вид тестування	Основні метрики	Що вимірюється
Функціональне тестування	Покриття тестами, кількість виявлених дефектів, час виявлення дефектів, кількість успішних/невдалих тестів, середній час виконання тестів, відсоток помилок до випуску, відсоток виконаних тестів	Якість тестування, функціональна придатність, ефективність виявлення помилок, стабільність системи
UI тестування	Кількість виявлених дефектів, кількість інтерфейсних протилежностей, час виявлення дефектів, задоволеність користувачів, продуктивність інтерфейсу, кількість помилкових дій користувача, вартість тестування, відсоток завершених тестів	Якість інтерфейсу користувача, відповідність дизайну, реакція на зміни, задоволеність користувачів, продуктивність UI
Тестування продуктивності	Час відгуку, пропускна здатність, відсоток відмов, завантаження сервера, відсоток використання ресурсів, скасовані запити, стабільність продукту під навантаженням, розмір даних в кеші, середній час відновлення	Швидкість обробки запитів, обсяг оброблюваних даних, надійність, навантаження на сервер, використання ресурсів, стабільність під навантаженням
Тестування безпеки	Кількість виявлених вразливостей, кількість ліквідованих вразливостей, рівень серйозності вразливостей, час виявлення вразливостей, популярність вразливостей, кількість фальшивих позитивів/негативів, вартість виправлення, рівень довіри користувачів	Рівень безпеки, ефективність виявлення вразливостей, серйозність ризиків, вартість виправлення, довіра користувачів
Тестування відмовостійкості	Середній час відновлення, кількість відмов, ступінь відмовостійкості, час відновлення до нормального режиму, реакція на аварійні ситуації, відсоток відновлених послуг, вартість відновлення, точки відмови	Швидкість відновлення, стійкість до відмов, надійність, ефективність відновлення, вартість відновлення

Час виявлення дефектів (Defect Detection Time):

- середній час виявлення дефектів після внесення змін (разработчиком) – вказує, як швидко та ефективно тестування реагує на нові зміни в коді;

- середній час виявлення дефектів після виправлення (виконавцем) – визначає, наскільки ефективно тестування перевіряє виправлення дефектів.

Кількість успішних та невдалих тестів (Pass/Fail Metrics) – визначає кількість успішних та невдалих тестових випробувань, надає інформацію про стабільність системи.

Середній час виконання тестів (Average Test Execution Time) – вказує, скільки часу зазвичай потрібно для виконання тестів, допомагає визначити продуктивність тестування.

Відсоток помилок, виявлених до випуску (Pre-release Defect Percentage) – визначає відсоток дефектів, виявлених до випуску програмного продукту, важливо для забезпечення високої якості перед випуском.

Відсоток виконаних тестових випробувань (Test Pass Percentage) – вказує, який відсоток тестових випробувань пройшли успішно, допомагає визначити загальний стан тестування.

Метрики оцінки результатів тестування інтерфейсу користувача (UI тестування) допомагають визначити якість тестування та користувацьку орієнтованість програмного продукту.

Кількість виявлених дефектів (Defect Metrics), включає:

- кількість виявлених дефектів – вимірює загальну кількість дефектів, виявлених під час тестування інтерфейсу користувача;

- кількість виявлених дефектів на одиницю часу – показує, наскільки ефективно тестування виявляє помилки в інтерфейсі користувача.

Кількість інтерфейсних протилежностей (UI Discrepancy Metrics) – вимірює кількість розбіжностей між дизайном інтерфейсу та фактичною реалізацією; визначає, наскільки точно програмний продукт відповідає дизайну;

Задоволеність користувачів (User Satisfaction) – вимірює, наскільки користувачі задоволені інтерфейсом; зазвичай ця метрика збирається через опитування користувачів;

Продуктивність інтерфейсу (UI Performance) – включає вимірювання часу відгуку інтерфейсу на дії користувача, такі як час завантаження сторінок або відповідь на кліки тощо.

Кількість помилкових дій користувача (User Errors) – вимірює кількість помилкових дій користувача в інтерфейсі, таких як неправильні кліки або введення невірних даних.

Вартість тестування (Testing Cost) – визначає вартість тестування інтерфейсу користувача, включаючи витрати на інструменти та ресурси.

Відсоток завершених тестових випробувань (Test Completion Percentage) – вказує, який відсоток тестових випробувань пройшли успішно.

Метрики оцінки результатів тестування продуктивності використовуються метрики допомагають визначити ефективність та швидкодію програмного продукту під час навантаження.

1. Час відгуку (Response Time) – вимірює час, який потрібен для обробки запиту користувача програмним продуктом Включає час очікування та час виконання операції;

2. Пропускна здатність (Throughput) – визначає, скільки запитів чи транзакцій програмний продукт може обробити за одиницю часу; вимірюється, наприклад, у запитах на секунду (requests per second) або транзакціях на секунду (transactions per second);

3. Відсоток відмов (Error Rate) – вказує, який відсоток запитів чи транзакцій завершився помилкою чи відмовою під час тестування продуктивності;

4. Завантаження сервера (Server Load) – вимірює навантаження на сервер, включаючи кількість одночасних запитів, завантаження процесора, пам’ять і мережу;

5. Відсоток використання ресурсів (Resource Utilization) – вказує, який відсоток ресурсів сервера (процесор, пам'ять, дисковий простір, мережа) використовується під час тестування;

6. Скасовані запити (Aborted Requests) – вимірює кількість запитів, які були скасовані через довгий час очікування чи помилки відповіді;

7. Стабільність продукту під навантаженням (Stability under Load) – оцінює, як стабільно програмний продукт працює під великим навантаженням, включає в себе оцінку витрат ресурсів та відсутність викидів;

8. Розмір даних в кеші (Cache Hit Rate) – вимірює, як часто запити задовольняються з кеша, що допомагає зменшити навантаження на сервер та прискорює відгук системи;

9. Середній час відновлення (Average Recovery Time) – вказує, який час потрібен для відновлення продуктивності після відмови чи перевантаження.

Метрики оцінки результатів тестування безпеки допомагають визначити рівень безпеки програмного продукту та ідентифікувати потенційні уразливості.

1. Кількість виявлених вразливостей (Vulnerabilities) – вимірює кількість потенційних вразливостей, виявлених під час тестування безпеки, таких як SQL-ін'єкції, переповнення буфера, кросс-сайтові атаки і т. д.;

2. Кількість ліквідованих вразливостей (Resolved Vulnerabilities) – вказує, скільки вразливостей було виправлено розробниками після виявлення під час тестування безпеки;

3. Рівень серйозності вразливостей (Severity Level of Vulnerabilities) – визначає серйозність виявлених вразливостей, допомагаючи пріоритезувати їх виправлення;

4. Час виявлення вразливостей (Vulnerability Detection Time) – вимірює, скільки часу знадобилося для виявлення вразливостей після початку тестування;

5. Популярність вразливостей (Common Vulnerabilities) – оцінює, наскільки виявлені вразливості загальновідомі та часто зустрічаються в програмному продукті, це допомагає визначити, наскільки програмний продукт піддається типовим атакам;

6. Кількість фальшивих позитивів (False Positives) – вказує, скільки заявлених вразливостей під час тестування безпеки виявились помилковими, тобто насправді вразливості відсутні;

7. Кількість фальшивих негативів (False Negatives) – вказує, скільки реальних вразливостей не було виявлено під час тестування безпеки;

8. Вартість виправлення вразливостей (Cost of Remediation) – оцінює, скільки ресурсів (час, кошти) потрібно для виправлення виявлених вразливостей;

9. Рівень довіри користувачів (User Trust Level) – вказує, наскільки користувачі довіряють програмному продукту, особливо в контексті забезпечення їхньої безпеки.

Метрики оцінки результатів тестування відмовостійкості (resilience testing) допомагають визначити стійкість програмного продукту до відмов і негативних впливів.

1. Середній час відновлення (Average Recovery Time) – вимірює, який час потрібен для відновлення роботи програмного продукту після виникнення відмови; ця метрика допомагає визначити, наскільки швидко програмний продукт може відновити нормальну роботу після відмови;

2. Кількість відмов (Failure Count) – вимірює загальну кількість відмов, які сталися під час тестування відмовостійкості;

3. Ступінь відмовостійкості (Resilience Level) – оцінює наскільки програмний продукт може витримати відмови та залишитися працездатним, виражається у відсотках або за шкалою балів;

4. Час відновлення до нормального режиму (Recovery Time to Normal) – вимірює, який час потрібен для повернення програмного продукту до нормального режиму роботи після відмови;

5. Реакція на аварійні ситуації (Emergency Response) – оцінює, наскільки ефективно програмний продукт реагує на аварійні ситуації і забезпечує безперервну роботу;

6. Відсоток відновлених послуг (Recovered Services Percentage) – вказує, який відсоток послуг чи функціональності було відновлено після виникнення відмов;

7. Вартість відновлення (Recovery Cost) – оцінює, скільки ресурсів (час, кошти) було витрачено на відновлення програмного продукту після відмов;

8. Точки відмови (Failure Points) – визначає критичні точки, де часто відбуваються відмови, що допомагає пріоритезувати заходи по покращенню відмовостійкості.

Висновки до розділу 1

1. Метод тестування «чорного ящика» базується на перевірці функціональності ПЗ без аналізу внутрішньої структури коду. Історія його розвитку відображає еволюцію підходів до тестування ПЗ, з акцентом на зовнішній поведінці та взаємодії з користувачем.

2. Існують різні види тестування (модульне, інтеграційне, системне, приймальне та ін.) та різні етапи процесу (від планування до виконання та аналізу результатів). Розуміння цих видів та етапів є важливим для ефективного управління тестуванням ПЗ.

3. Тестування ПЗ регулюється рядом стандартів та вимог, що включають законодавчі норми, стандарти безпеки, якості, сумісності, доступності, відкритості та інтероперабельності. Вони визначають ключові аспекти процесу тестування та його цілей.

4. Алгоритми тестування включають різні методики та підходи, спрямовані на ідентифікацію помилок та вад в програмному забезпеченні. Ці методики допомагають систематизувати процес тестування і підвищити його ефективність.

5. Метрики тестування важливі для оцінки ефективності та якості процесу тестування. Вони включають показники, які відображають кількість виявлених

помилок, ступінь покриття тестами, час, витрачений на тестування, та інші критичні параметри.

Результати розділу показують важливість комплексного підходу до тестування ПЗ, що включає розуміння різних методів, стандартів та критеріїв якості, а також використання ефективних алгоритмів та метрик для забезпечення надійності та якості програмного продукту.

РОЗДІЛ 2
АНАЛІЗ МЕТОДІВ ТА ІНСТРУМЕНТІВ ТЕСТУВАННЯ «ЧОРНОГО ЯЩИКА»

2.1 Класифікація методів тестування «чорного ящика»

Методи тестування «чорного ящика» орієнтовані на перевірку функціональності та поведінки програмного продукту без знання внутрішньої реалізації коду. Класифікація цих методів відповідає основним методам тестування, і може бути конкретизована, з урахуванням особливостей тестування «чорного ящика», наступним чином [45].

Функціональне тестування (Functional Testing) – тестування функціональних можливостей програми, перевіряє, чи виконує ПЗ очікувані функції і операції. У цей клас входять такі види тестування, як:

- тестування коректності (Correctness Testing) – перевірка на відповідність функціональним вимогам;
- тестування на відповідність стандартам (Compliance Testing) – визначення, чи відповідає ПЗ стандартам та правилам.

Тестування інтерфейсу користувача (UI Testing) – оцінює, як користувачі взаємодіють з інтерфейсом програми, перевіряється коректність відображення та функціональність інтерфейсу.

Тестування продуктивності (Performance Testing) – оцінює швидкість, витрати ресурсів та стабільність ПЗ під навантаженням, включає:

- тестування навантаження (Load Testing) – перевірка роботи програми при максимальному або очікуваному навантаженні;
- тестування продуктивності (Performance Testing) – вимірює час відгуку та швидкість виконання операцій;
- тестування стабільності (стресостійкості) (Stress Testing) – оцінює, як ПЗ поводиться під екстремальним навантаженням.

Тестування безпеки (Security Testing) – орієнтоване на виявлення вразливостей та покращення захисту програми від атак і порушень безпеки.

Тестування відмовостійкості (Resilience Testing) – оцінює стійкість ПЗ до відмов, включаючи аварійні ситуації і негативні впливи.

Тестування сумісності (Compatibility Testing) – перевіряє, як програма працює на різних платформах, браузерах та пристроях.

Тестування відновлення (Recovery Testing) – оцінює, наскільки швидко та ефективно програмний продукт відновлюється після відмов.

Тестування сценаріїв використання (Use Case Testing) – перевірка відповідності програми реальним сценаріям використання, включаючи послідовності операцій.Класифікація методів тестування «чорного ящика» представлена у табл. 2.1.

Таблиця 2.1 – Класифікація методів тестування «чорного ящика»

Вид Тестування	Описання
Функціональне тестування (Functional Testing)	Перевірка функціональних можливостей програми, включає тестування коректності, на відповідність стандартам
Тестування інтерфейсу користувача (UI Testing)	Оцінка взаємодії користувачів з інтерфейсом програми, перевірка коректності відображення та функціональності інтерфейсу
Тестування продуктивності (Performance Testing)	Оцінка швидкості, витрат ресурсів та стабільності програмного продукту, включає тестування навантаження, продуктивності та стабільності
Тестування безпеки (Security Testing)	Виявлення вразливостей та покращення захисту програми від атак і порушень безпеки
Тестування відмовостійкості (Resilience Testing)	Оцінка стійкості програмного продукту до відмов, включаючи аварійні ситуації та негативні впливи
Тестування сумісності (Compatibility Testing)	Перевірка роботи програми на різних платформах, браузерах та пристроях
Тестування відновлення (Recovery Testing)	Оцінка швидкості та ефективності відновлення програмного продукту після відмов
Тестування сценаріїв використання (Use Case Testing)	Перевірка відповідності програми реальним сценаріям використання, включаючи послідовності операцій

На практиці, використовуються й інші підходи до класифікації методів тестування «чорного ящика», зокрема, за різними критеріями [46].

У табл. 2.2 узагальнено різні підходи до методів тестування «чорного ящика» залежно від засобів представлення вхідних даних, рівнів тестування, рівня знання про програму, цілей тестування та підходів до тестування.

Таблиця 2.2 – Класифікація методів тестування «чорного ящика»

Методи тестування	Описання
Методи тестування на основі специфікації	Використовують формальні або написані вимоги для створення тестових випробувань
Методи тестування на основі моделі	Використовують моделі системи або її компонентів для автоматичної генерації тестів
Методи тестування на основі спостереження	Вимірюють поведінку програми під час виконання тестів без аналізу коду
Методи юніт-тестування	Орієнтовані на окремі компоненти або функції програми
Методи інтеграційного тестування	Перевіряють взаємодію між компонентами
Методи системного тестування	Оцінюють роботу всієї системи
Методи з повним доступом до коду	Тестувальники мають знання про внутрішню реалізацію програми
Методи без знання про код	Тестувальники не мають доступу до коду та базуються на зовнішньому поведінці програми
Функціональне тестування	Перевірка відповідності програми функціональним вимогам
Тестування продуктивності	Вимірювання швидкості та ефективності роботи програми
Тестування безпеки	Виявлення вразливостей і покращення захисту програми
Тестування з використанням сценаріїв	Використовує випробовування конкретних сценаріїв використання
Тестування з використанням технік	Використання певних методик та технік тестування
Позитивні тести	Перевіряють правильну роботу програми відповідно до специфікації
Негативні тести	Перевіряють реакцію програми на невірні або аномальні вхідні дані

Крім цього, використовується також класифікація методів тестування «чорного ящика» відповідно до їх призначення та способу застосування [47].

Наступна табл. 2.3 відображає різні види методів тестування «чорного ящика» відповідно до їх призначення та способу застосування.

Таблиця 2.3 – Види методів тестування «чорного ящика» за призначенням та застосуванням

Вид тестування	Описання
Функціональне тестування (Functional Testing)	Включає тестування відповідності та тестування правильності для перевірки відповідності програми специфікації та її функціональності
Тестування інтерфейсу користувача (UI Testing)	Включає тестування взаємодії користувача та тестування на доступність для оцінки зручності інтерфейсу та доступності для користувачів з обмеженими можливостями
Тестування продуктивності (Performance Testing)	Включає тестування завантаження та тестування швидкості для оцінки роботи програми під підвищеним навантаженням та швидкості відгуку
Тестування безпеки (Security Testing)	Включає тестування вразливостей та тестування автентифікації та авторизації для пошуку вразливостей та перевірки управління доступом
Тестування відмовостійкості (Resilience Testing)	Оцінка стійкості програми під впливом відмов і помилок
Тестування сумісності (Compatibility Testing)	Перевірка роботи програми на різних операційних системах та пристроях
Тестування відновлення (Recovery Testing)	Перевірка здатності програми відновитися після аварії або відмови
Тестування сценаріїв використання (Use Case Testing)	Перевірка програми на відповідність конкретним сценаріям використання

Методи тестування «чорного ящика» можна розділити на декілька основних класів з такими характерними рисами та особливостями [48].

Функціональне тестування (Functional Testing). Перевіряє, чи програма відповідає функціональним вимогам та специфікації, тобто спрямоване на перевірку правильності функціоналу програми. Тести базуються на зовнішньому поведінці програми. Використовується для перевірки, чи програма робить те, що має робити, та чи відповідає специфікації.

Тестування інтерфейсу користувача (UI Testing), Виконує оцінку користувацького інтерфейсу програми та її взаємодії з користувачем. Тести спрямовані на зручність та доступність інтерфейсу для користувачів. Важливо для програм, які взаємодіють з користувачем. Тести включають оцінку навігації, відображення даних тощо.

Кожен клас методів тестування «чорного ящика» використовується в залежності від конкретних потреб та особливостей програмного продукту (табл. 2.4).

Таблиця 2.4 – Характеристика методів тестування «чорного ящика»

Вид тестування	Характеристика	Особливості	Використання
Функціональне тестування (Functional Testing)	Перевірка відповідності функціональним вимогам та специфікації	Спрямоване на перевірку правильності функціоналу програми, базується на зовнішньому поведінці	Використовується для перевірки, чи програма робить те, що має робити, та чи відповідає специфікації
Тестування інтерфейсу користувача (UI Testing)	Оцінка користувальницького інтерфейсу програми та взаємодії з користувачем	Тести спрямовані на зручність та доступність інтерфейсу	Важливо для програм, які взаємодіють з користувачем; оцінка навігації, відображення даних тощо
Тестування продуктивності (Performance Testing)	Вимірювання швидкості та продуктивності програми під навантаженням	Включає тестування завантаження, швидкості відгуку та інші параметри продуктивності	Важливо для великих систем та вебдодатків, де продуктивність є ключовою
Тестування безпеки (Security Testing)	Виявлення вразливостей та заходи безпеки програми	Тести націлені на захист від вторгнень та зловживань	Важливо для програм, що містять конфіденційну інформацію або використовуються в критичних сферах
Тестування відмовостійкості (Resilience Testing)	Оцінка стійкості програми до відмов та помилок	Тести спрямовані на виявлення реакції програми на відмови	Важливо для систем, де надійність є ключовою, наприклад, в медичному обладнанні або авіаційній техніці
Тестування сумісності (Compatibility Testing)	Перевірка роботи програми на різних платформах та пристроях	Спрямоване на забезпечення сумісності та відтворюваності програми	Важливо для додатків, доступних на різних платформах

Тестування продуктивності (Performance Testing). Вимірювання швидкості та продуктивності програми під навантаженням. Включає тестування завантаження, швидкості відгуку та інші параметри продуктивності. Важливе для великих систем та вебдодатків, де продуктивність грає ключову роль.

Тестування безпеки (Security Testing). Виявлення вразливостей та заходи безпеки програми. Тести націлені на захист від вторгнень та зловживань. Важливо для програм, які містять конфіденційну інформацію або використовуються в критичних сферах.

Тестування відмовостійкості (Resilience Testing). Оцінка стійкості програми до відмов та помилок. Тести спрямовані на виявлення, як програма веде себе в умовах відмови. Важливо для систем, де надійність грає ключову роль, таких як медичне обладнання або авіаційна техніка.

Тестування сумісності (Compatibility Testing). Перевірка, як програма працює на різних платформах та пристроях. Спрямоване на забезпечення сумісності та відтворюваності програми на різних пристроях. Застосовується для додатків, які мають бути доступними на різних платформах.

Розглянемо приклади специфіки використання методів тестування «чорного ящика» з різних класів.

Функціональне тестування: тест застосунку для операційної системи, який повинен виконувати операції створення, читання, оновлення та видалення даних. Тести функціонального тестування перевіряють, чи програма виконує ці операції правильно.

Тестування інтерфейсу користувача: тест вебсайту електронного магазину. Тести інтерфейсу користувача перевіряють, як користувачі взаємодіють з сайтом – просто переглядають або використовують функціональність покупок.

Тестування продуктивності: тестування великої корпоративної бази даних. Тести продуктивності перевіряють, як швидко система відгукується на запити, як вона обробляє багато одночасних запитів і чи масштабована для великих обсягів даних.

Тестування безпеки: тест мобільного додатку для банківських операцій. Тести безпеки перевіряють, чи можливо вразити додаток, зловживати даними користувачів, зламати фінансову інформацію.

Тестування відмовостійкості: медична система моніторингу пацієнтів. Тести відмовостійкості перевіряють, як система веде себе під час відмови мережі, якщо частина обладнання вийде з ладу тощо.

Тестування сумісності: мобільний додаток для фотографій. Тести сумісності перевіряють, як добре додаток працює на різних версіях операційних систем, різних пристроях та розширеннях.

Табл. 2.5 узагальнює приклади використання методів тестування «чорного ящика» з різних класів.

Таблиця 2.5 – Використання різних видів тестування «чорного ящика»

Вид тестування	Приклад використання
Функціональне тестування	Додаток для операційної системи, який виконує операції створення, читання, оновлення, видалення даних. Тести перевіряють, чи правильно програма виконує ці операції.
Тестування інтерфейсу користувача	Вебсайт електронного магазину. Тести перевіряють, як користувачі взаємодіють з сайтом, простоту навігації, використання функціональності покупок.
Тестування продуктивності	Велика корпоративна база даних. Тести перевіряють швидкість відгуку системи на запити, обробку багатьох одночасних запитів, масштабованість для великих обсягів даних.
Тестування безпеки	Мобільний додаток для банківських операцій. Тести перевіряють можливість вразити додаток, зловживання даними користувачів, взлом фінансової інформації.
Тестування відмовостійкості	Медична система моніторингу пацієнтів. Тести перевіряють поведінку системи під час відмови мережі, виходу частини обладнання з ладу.
Тестування сумісності	Мобільний додаток для фотографій. Тести перевіряють, як добре додаток працює на різних версіях ОС, різних пристроях та розширеннях.

Порівняння та аналіз переваг та недоліків кожного методу з конкретними прикладами, представлені у табл. 2.6.

Таблиця 2.6 – Переваги та недоліки методів тестування «чорного ящика»

Вид тестування	Переваги	Недоліки	Приклади
Функціональне тестування	Визначає відповідність функціональним вимогам, просте у використанні	Не виявляє внутрішніх помилок, може не виявити недоліки при взаємодії функцій	Електронний кабінет банку: тестування операцій переказу грошей, перевірки балансу
Тестування інтерфейсу користувача	Оцінює зручність і доступність, виявляє проблєми навігації	Чутливе до змін у дизайні, не виявляє внутрішніх помилок	Мобільний додаток соціальної мережі: тестування зручності користування кнопками, меню
Тестування продуктивності	Виявляє проблєми продуктивності і масштабованості	Вимагає ресурсів для створення навантаження, може бути дорогим	Онлайн-торговий сайт: тестування обробки великої кількості запитів
Тестування безпеки	Виявляє вразливості, захищає від зловживань	Вимагає спеціалізованих знань, часомістке	Мобільний додаток для банківських операцій: тестування на вразливості, захист конфіденційних даних
Тестування відмовостійкості	Виявляє поведінку при відмовах, важливий для надійних систем	Важко підготувати сценарії відмов, вимагає спеціалізованих знань	Медична система моніторингу пацієнтів: тестування на стійкість до відмов обладнання
Тестування сумісності	Допомагає переконатися, що програма працює на різних платформах	Вимагає тестування на різних пристроях, часомістке	Мобільний додаток для фотографій: тестування на різних версіях ОС і пристроях

2.2. Стратегії та стандарти тестування «чорного ящика»

Тестування «чорного ящика» використовує різні стратегії для виявлення помилок і недоліків у програмному забезпеченні. Основні стратегії включають [49]:

- функціональне тестування – спрямоване на перевірку функціональності програми; перевіряє, чи виконує програма очікувані дії та чи поводиться правильно при різних вхідних даних;

- тестування інтерфейсу користувача (UI тестування) – оцінює користувацький інтерфейс програми; перевіряє, чи інтерфейс легко зрозумілий та чи реагує правильно на взаємодію користувача;

- тестування продуктивності – в цій стратегії важливо визначити, як швидко і наскільки ресурсоємно працює програма під навантаженням; тестувальники можуть створити ситуації з великим навантаженням для оцінки продуктивності;

- тестування безпеки – стратегія спрямована на виявлення вразливостей та можливих атак на програму; тестувальники намагаються проникнути в систему, використовуючи різні способи, щоб виявити слабкі місця в безпеці програми;

- тестування відмовостійкості – в цій стратегії перевіряється, як програма веде себе під час відмов апаратних чи програмних компонентів; тестувальники можуть вимушено викликати відмови, щоб визначити, як програма реагує;

- тестування відповідності стандартам та вимогам – тестувальники перевіряють, чи програма відповідає стандартам і вимогам, встановленим для конкретного проєкту чи галузі.

Зазвичай, використовують комбінацію різних стратегій, щоб забезпечити повноту тестування та виявлення різних категорій проблєм.

Отже, стратегія функціонального тестування полягає у перевірці та оцінці функціональності ПЗ. Основні аспекти цієї стратегії включають:

- визначення функціональності. Перш за все, необхідно визначити очікувану функціональність програми, створити детальний список функцій і можливостей, які повинні бути доступні користувачам;

- створення тест-кейсів. На основі визначеної функціональності створюються тест-кейси, які включають набори вхідних даних, дії користувача та очікувані результати. Кожен тест-кейс спрямований на перевірку певної функції чи можливості;

- виконання тестів. Тестувальники виконують створені тест-кейси відповідно до сценаріїв, які передбачають взаємодію користувача з програмою.

Вони вводять вхідні дані, виконують дії та порівнюють отримані результати з очікуваними;

- реєстрація дефектів. Якщо тест-кейси виявляють помилки, тестувальники реєструють їх у системі для відстеження дефектів. Дефекти включають неправильну реакцію програми на вхідні дані, непередбачувані поведінки чи будь-які аномалії;

- перевірка відповідності функціональності вимогам. Кожен тест-кейс перевіряє, чи відповідає програма вимогам та специфікаціям. Якщо будь-яка функціональність не відповідає вимогам, це вважається дефектом;

- регресійне тестування. Після виявлення та виправлення дефектів необхідно повторно виконати функціональні тест-кейси, щоб переконатися, що виправлення не вплинули на інші частини програми;

- автоматизація тестування. Для ефективності і повторюваності тестування багато функціональних тестів можуть бути автоматизовані за допомогою спеціальних інструментів.

Стратегія тестування інтерфейсу користувача (UI тестування) полягає у перевірці та оцінці користувацького інтерфейсу ПЗ, зокрема візуальної і функціональної частини. Основні аспекти цієї стратегії включають:

- візуальна перевірка дизайну. Тестувальники оцінюють естетичний дизайн інтерфейсу, включаючи колір, шрифти, розміщення та інші аспекти, щоб переконатися, що він виглядає професійно і зручно для користувача;

- перевірка коректності відображення. Важливо впевнитися, що інтерфейс правильно відображається на різних типах пристроїв та роздільних здатностях екрану. Тестувальники переконуються, що текст та зображення відображаються коректно;

- функціональність інтерфейсу. Перевірка включає в себе тестування всіх елементів керування (кнопки, посилання, поля введення тощо), щоб переконатись, що вони працюють правильно та взаємодіють з іншими частинами програми;

- взаємодія з користувачем. Тестувальники перевіряють, чи програма взаємодіє з користувачем правильно. Це включає в себе перевірку реакції на введення користувача, сповіщення та інші види спілкування;

- адаптивність і реагування на розмір екрану. Тестувальники перевіряють, як інтерфейс пристосовується до різних розмірів екранів та орієнтацій пристроїв (портретна або ландшафтна);

- браузерна та платформена сумісність. Важливо переконатися, що інтерфейс працює на різних браузерах та платформах (наприклад, на різних операційних системах);

- тестування взаємодії з різними роздільними здатностями. Перевірка інтерфейсу з різними розширеннями екрану, а також на різних пристроях, включаючи мобільні та планшетні пристрої;

- автоматизація тестування інтерфейсу. Для ефективності, деякі аспекти тестування інтерфейсу можуть бути автоматизовані за допомогою спеціалізованих інструментів.

Стратегія тестування продуктивності спрямована на оцінку та перевірку рівня продуктивності, швидкості та завантаженості ПЗ в реальних умовах використання. Основні аспекти цієї стратегії включають:

- визначення цілей продуктивності. Спочатку необхідно визначити метрики продуктивності, які важливі для даного ПЗ. Це можуть бути час відгуку системи, пропускна здатність, завантаження серверів тощо;

- створення навантаження. Створюється навантаження, яке моделює реальні умови використання ПЗ. Наприклад, одночасний доступ багатьох користувачів, обробку великої кількості даних та інші аспекти, які можуть вплинути на продуктивність;

- виконання тестів навантаження. Тестувальники виконують навантаження на систему і вимірюють реакцію ПЗ. Спостерігають за часом відгуку, швидкістю операцій та реакцією системи на максимальне навантаження;

- аналіз результатів. Після виконання тестів аналізуються їх результати, порівнюються з встановленими метриками продуктивності, щоб виявити

проблєми з продуктивністю, такі як граничні навантаження, затримки або несправності;

- оптимізація та усунення проблєм. Якщо в ході тестування виявляються проблєми з продуктивністю, розробники та інженери мають вжити заходів для їх усунення. Наприклад, оптимізацію коду, збільшення ресурсів серверів та інші заходи;

- регулярне тестування продуктивності. Тестування продуктивності повинно бути регулярним процесом, оскільки продукт може змінюватися з часом. Навантаження та метрики продуктивності повинні оновлюватися для відображення актуальних умов;

- скейлінг-тестування. Тестувальники можуть виконувати тестування на різних рівнях навантаження, включаючи тести зі збільшенням кількості користувачів або об'єму даних, щоб оцінити, як система реагує на різні обставини.

Стратегія тестування безпеки спрямована на ідентифікацію потенційних вразливостей та захисних вад у програмному забезпеченні з метою забезпечення захисту від зловмисних атак і несанкціонованого доступу. Основні аспекти цієї стратегії включають:

- аналіз потенційних загроз – визначення потенційних загроз та ризиків для ПЗ, включаючи атаки ззовні, зловмисних користувачів і внутрішні загрози;

- планування тестування – розроблення плану тестування безпеки, який включає в себе визначення обсягу та об'єктів тестування, методології та інструментів, які будуть використовуватися;

- ідентифікація вразливостей – тестувальники намагаються знайти вразливості в програмному забезпеченні, такі як недоліки в захисті, помилки конфігурації і інші аспекти, які можуть бути використані для атак;

- сканування безпеки – використання спеціалізованих інструментів для сканування ПЗ на предмет вразливостей та автоматичного виявлення можливих проблєм безпеки;

- тестування атаками – проведення реальних атак на ПЗ з метою перевірки його стійкості та можливості відновлення після атак;

- аналіз результатів – оцінка результатів тестування, виявлення вразливостей та розробка плану їх усунення;

- усунення вразливостей – розробка та впровадження заходів безпеки для усунення виявлених вразливостей;

- перевірка відповідності стандартам безпеки – перевірка, що ПЗ відповідає вимогам до безпеки, включаючи стандарти та рекомендації;

- тестування обмежень доступу – перевірка, як обмеження доступу впливають на безпеку ПЗ та на його захист від несанкціонованого доступу.

Стратегія тестування відмовостійкості (resilience testing) спрямована на оцінку та перевірку того, наскільки добре ПЗ або система може відновлюватися та продовжувати надавати свої послуги після виникнення помилок, відмов, атак або незвичайних ситуацій. Основні аспекти цієї стратегії:

- аналіз вразливостей – визначення потенційних вразливостей та точок відмови в системі, які можуть викликати її некоректну роботу або відмови;

- планування тестування – розроблення плану тестування відмовостійкості, включаючи визначення сценаріїв тестування та обсягу випробувань;

- симуляція відмов – проведення тестів, які спрямовані на симуляцію різних видів відмов та незвичайних ситуацій. Наприклад, симуляцію атак, відмов обладнання, навантаження та інше;

- аналіз відновлення – оцінка, наскільки швидко та ефективно система відновлюється після виникнення відмови або атаки. Це включає відновлення послуг для користувачів та відновлення нормальної роботи системи;

- перевірка ефективності захисних заходів – перевіряється, що захисні заходи, включаючи засоби захисту, резервне копіювання і системи моніторингу, працюють як очікується та допомагають відновити систему;

- оцінка витрат на відновлення – визначення витрат, необхідних для відновлення системи та даних після відмови;

- аналіз результатів – оцінка результатів тестування відмовостійкості та ідентифікація проблєм, виявлених під час випробувань;

- удосконалення та покращення – розробка та впровадження заходів для покращення відмовостійкості системи на основі виявлених проблєм.

Стратегія тестування відповідності стандартам та вимогам (compliance testing) спрямована на перевірку того, чи відповідає ПЗ чи система встановленим стандартам, нормативам та вимогам Основні аспекти цієї стратегії включають:

- аналіз стандартів і вимог – ознайомлення з усіма вимогами, стандартами та нормативами, якими повинне відповідати ПЗ чи система;

- планування тестування – розроблення плану тестування, включаючи визначення, які стандарти та вимоги мають бути перевірені, і створення тестових сценаріїв;

- виконання тестування – проведення тестування, яке спрямоване на перевірку відповідності ПЗ встановленим стандартам і вимогам;

- запис результатів – документування результатів тестування, включаючи інформацію про виявлені відхилення від стандартів;

- аналіз відхилень – оцінка виявлених відхилень та невідповідностей і визначення, чи потрібні корективні заходи;

- усунення відхилень – розробка та впровадження заходів для усунення виявлених відхилень та досягнення відповідності стандартам та вимогам;

- підтвердження відповідності – після усунення відхилень проведення повторних тестів для підтвердження відповідності ПЗ стандартам.

2.3.Метрики оцінювання результатів тестування «чорного ящика»

У табл. 2.7 представлені типові метрики, які дозволяють оцінити результати тестування, визначити ефективність кожного методу тестування, виявити його переваги та недоліки в конкретному контексті розробки ПЗ [50].

Таблиця 2.7 – Метрики оцінювання результатів тестування «чорного ящика»

Метрика	Що вимірює	Коментар
Функціональне тестування		
Покриття функцій	Яка частина функціональності програми була протестована	Високе покриття свідчить про ретельне тестування, але не гарантує відсутності помилок
Кількість виявлених дефектів	Кількість помилок, знайдених під час тестування	Велика кількість дефектів може вказувати на недоліки в програмі
Тестування інтерфейсу користувача		
Час виконання завдань користувача	Час, потрібний для виконання завдань через інтерфейс	Зменшення часу вказує на покращення зручності користування
Кількість помилок користувача	Кількість помилок, які роблять користувачі під час взаємодії з інтерфейсом	Велика кількість помилок може свідчити про складність інтерфейсу
Тестування продуктивності		
Час відгуку системи	Швидкість реакції програми на запити користувачів або навантаження	Швидкий час відгуку вказує на продуктивність
Кількість помилок завдань на навантаження	Кількість помилок, які виникають під час навантаження	Багато помилок може вказувати на проблєми з продуктивністю
Тестування безпеки		
Кількість виявлених вразливостей	Кількість потенційних вразливостей, знайдених під час тестування	Багато вразливостей може свідчити про недостатню безпеку
Час виявлення та виправлення вразливостей	Швидкість виявлення та виправлення вразливостей	Швидка реакція важлива для запобігання потенційним атакам
Тестування відмовостійкості		
Тривалість відновлення	Час, потрібний для відновлення системи після відмови	Швидке відновлення важливе для підтримання доступності
Кількість відмов під час навантаження	Кількість відмов, які виникають під час навантаження	Багато відмов може свідчити про проблєми з відмовостійкістю

Розуміння того, що використаний метод тестування «чорного ящика» не виявив всі проблєми, може бути важливим для поліпшення стратегії тестування. Ось кілька способів, які можуть вказувати на недоліки методу тестування:

- невідомі проблєми користувача. Метрики можуть вказати на певну кількість помилок, які були виявлені під час тестування, але це не означає, що всі проблєми, з якими стикаються користувачі, були виявлені. Іноді користувачі

можуть використовувати програму або систему навіть неправильно, що може призводити до помилок, які не були передбачені тестувальниками;

- помилки, які з'являються під навантаженням. Деякі методи тестування «чорного ящика» можуть не виявляти проблєми, які з'являються лише під час великого навантаження. Метрика продуктивності може вказувати на швидкість відгуку системи, але вона може не показати, як система буде себе вести під час справжнього навантаження;

- внутрішні помилки програми. Деякі проблєми можуть бути пов'язані з внутрішніми алгоритмами або структурами програми, і вони можуть залишитися невиявленими методами тестування «чорного ящика», які перевіряють зовнішню функціональність. Метрики можуть показувати, що функціональність працює правильно, але не розкривати деталей про те, як це виконується;

Наступна табл. 2.8 узагальнює відомості стосовно можливих недоліків у методах тестування «чорного ящика».

Таблиця 2.8 – Можливі недоліки у методах тестування «чорного ящика»

Тип недоліку	Характеристика	Спосіб виявлення	Рекомендації щодо покращення
Невідомі проблєми користувача	Помилки, які виявляють користувачі, але не виявлені під час тестування	Аналіз звітів користувачів, моніторинг реального використання	Використання методів тестування на основі реального використання і зворотного зв'язку від користувачів
Помилки під навантаженням	Проблєми, які з'являються лише під час великого навантаження	Тестування продуктивності під реальними умовами	Використання тестування продуктивності та стрес-тестування для виявлення проблєм продуктивності
Внутрішні помилки програми	Проблєми з внутрішніми алгоритмами або структурами програми	Аналіз коду та структури програми	Застосування методів "білого ящика" для аналізу внутрішньої структури та алгоритмів програми
Неправильне спілкування з іншими системами	Проблєми у взаємодії з іншими системами або службами	Тестування інтеграції та взаємодії з іншими системами	Використання інтеграційного тестування для перевірки взаємодії з іншими системами

- неправильне спілкування з іншими системами. Якщо програма взаємодіє з іншими системами або службами, методи тестування «чорного ящика» можуть не виявити проблєми, пов'язані з неправильним обміном даними або некоректними запитами до інших систем.

Для виявлення цих недоліків може бути корисним поєднання різних методів тестування, включаючи методи «білого ящика» (які аналізують внутрішню структуру програми) та методи тестування безпеки. Важливо також здійснювати моніторинг реального використання програми та вивчати звіти від користувачів для виявлення проблєм, які можуть не бути виявлені тестуванням.

Табл. 2.8 підтверджує важливість комбінування різних методів тестування, включаючи як «чорний», так і «білий» ящики, для повної та ефективної перевірки програмного продукту. Також важливо здійснювати постійний моніторинг та аналіз реального використання продукту для виявлення та виправлення можливих проблєм.

2.4 Програмне забезпечення тестування «чорного ящика»

Розглянемо варіанти ПЗ, що використовується у різних випадках.

Функціональне тестування використовує:

- тестувальні фреймворки, наприклад, Selenium для тестування вебдодатків, Appium для мобільних додатків;

- інструменти автоматизації тестування, наприклад, Junit [51], TestNG [52], PyTest [53] для автоматизованого тестування функціональності;

Тестування інтерфейсу користувача (UI тестування) використовує:

- UI-тестувальні фреймворки, наприклад, Protractor [54] для вебдодатків Angular [55], Espresso [56] для мобільних додатків;

- інструменти для запису дій користувача, наприклад, Selenium IDE [57] для створення тестів шляхом запису дій користувача.

Тестування продуктивності:

- інструменти для навантаження та моніторингу, наприклад, Apache JMeter [43] для тестування навантаження серверів, New Relic [58] для моніторингу продуктивності;

Табл. 2.9 містить приклади ПЗ для різних класів тестування «чорного ящика».

Таблиця 2.9 – Програмне забезпечення для різних видів тестування «чорного ящика»

Клас тестування	Інструменти/технології	Опис/використання
Функціональне тестування	Selenium, Appium, JUnit, TestNG, PyTest, Cucumber, SpecFlow	Тестувальні фреймворки для веб та мобільних додатків; автоматизаційні інструменти та BDD-інструменти.
Тестування інтерфейсу користувача (UI)	Protractor, Espresso, Selenium IDE, Katalon Studio, UFT (раніше QTP)	UI-тестувальні фреймворки та інструменти для запису дій користувача; універсальні інструменти для автоматизації тестування UI.
Тестування продуктивності	Apache JMeter, New Relic, LoadRunner, Gatling, BlazeMeter	Інструменти для навантаження, стрес-тестування та моніторингу продуктивності; підтримка різних протоколів та сервісів.
Тестування безпеки	Burp Suite, Wireshark, OWASP ZAP, Nessus, SQLMap	Сканери вразливостей, інструменти для аналізу мережевого трафіку, автоматизоване тестування безпеки.
Тестування відмовостійкості	Chaos Monkey, Gremlin, Simian Army	Фреймворки та інструменти для створення тестів стійкості систем до відмов та симуляції різних видів відмов.
Тестування відповідності стандартам	OWASP ZAP, SonarQube, Veracode	Інструменти для автоматизованого тестування відповідності стандартам безпеки, код-рев'ю, аналізу якості коду.

Тестування безпеки:

- сканери вразливостей, наприклад, Burp Suite [45] для тестування вразливостей в вебдодатках;

- інструменти для перехоплення та аналізу трафіку, наприклад, Wireshark [59] для аналізу мережевого трафіку.

Тестування відмовостійкості:

- фреймворки для створення тестів відмовостійкості, як Chaos Monkey [60] для тестування стійкості систем до відмов.

Тестування відповідності стандартам:

- інструменти для автоматизованого тестування відповідності, як OWASP ZAP [61] для тестування відповідності вебдодатків стандартам безпеки.

Табл. 2.10 містить вказівки, для тестування якого типу ПЗ рекомендуються конкретні інструменти.

Таблиця 2.10 – Рекомендовані інструменти та сфери застосування

Клас тестування	Рекомендовані інструменти та їх застосування	Рекомендації
Функціональне тестування	Selenium (вебдодатки), Appium (мобільні додатки), JUnit/TestNG/PyTest (Java/Python додатки), Cucumber/SpecFlow (BDD)	Вибирайте інструменти відповідно до технологічного стеку вашого проєкту та потреб у BDD.
Тестування інтерфейсу користувача (UI)	Protractor (Angular вебдодатки), Espresso (Android додатки), Selenium IDE (прості записи дій користувача), Katalon Studio/UFT (загальне UI тестування)	Вибирайте інструменти, які забезпечують гнучке тестування UI та підходять для вашої платформи.
Тестування продуктивності	Apache JMeter (навантажувальне тестування серверів), New Relic/LoadRunner/Gatling/BlazeMeter (комплексне тестування продуктивності)	Обирайте інструменти, що дозволяють моделювати реальні умови навантаження і надають детальний аналіз.
Тестування безпеки	Burp Suite (вебдодатки), Wireshark (мережевий аналіз), OWASP ZAP/Nessus/SQLMap (виявлення вразливостей)	Обирайте інструменти для детального аналізу вразливостей та симуляції атак.
Тестування відмовостійкості	Chaos Monkey (тестування стійкості систем до відмов), Gremlin (симуляція різних відмов)	Використовуйте для тестування надійності в критичних системах, де важлива стабільність роботи.
Тестування відповідності стандартам	OWASP ZAP (вебдодатки та безпека), SonarQube (якість коду), Veracode (безпека додатків)	Використовуйте для забезпечення відповідності програмних продуктів стандартам безпеки та якості коду.

Загальні рекомендації щодо вибору ПЗ для тестування «чорного ящика» залежать від виду тестування, і полягають у наступному.

Функціональне тестування: Selenium для вебдодатків, Appium [62] для мобільних додатків, JUnit або TestNG для Java-додатків, PyTest для Python-додатків. Слід вибирати інструмент, який найкраще підходить до технологічного стеку проєкту. Для проєктів, що вимагають BDD (Behavior-Driven Development), краще підходять Cucumber [63] або SpecFlow [64].

Тестування інтерфейсу користувача (UI): Protractor для Angular-додатків, Espresso для Android-додатків, Selenium IDE для простих записів дій користувача. Якщо тестування UI важливе для продукту, слід обирати інструменти, які забезпечують широкі можливості для взаємодії з інтерфейсом та його аналізу.

Тестування продуктивності: Apache JMeter для навантажувального тестування, New Relic для моніторингу продуктивності. Слід обирати інструменти, які можуть моделювати реальні умови навантаження та надають детальний аналіз продуктивності.

Тестування безпеки: Burp Suite для тестування вебдодатків, Wireshark для аналізу мережевого трафіку. Важливо обрати інструменти, які надають глибокий аналіз потенційних вразливостей і дозволяють моделювати різні типи атак.

Тестування відмовостійкості: Chaos Monkey для симуляції відмов. Слід використовувати такі інструменти в середовищах з високими вимогами до надійності і доступності, як-то критичні системи чи високонавантажені сервіси.

Тестування відповідності стандартам: OWASP ZAP для перевірки вебдодатків на відповідність стандартам безпеки. Цей тип тестування особливо важливий для вебдодатків, які зберігають чи обробляють конфіденційну інформацію.

Загалом, під час вибору ПЗ для тестування «чорного ящика» важливо враховувати специфіку проєкту, технологічний стек, а також ресурси та компетенції команди. Іноді ефективне тестування вимагає поєднання декількох інструментів для досягнення кращих результатів. Також, при виборі

інструментів слід враховувати такі аспекти, як легкість інтеграції з існуючим середовищем розробки, підтримка спільноти, можливості для масштабування та автоматизації. Крім того, важливо оцінювати можливість адаптації інструментів до змінюваних вимог проєкту та специфіки ринку.

Висновки до розділу 2

1. Методи тестування «чорного ящика» класифіковані на основі їх цілей та підходів. Основні класи включають функціональне тестування, тестування інтерфейсу користувача, тестування продуктивності, тестування безпеки, тестування відмовостійкості та тестування сумісності. Кожен клас має свої особливості та області застосування, які були виявлені в ході аналізу.

2. Проаналізовано переваги та недоліки методів тестування. Встановлено, що функціональне тестування ефективне для перевірки відповідності функціональних вимог, але не здатне виявити внутрішні помилки. Тестування продуктивності важливе для визначення швидкодії систем, але може бути ресурсномісткім. Зроблено висновок про необхідність вибору методу тестування відповідно до потреб проєкту.

3. Проаналізовано стратегії та стандарти, які застосовуються в тестуванні «чорного ящика», зокрема підходи до вибору та використання різних методів відповідно до потреб проєкту.

4. Систематизовано та описано метрики, які допомагають оцінити результати тестування. Зокрема, такі як покриття функцій, час відгуку системи, кількість виявлених дефектів тощо.

5. Виконано огляд та систематизацію ПЗ, що використовується для тестування «чорного ящика». Описано основні інструменти,їх особливості, як вони можуть бути застосовані в різних сценаріях тестування. Запропоновано рекомендації щодо вибору методів та інструментів тестування «чорного ящика» в залежності від конкретних цілей та вимог проєкту.

РОЗДІЛ 3
ПРАКТИЧНІ АСПЕКТИ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ «ЧОРНОГО ЯЩИКА»

3.1 Експеримент у тестуванні програмного забезпечення

У літературних джерелах з тестування ПЗ підходи до поняття «експеримент» можуть розрізнятися, але загальною є ідея систематичного тестування для збору даних і оцінки якості програми. Поняття «експеримент» в тестуванні ПЗ відображає ідею контрольованого і докладного дослідження для забезпечення якості та надійності програми перед її впровадженням або після нього. Експерименти у тестуванні можуть включати запуск тестових сценаріїв, введення різних навантажень, аналіз результатів та оцінку даних про продукт. Вони спрямовані на виявлення дефектів, оцінку якості та визначення того, наскільки ПЗ відповідає встановленим критеріям. Мета експерименту – отримання об'єктивних та надійних даних, які допоможуть виявити проблєми та дефекти ПЗ.

Поняття «експеримент» у тестуванні ПЗ визначається, як процес, що включає в себе систематичне планування та проведення тестових активностей для збору даних про продукт з метою оцінки його якості та надійності [65]. Тобто, у тестуванні ПЗ «експеримент» – це процес систематичного дослідження або перевірки ПЗ з метою отримання об'єктивних даних про його функціональність, надійність, продуктивність або інші характеристики. Експеримент включає такі основні етапи:

- планування експерименту – визначення мети експерименту, розробка тестових сценаріїв, вибір тестових кейсів та визначення критеріїв оцінки;
- проведення експерименту – виконання тестових сценаріїв, введення навантажень на систему та генерація тестових даних;
- аналіз результатів – оцінка результатів тестування, виявлення дефектів та аномалій, збір даних про продукт;

- звіт про експеримент – підготовка звіту, що включає в себе інформацію про виявлені дефекти, оцінку продукту та рекомендації щодо подальших дій.

Поняття «експеримент» у тестуванні ПЗ також розглядається як процес систематичного використання тестових кейсів та сценаріїв для перевірки функціональності програми з метою виявлення дефектів та аномалій [66]. З цього погляду, експеримент у тестуванні ПЗ передбачає наступну послідовність дій:

- проведення тестових експериментів – виконання конкретних послідовностей дій, що є частиною тестових сценаріїв, для перевірки, чи працює програма відповідно до очікувань;

- збір та аналіз результатів – фіксування результатів тестів, виявлення дефектів, які можуть бути відзначені під час експерименту, і збір інформації про продуктивність програми;

- порівняння з очікуваннями – порівняння фактичних результатів експерименту з очікуваними результатами, які визначаються на етапі планування тестування;

- узагальнення результатів – вибірковий аналіз результатів тестування та їх узагальнення до всього програмного продукту.

Також, поняття «експеримент у тестуванні» трактується як систематичний процес перевірки та валідації ПЗ з метою виявлення помилок і дефектів. В цьому розумінні експеримент означає набір конкретних тестових дій та вхідних даних, які застосовуються до програми для оцінки її роботи та виявлення недоліків [67]. Такий підхід передбачає виконання різноманітних тестів, які включають у себе різні види тестових сценаріїв та тестові дані Експерименти можуть бути проведені для перевірки різних аспектів програми, таких як правильність виконання функцій, продуктивність, надійність та безпека.

Під час проведення експерименту у тестуванні ПЗ важливо систематично фіксувати результати, аналізувати їх та спостерігати за виникненням дефектів. Такий підхід допомагає виявити проблєми та недоліки програми. Розглядаються різні методи та техніки тестування, які можна використовувати в експериментах для забезпечення якості ПЗ.

3.2 Методологія тестування «чорного ящика»

Методологія тестування «чорного ящика» описує систематичний підхід до планування, проведення і аналізу тестових експериментів для оцінки якості ПЗ, що визначає правила та процедури для проведення тестів і дослідження різних аспектів програми без знання її внутрішньої реалізації.

Загальний опис методології експерименту з тестування ПЗ «чорного ящика» представлений у табл. 3.1.

Таблиця 3.1 – Етапи експерименту з тестування ПЗ «чорного ящика»

Етап	Дії та кроки
Визначення цілей	Аналіз вимог, формулювання мети експерименту, специфікація критеріїв успіху.
Планування експерименту	Визначення тестових сценаріїв, тестових даних, методики тестування, учасників, налаштування тестового середовища.
Підготовка	Запуск вебсайту, підготовка бази даних, створення тестових облікових записів, заповнення тестових даних.
Виконання експерименту	Визначення тестових сценаріїв, підготовка та виконання тестових сценаріїв, фіксація кожного кроку та результатів.
Аналіз результатів	Відновлення даних, порівняння з критеріями успіху, визначення помилок, оцінка серйозності, підготовка звіту.
Підготовка звіту	Опис виявлених проблєм, визначення причин, розробка рекомендацій, підготовка документації та звіту про результати.

Основні етапи проведення експерименту у тестуванні «чорного ящика» включають:

1. Визначення чітких цілей експерименту, таких як перевірка певної функціональності, виявлення помилок, оцінка продуктивності тощо;

2. Розроблення плану експерименту, зокрема, вибір тестових сценаріїв, тестових даних та встановлення критеріїв успіху;

3. Підготовка тестового середовища, збір і підготовка тестових даних, а також вибір інструментів для тестування;

4. Виконання тестових сценаріїв та збір результатів експерименту;

5. Аналіз результатів, ідентифікація помилок, оцінка відповідності результатів цілям експерименту;

6. Підготовка звіту про результати експерименту, включаючи виявлені проблєми, рекомендації щодо виправлення дефектів та покращення якості програми.

Наприклад, методологія визначення цілей експерименту у тестуванні функціональності реєстрації на вебсайті включає наступні кроки:

1. Аналіз вимог та очікувань – аналіз вимог до функціональності реєстрації на вебсайті, таких як вимоги з технічної документації, специфікацій або вимоги, що встановлюються замовником. Важливо враховувати очікування користувачів щодо реєстрації на вебсайті, які можуть бути встановлені на основі досвіду користувачів аналогічних ресурсів або внутрішніх стандартів;

2. Формулювання основної мети – визначення основної мети експерименту. У даному випадку, основною метою є перевірка функціональності реєстрації на вебсайті з точки зору відповідності вимогам і задоволення очікувань користувачів;

3. Специфікація критеріїв успіху – розробка конкретних критеріїв, які дозволять визначити, чи була досягнута основна мета експерименту. Приклади критеріїв успіху можуть включати: можливість успішної реєстрації на сайті, надходження підтвердження реєстрації на електронну пошту, відсутність помилок або відповідність реєстраційного процесу зазначеним вимогам;

4. Планування експерименту – розробка плану експерименту, який включає в себе опис тестових сценаріїв, обрання тестових даних і методику тестування.

Методологія планування експерименту у тестуванні функціональності реєстрації на вебсайті включає наступні кроки:

1. Визначення тестових сценаріїв – розробляємо тестові сценарії, які охоплюють різні аспекти функціональності реєстрації, включаючи різні можливі варіанти введення даних (наприклад, коректні та некоректні дані), різні сценарії помилок, а також процес підтвердження реєстрації;

2. Визначення тестових даних – визначаємо тестові дані, які будуть використовуватися під час тестування. Це може включати введення реальних або

симульованих даних користувачів, таких як ім'я, електронна пошта, пароль й таке ін.;

3. Методика тестування – описання методики тестування, включаючи послідовність дій, які тестувальник повинен виконати під час тестування; визначення очікуваних результатів для кожного тестового сценарію;

4. Визначення учасників тестування (тестувальників) – хто буде виконувати тестування. Це можуть бути внутрішні члени команди розробників або спеціально наймані тестувальники;

5. Налаштування тестового середовища, що відтворює умови реального використання вебсайту. Наприклад, встановлення тестового сервера або налаштування доступу до тестового сайту;

6. Виконання тестових сценаріїв – тестувальники вводять дані, виконують дії, які передбачені сценарієм, фіксують результати;

7. Аналіз результатів – оцінка результатів тестування. Виявлені помилки фіксуються і передаються для подальшого виправлення розробникам;

8. Звітність – формування звіту про проведене тестування, де віідображається інформація про виявлені помилки, результати виконаних сценаріїв та відповідність функціональності вимогам.

Методологія підготовки експерименту для тестування функціональності реєстрації на вебсайті включає наступні кроки.

Підготовка тестового середовища:

1. Запуск вебсайту – встановлення та налаштування вебсайту в тестовому середовищі. Для цього можна використовувати клон робочого середовища, тестовий сервер або локальний сервер;

2. Підготовка бази даних – створення та налаштування тестової бази даних, яка використовується в процесі реєстрації користувачів. База даних повинна бути відокремлена від робочої бази для уникнення впливу на реальні дані.

Підготовка тестових даних:

1. Створення тестових облікових записів – створення облікових записів користувачів, які будуть використовуватися під час тестування. Ці облікові

записи повинні включати різні комбінації даних, включаючи коректні та некоректні дані для реєстрації;

2. Заповнення тестових даних – введення тестових даних, які користувачі повинні вказати під час реєстрації. Це може включати ім'я, електронну пошту, пароль та інші обов'язкові та необов'язкові поля;

3. Створення тестових сценаріїв – розробка тестових сценаріїв, які використовують тестові дані для виконання реєстрації. Сценарії мають включати різні варіанти введення даних та дії користувачів;

4. Запуск тестових сценаріїв – під час тестування запускаються розроблені тестові сценарії з використанням тестових даних. Реєстрація відбувається згідно зі сценарієм, і результати фіксуються для аналізу.

Методологія виконання експерименту для тестування функціональності реєстрації на вебсайті:

1. Визначення тестових сценаріїв – визначення конкретних сценаріїв реєстрації, які будуть використовуватися під час експерименту. Наприклад, це може бути сценарій коректної реєстрації, сценарій з неправильною електронною поштою, тощо;

2. Підготовка тестових даних – відповідно до кожного сценарію готуються тестові дані, включаючи ім'я, електронну пошту, пароль та інші обов'язкові та необов'язкові дані користувача, які вимагаються під час реєстрації;

3. Виконання тестового сценарію – запуск обраного тестового сценарію з використанням підготовлених тестових даних. Важливо крок за кроком реєструвати всі взаємодії користувача з вебсайтом під час реєстрації;

4. Фіксація кожного кроку – під час виконання тестового сценарію фіксується кожен крок, який виконується на вебсайті, включаючи введення даних, натискання кнопок, переходи між сторінками тощо;

5. Фіксація результатів кожного кроку, включаючи повідомлення про помилки, якщо вони виникають, або підтвердження успішного завершення реєстрації;

6. Аналіз зафіксованих результатів для виявлення проблєм, помилок та аномалій під час реєстрації. Результати можуть порівнюватися з очікуваними результатами;

7. Повторення всіх кроків для кожного визначеного сценарію реєстрації.

Методологія аналізу результатів тестування функціональності реєстрації на вебсайті:

1. Відновлення всіх зібраних даних, включаючи фіксовані кроки, введені дані, реакцію вебсайту та реєстраційні результати для кожного тестового сценарію;

2. Порівняння отриманих результатів із попередньо визначеними критеріями успіху для кожного сценарію, що можуть включати правильність та повноту реєстраційних даних, відсутність помилок під час процесу реєстрації, швидкодію вебсайту тощо;

3. Визначення всіх помилок та несправностей, виявлених під час експерименту, для кожного тестового сценарію. Це може включати помилки при введенні даних, внутрішні помилки сервера або незрозумілі повідомлення про помилки на вебсайті;

4. Визначення пріоритету помилок відповідно до їх серйозності та можливого впливу на користувачів. Наприклад, серйозні помилки, які призводять до неможливості реєстрації, можуть бути визнані важливішими, ніж помилки відображення повідомлень;

5. Підготовка звіту та документації, яка включає всі виявлені помилки, результати порівняння з критеріями успіху та їх пріоритетами;

6. Виправлення виявлених помилок та перевірка реєстраційного процесу після виправлень для перевірки ефективності;

7 Повторення аналізу результатів для кожного сценарію після виправлень для впевненості в тому, що всі проблєми були вирішені.

Методологія підготовки звіту за результатами тестування функціональності реєстрації на вебсайті:

1. Опис виявлених проблєм – надається для кожного виявленого дефекту або проблєми з описом, фіксованими кроками для відтворення та конкретними обставинами, у яких вони виникають. Опис включає в себе інформацію, яким чином проблєма впливає на користувачів та функціональність вебсайту;

2. Визначення можливих причин виявлених проблєм та їх можливого впливу на інші частини програми або інші аспекти функціональності вебсайту;

3. Визначення серйозності кожного виявленого дефекту або проблєми, що оцінку впливу на користувачів та функціональність;

4. Розробка рекомендацій щодо виправлення виявлених дефектів та покращення якості програми. Це може включати рекомендації щодо покращення коду, внесення змін у реєстраційний процес, або удосконалення відображення повідомлень про помилки для користувачів;

5. Підготовка супровідної документації, яка включає в себе звіт за результатами тестування, список виявлених проблєм та рекомендації щодо виправлення, а також зображення результатів тестування (наприклад, скріншоти);

6. Збереження всіх документів зі звітом та виявленими проблємами для подальшого використання і виправлення дефектів.

Відповідно до описаної методології, для проведення експерименту з перевірки функціональності реєстрації на вебсайті потрібно використати наступні кроки:

1. Визначення цілей:

- визначення основної мети експерименту, яка полягає в перевірці функціональності реєстрації на вебсайті;

- визначення конкретних функціональних вимог до реєстраційного процесу.

2. Планування:

- розроблення плану тестування, включаючи опис тестових сценаріїв для реєстрації користувачів на вебсайті;

- визначення тестових даних, таких як ім'я користувача, електронна пошта, пароль, тощо, для введення під час реєстрації;
- встановлення критеріїв успіху для кожного тестового сценарію (наприклад, користувач повинен отримати підтвердження реєстрації на електронну пошту);

3. Підготовка:

- підготовка тестового середовища, включаючи запуск вебсайту та бази даних, необхідних для реєстрації;
- підготовка тестових даних, включаючи створення тестових облікових записів користувачів.

4. Виконання:

- виконання тестових сценаріїв реєстрації з використанням підготовлених тестових даних;
- фіксація кожного кроку виконання тестового сценарію та результатів.

5. Аналіз:

- аналіз результатів тестування для кожного тестового сценарію;
- виявлення помилок та несправностей у реєстраційному процесі;
- порівняння отриманих результатів із визначеними критеріями успіху.

6. Звітність:

- підготовка звіту про результати експерименту;
- у звіті вказуються виявлені проблєми та дефекти реєстраційного процесу, а також рекомендації щодо їх виправлення.

3.3 Вибір програмного забезпечення

Тестування ПЗ «чорного ящика» передбачає перевірку функціональності, без знання внутрішньої структури коду. Для здійснення такого тестування використовують спеціальне програмне забезпечення. При виборі ПЗ для тестування «чорного ящика», слід враховувати такі ключові моменти:

- функціональні можливості. Вибране ПЗ повинно відповідати потребам вашого проєкту. Слід визначити, які саме функціональні можливості необхідні для тестування вашого програмного продукту, і знайдіть інструмент, який ці вимоги задовольняє;

- сумісність. Слід переконатися, що вибране ПЗ сумісне з тими технологіями, мовами програмування та платформами, які використовуються у вашому проєкті;

- легкість використання. ПЗ для тестування «чорного ящика» повинно бути досить інтуїтивним і легким у використанні, оскільки в ньому працюватимуть тестувальники з різними рівнями кваліфікації;

- можливості автоматизації. Якщо планується автоматизувати тестування, слід переконатись, що вибране ПЗ підтримує автоматизацію та інтеграцію з іншими інструментами;

- звітність та аналітика. ПЗ повинно надавати засоби для створення звітів та аналізу результатів тестування;

- підтримка та спільнота користувачів. Слід переконатись, що є доступ до технічної підтримки та можливість обміну досвідом з іншими користувачами ПЗ;

- вартість. Слід оцінити бюджет для ПЗ та знайти інструмент, який відповідає фінансовим можливостям проєкту;

- репутація постачальника ПЗ. Перед вибором ПЗ слід оцінити репутацію постачальника та перевірити відгуки від інших користувачів;

- документація та навчання. Слід переконатись, що є відповідна документація та навчальні матеріали для вивчення ПЗ ;

- оновлення та підтримка. Слід перевірити, чи регулярно оновлюється ПЗ і чи надається технічна підтримка.

Ретельне вивчення цих аспектів дозволяє зробити інформований вибір ПЗ для тестування «чорного ящика», яке відповідає потребам та вимогам проєкту (табл. 3.2).

Таблиця 3.2 – Вибір ПЗ для тестування «чорного ящика»

Критерій	Опис
Функціональні можливості	ПЗ має відповідати потребам проєкту; важливо визначити необхідні функціональні можливості і знайти інструмент, який ці вимоги задовольняє.
Сумісність	Переконатися, що ПЗ сумісне з технологіями, мовами програмування та платформами, які використовуються у проєкті.
Легкість використання	ПЗ має бути інтуїтивно зрозумілим і легким у використанні, адаптованим для тестувальників з різними рівнями кваліфікації.
Можливості автоматизації	Якщо планується автоматизація тестування, важливо, щоб ПЗ підтримувало автоматизацію та інтеграцію з іншими інструментами.
Звітність та аналітика	ПЗ має надавати засоби для створення звітів та аналізу результатів тестування.
Підтримка та спільнота	Наявність технічної підтримки та можливість обміну досвідом з іншими користувачами ПЗ.
Вартість	Оцінка бюджету для ПЗ та вибір інструменту, який відповідає фінансовим можливостям проєкту.
Репутація постачальника ПЗ	Оцінка репутації постачальника та перевірка відгуків від інших користувачів.
Документація та навчання	Наявність відповідної документації та навчальних матеріалів для вивчення ПЗ.
Оновлення та підтримка	Перевірка регулярності оновлень ПЗ і наявності технічної підтримки.

Розглянемо процедуру вибору та використання ПЗ для проведення експерименту у тестуванні функціональності реєстрації на вебсайті, як «чорного ящика». Основні кроки полягають у наступному:

1. Визначення цілей експерименту:

- визначити, що саме потрібно перевірити під час експерименту. У даному випадку, це функціональність реєстрації на вебсайті;
- визначити конкретні питання, на які потрібно знайти відповіді шляхом тестування.

2. Вибір критеріїв та показників – вибрати критерії успіху для експерименту, наприклад, час реєстрації, правильність обробки даних, повідомлення про успішну реєстрацію тощо;

3. Вибір інструментів та технологій. Проведення експерименту у тестуванні «чорного ящика» вимагає вибору тестового автоматизованого інструменту або скрипту. Популярні інструменти Selenium, JUnit, TestNG, або інші, залежно від вибору мови програмування. Необхідно перевірити сумісність

інструменту з вашим вебдодатком та технологіями, що використовуються в ньому, а також вибраними критеріями успіху;

4. Створення тестових сценаріїв – тестові сценарії для перевірки функціональності реєстрації повинні відображати типові дії користувача під час реєстрації на вашому вебсайті;

5. Підготовка тестових даних – тестові дані, як ім'я, адреса електронної пошти, пароль тощо будуть використовуватися під час реєстрації;

6. Виконання тестових сценаріїв:

- запуск тестового сценарію на вебсайті за допомогою обраного тестового інструменту;
- фіксування результаті тестів, як час виконання та будь-які помилки чи недоліки;

7. Аналіз результатів:

- порівняння результатів тестування з визначеними критеріями успіху;
- визначення, чи функціональність реєстрації вебсайту відповідає очікуванням;

8. Підготовка звіту, в якому описані результати тестування, виявлені проблєми та рекомендації щодо виправлення дефектів чи покращення функціональності;

9. Виправлення та повторення:

- в разі виявлення помилок, вони передаються розробникам для виправлення;
- повторне тестування після виправлення помилок.

Для перевірки функціональності реєстрації на вебсайті в експерименті «чорного ящика», слід формулювати конкретні запитання, на які потрібна відповідь.

У табл. 3.3 узагальнено відомості про перевірку функціональності реєстрації на вебсайті в експерименті «чорного ящика».

Таблиця 3.3 – Відомості про перевірку функціональності реєстрації на вебсайті в експерименті «чорного ящика»

Запитання для перевірки	Критерії та показники для оцінки
Чи можливо успішно зареєструватися на вебсайті?	Успішна реєстрація, загальна задоволеність користувачів
Чи правильно валідуються обов'язкові поля реєстрації?	Перевірка обов'язкових полів, повідомлення про помилки
Чи відбувається перевірка унікальності імені користувача або email?	Унікальність облікового запису
Чи призводять неправильні дані до помилок?	Обробка некоректних даних
Чи призводять правильні дані до успішної реєстрації?	Час реєстрації, підтвердження реєстрації
Чи відбувається надсилання підтвердження реєстрації на email?	Підтвердження реєстрації
Чи виникають помилки, які не призводять до збоїв системи?	Стійкість до атак, повідомлення про помилки

Табл. 3.3 включає запитання, які потрібно розглянути під час тестування, та відповідні критерії та показники для оцінки ефективності і безпеки функціональності реєстрації.

3.4 Інструменти автоматизації тестування

Selenium, JUnit та TestNG – це приклади інструментів, які використовуються для автоматизації тестування ПЗ, але вони призначені для різних цілей та мають свої особливості.

Розглянуті інструменти мають безкоштовні версії або плани, які можна використовувати для базового тестування, включаючи Selenium, TestCafe, Postman, JMeter, і Cypress. Selenium є безкоштовним програмним забезпеченням [57]. TestCafe надає безкоштовну версію для використання, що є досить потужною для більшості проєктів [66]. Postman також має безкоштовну версію з можливістю тестування API [67]. Apache JMeter є вільним та відкритим програмним забезпеченням [43]. Cypress також має безкоштовну версію для використання [68].

У табл. 3.4 представлений огляд інструментів для тестування вебзастосунків.

Таблиця 3.4 – Інструменти для тестування вебсайтів

Інструмент	Призначення	Функціональність	Особливості	Рекомендації
Selenium	Автоматизація тестування	Взаємодія з вебелементами	Вибір мови програмування	Розширення та структурування тестів
TestCafe	Автоматизація без розширень	Тестування в різних браузерах	Не потребує додаткового ПЗ	Якщо не хочете встановлювати розширення
Postman	Тестування API	Надсилання URL: HTTP-запитів	Ідеально для тестування API	Для тестування реєстраційних API
JMeter	Тестування продуктивності	Створення та аналіз навантажень	Визначення максимального навантаження	Перевірка продуктивності реєстраційних сторінок
Cypress	Автоматизація тестування фронтенду	Перевірка взаємодії з вебелементами	Інтеграція з різними фреймворками	Тестування реєстраційних форм

Автоматизоване тестування за допомогою інструментів, таких як Selenium, TestCafe, Postman, JMeter і Cypress, виконується шляхом створення тестових скриптів або сценаріїв, які описують послідовність кроків, які має виконати тест. Розробка тестових сценаріїв – це процес, що вимагає уважності та докладності, оскільки від якості тестових сценаріїв залежить ефективність та надійність автоматизованого тестування.

Загальна процедура і алгоритм автоматизованого тестування:

1. Визначення цілей тестування, що саме потрібно перевірити або протестувати. У випадку функціонального тестування реєстрації це може бути, наприклад, перевірка, чи працює процес реєстрації користувача на вашому вебсайті належним чином;

2. Вибір інструменту для автоматизації тестів Наприклад, можна використовувати Selenium для вебінтерфейсу, Postman для API, JMeter для навантаження, і т.д.;

3. Створення тестових скриптів або сценаріїв для перевірки бажаної функціональності – написання коду або скриптів, які відтворюють послідовність кроків, які користувач виконує на вебсайті або іншій системі;

4. Конфігурація і налаштування – тестових скриптів і інструментів відповідно до вебсайту або додатку, включаючи введення тестових даних, URL-адреси, інструкції для запуску тесту та інше;

5. Запуск тестів – тестові скрипти запускаються за допомогою вибраного інструменту. Вони автоматично виконають послідовність дій, перевіряючи функціональність реєстрації;

6. Збір результатів. Інструменти автоматизації зазвичай збирають дані про тестування, такі як час виконання, статус тесту, помилки, інші показники проходження тестів;

7. Аналіз результатів тестування, щоб визначити, які були виявлені помилки чи несправності. Їх потрібно знайти і відправити звіт розробникам для виправлення;

8. Повторне тестування після виправлення помилок або змін у ПЗ, щоб переконатися, що вони більше не виникають;

9. Звіт про результати тестування та документація, що описує виявлені проблєми та рекомендації щодо їх виправлення;

10. Моніторинг: періодичне тестування, оновлюючи тестові скрипти при необхідності та повторюючи тестування під час регулярних випусків ПЗ.

Ця процедура може бути застосована до багатьох видів тестування, включаючи функціональне, навантаження, безпеку та інші

Розробка тестових сценаріїв є ключовою частиною процесу автоматизованого тестування. Вона включає в себе створення послідовності дій або інструкцій, які тестовий скрипт повинен виконувати для перевірки функціональності ПЗ. Загальний підхід до розроблення тестових сценаріїв включає наступні кроки:

1. Аналіз вимог до ПЗ: ретельно вивчить документацію, специфікації та будь-які інші матеріали, які описують очікувану функціональність;

2. Визначення цілей тестування: визначте конкретні цілі для кожного тестового сценарію. Які результати ви очікуєте від тесту?

3. Сценарій дій: створіть послідовність дій, які тестовий скрипт повинен виконувати, що включає кроки, які користувач виконує, щоб взаємодіяти з ПЗ. Кожен крок повинен бути якомога більш деталізованим та однозначним;

4 Тестові дані: визначте, які тестові дані необхідні для виконання тесту. Наприклад, це можуть бути вхідні дані для тесту або значення, які необхідно перевірити після виконання тестування;

5. Очікувані результати: визначте очікувані результати для кожного кроку та кінцевий результат тестового сценарію. Якщо ПЗ працює належним чином, що ви очікуєте побачити?

6. Завдання для автоматизації: визначте, які кроки тестового сценарію можуть бути автоматизовані. Це завдання може бути виконано за допомогою скриптів або іншого ПЗ для автоматизації тестування;

7. Реалізація: реалізуйте автоматизований тестовий скрипт з урахуванням всіх розроблених кроків та вимог до тестування;

8. Валідація та налагодження: перевірте та налаштуйте тестовий скрипт, щоб переконатись, що він працює належним чином і дає очікувані результати;

9. Збереження та управління: збережіть та ведіть облік всіх тестових сценаріїв, додавайте, оновлюйте і видаляйте їх за потреби;

10. Повторне використання: проєктуйте тестові сценарії з метою повторного використання. Це допоможе зекономити час та зусилля у майбутньому;

11. Документація: підготуйте документацію, яка описує кожен тестовий сценарій, його призначення та використання;

12. Підтримка: надавайте підтримку іншим членам команди тестування та розробки, що використовують тестові сценарії;

13. Оновлення: оновлюйте тестові сценарії при змінах в програмному забезпеченні або вимогах;

14. Моніторинг: здійснюйте моніторинг результатів тестування та налаштовуйте тестові сценарії при необхідності.

Табл. 3.5 описує процес розробки автоматизованих тестових сценаріїв, від аналізу вимог до моніторингу результатів тестування.

Таблиця 3.5 – Узагальнення процедури розробки тестових сценаріїв

Крок	Опис
Аналіз вимог до ПЗ	Вивчення документації та специфікацій для розуміння очікуваної функціональності.
Визначення цілей тестування	Встановлення конкретних цілей та очікуваних результатів для кожного тестового сценарію.
Сценарій дій	Створення детальної послідовності дій для тестового скрипта.
Тестові дані	Визначення необхідних тестових даних.
Очікувані результати	Визначення результатів, які повинен показати тест при коректній роботі ПЗ.
Завдання для автоматизації	Вибір кроків сценарію, які підлягають автоматизації.
Реалізація	Розробка автоматизованого тестового скрипта.
Валідація та налагодження	Перевірка і налаштування скрипта для забезпечення його коректної роботи.
Збереження та управління	Управління тестовими сценаріями, включаючи їх збереження, оновлення та видалення.
Повторне використання	Розробка сценаріїв із можливістю їх повторного використання.
Документація	Підготовка опису кожного тестового сценарію.
Підтримка	Надання допомоги команді з використанням тестових сценаріїв.
Оновлення	Оновлення сценаріїв у відповідності до змін в ПЗ або вимогах.
Моніторинг	Спостереження за результатами тестування та необхідність у налаштуванні сценаріїв.

Розглянемо процедуру розробки тестового сценарію на прикладі тестування функціональності реєстрації на вебсайті.

Крок 1. Розуміння функціональності. Першим кроком є ретельне вивчення функціональності реєстрації на вебсайті, поглиблене вивчення документації, вимог та специфікацій є ключовим. Потрібно зрозуміти, які дані користувач повинен ввести, які дії він повинен виконати, і які результати мають бути досягнуті після реєстрації.

Крок 2. Визначення цілей тестування. Визначте, що саме потрібно перевірити під час цього тесту. Наприклад, цілями тесту можуть бути:

- перевірка можливості користувача зареєструватися на вебсайті;

- перевірка правильності обробки введених даних;

- перевірка відправлення підтверджувального листа на пошту користувача тощо.

Крок 3. Створення послідовності дій. Розробіть послідовність дій, які користувач повинен виконати для реєстрації на вебсайті. Опишіть кожен крок докладно і однозначно. Наприклад:

(1) Перейдіть на головну сторінку вебсайту;

(2) Клацніть на кнопку «Зареєструватися»;

(3) Введіть ім'я користувача;

(4) Введіть адресу електронної пошти;

(5) Виберіть пароль і підтвердіть його;

(6) Натисніть кнопку «Зареєструватися».

Крок 4. Визначення тестових даних. Визначте, які тестові дані вам потрібні для виконання тесту. Наприклад, це можуть бути дійсні дані, які користувач вводить, або штучно створені дані для тестування.

Крок 5. Визначення очікуваних результатів. Для кожного кроку визначте, які результати ви очікуєте побачити. Наприклад, після завершення реєстрації очікується відображення особистого кабінету користувача.

Крок 6. Завдання для автоматизації. Визначте, які кроки можна автоматизувати і якими інструментами це краще робити. Наприклад, для автоматизованого тестування вебсайтів можна використовувати Selenium.

Крок 7. Реалізація тестового скрипта. Реалізуйте тестовий скрипт на основі розробленого сценарію, використовуючи вибрані інструменти для автоматизації.

Крок 8. Проведення тесту. Виконайте тестовий скрипт на вебсайті та збережіть результати виконання.

Крок 9. Аналіз результатів. Проаналізуйте результати тесту та порівняйте їх із очікуваними результатами. Визначте, чи було досягнуто цілей тестування та чи виникли помилки.

Крок 10. Звітність. Підготуйте звіт про результати тестування, з описанням виявлених проблєм та рекомендаціями щодо виправлення дефектів.

Крок 11. Підтримка та оновлення. Підтримуйте тестові сценарії та оновлюйте їх при змінах в програмному забезпеченні або вимогах.

Табл. 3.6 узагальнює процедури розробки тестового сценарію для функціональності реєстрації на вебсайті.

Таблиця 3.6 – Процедури розробки сценарію тестування функціональності реєстрації на вебсайті

Крок	Опис дій
Розуміння функціональності	Вивчення функціональності реєстрації, аналіз документації, вимог та специфікацій.
Визначення цілей тестування	Встановлення конкретних цілей тесту, наприклад, перевірка можливості реєстрації та обробки даних.
Створення послідовності дій	Розробка деталізованих кроків користувача для реєстрації на сайті.
Визначення тестових даних	Вибір дійсних або штучних даних для тестування.
Очікувані результати	Визначення очікуваних результатів кожного кроку та загального результату реєстрації.
Завдання для автоматизації	Вибір кроків для автоматизації та відповідних інструментів, наприклад, Selenium.
Реалізація тестового скрипта	Створення тестового скрипта на основі розробленого сценарію.
Проведення тесту	Виконання тестового скрипта та збереження результатів.
Аналіз результатів	Аналіз результатів та порівняння їх з очікуваними.
Звітність	Підготовка звіту про результати тестування.
Підтримка та оновлення	Підтримка та оновлення тестових сценаріїв при змінах у програмному забезпеченні або вимогах.

3.5 Методика тестування та аналіз результатів

Для виконання тестування та збору результатів були розроблені чотири алгоритми автоматизованого тестування за допомогою ПЗ Selenium з використанням мов програмування Java, Python, JavaScript та їх комбінації. Методика тестування та збір результатів тестування представлені у додатку А.

Результати проведеного експерименту з тестування вебзастосунку представлені у табл. 3.7.

Таблиця 3.7 – Узагальнені результати проведеного експерименту

Метрика	Значення	Опис
Час виконання тестів	120 хвилин	Загальний час, необхідний для виконання всіх тестових сценаріїв.
Кількість Виявлених Помилок	5 помилок	Загальна кількість помилок, виявлених під час тестування.
Кількість Успішно Пройдених Тестів	95 тестів	Кількість тестових сценаріїв, які завершилися без помилок.
Процент Успішності Тестування	95%	Відсоток тестів, які були успішно пройдені з усіх проведених.
Час Реакції Системи	0.5 секунди	Середній час реакції системи на дії користувача під час тестування.

Отримані дані демонструють, що тестування було проведено ефективно з мінімальною кількістю помилок і високим рівнем успішності. Короткий час реакції системи свідчить про високу продуктивність та оптимізацію процесів.

У табл. 3.8 представлене порівняння результатів чотирьох розроблених автоматизованих алгоритмів тестування, які використовують ПЗ Selenium із мовами програмування Java, Python, JavaScript та їх комбінацією Python+JavaScript.

Таблиця 3.8 – Порівняння автоматизованих алгоритмів тестування з використанням різних мов програмування

Мова програмування	Час виконання тестів, хв.	Кількість виявлених помилок	Кількість успішно пройдених тестів	Успішнвсть тестування, %	Час реакції системи, с
Java	110	4	96	96%	0,4
Python	120	6	94	94%	0,6
JavaScript	115	5	95	95%	0,5
Python+JavaScript	125	3	97	97%	0,3

Табл. 3.8 показує, що кожен з алгоритмів має свої переваги та недоліки залежно від використаної мови програмування та її комбінації. Комбінація Python+JavaScript показала найвищий процент успішності тестування та найкоротший час реакції системи, тоді як алгоритм на основі Python мав найбільшу кількість виявлених помилок.

Аналіз результатів проведеного експерименту з тестування програмного забезпечення за допомогою Selenium можна представити наступним чином.

Основні висновки з експерименту:

1. Чотири розроблені алгоритми автоматизованого тестування, які використовують Selenium із мовами програмування Java, Python, JavaScript та їх комбінаціями, продемонстрували високу ефективність у виявленні помилок та невідповідностей у процесі реєстрації на вебсайті.

2. За допомогою Selenium було розроблено стабільні та надійні тестові сценарії, що дозволили імітувати користувацькі дії на вебсайті та перевірити функціональність реєстрації з різних перспектив.

3. Експеримент підтвердив, що Selenium може бути легко адаптований під різні мови програмування, що надає гнучкість у виборі інструментів для тестування залежно від специфіки проекту та вимог розробників.

4.. Результати тестування виявилися точними та чіткими, забезпечуючи детальне виявлення та документування помилок. Виявлені недоліки були чітко задокументовані, що сприяло їх ефективному усуненню.

5. Підтримка та оновлення тестових сценаріїв залишаються важливими аспектами, що вимагають регулярного перегляду для забезпечення їх актуальності у відповідності до оновлень ПЗ або змін у вимогах.

Рекомендації:

1. Для підвищення ефективності тестування рекомендується подальша оптимізація та автоматизація тестових сценаріїв з використанням додаткових інструментів та бібліотек.

2. На основі отриманих результатів рекомендується розширити обсяг тестування, включивши додаткові функціональності та сценарії користувача, для більш всебічного аналізу ПЗ.

3. Рекомендується провести подальші дослідження з використанням Selenium для інших видів тестування, таких як навантажувальне та інтеграційне тестування.

3.6 Економічна оцінка проєкту

Для економічної оцінки проєкту з тестування програмного забезпечення «чорного ящика» потрібно врахувати:

1. Вартість ресурсів, що включає вартість людських ресурсів (заробітна плата тестувальників, менеджерів проєкту), вартість інструментів та обладнання, необхідного для тестування, а також витрати на навчання та розвиток команди;

2. Тривалість проєкту – довший проєкт може вимагати більше технічних ресурсів та часу людських ресурсів;

3. Ризики проєкту, такі як можливість затримок, збоїв в програмному забезпеченні, які можуть збільшити витрати;

4. Очікувані переваги –потенційне зменшення витрат завдяки виявленню та усуненню помилок на ранніх етапах розробки, а також підвищення якості кінцевого продукту;

5. Альтернативні стратегії тестування – порівняння витрат та ефективності чорного ящика з іншими методами тестування, наприклад, тестуванням «білого ящика» або автоматизованим тестуванням.

Виконаємо розрахунки для економічної оцінки проєкту тестування програмного забезпечення методом «чорного ящика», використовуючи прикладні дані (табл. 3.9).

Таблиця 3.9 – Приклад розрахунку витрат проєкту з тестування ПЗ

Ресурси	Види витрат	Кількість	Вартість за одиницю	Сума витрат	Примітка
Людські ресурси	Заробітна плата	3	30000 грн./міс.	540000 грн.	Тривалість 6 місяців
Обладнання та інструменти	Ліцензії на тестувальне ПЗ	1	50000 грн.	50000 грн.	-
Обладнання та інструменти	Обладнання (сервери, комп'ютери)	1	100000 грн.	100000 грн.	-
Навчання	Тренінги	3	20000 грн.	60000 грн.	-

Табл. 3.10 додатково враховує витрати, пов'язані з ризиками проєкту, очікувані переваги та альтернативні стратегії тестування, що надає більш повне уявлення про проєкт і його потенційний фінансовий результат.

Таблиця 3.10 – Повне описання витрат проєкту*

Ресурси	Види витрат	Кількість	Вартість за од.	Сума витрат	Примітка
Людські ресурси	Заробітна плата	3	30,000 грн./міс.	540,000 грн.	Тривалість: 6 місяців
Обладнання та інструменти	Ліцензії на тестувальне ПЗ	1	50,000 грн.	50,000 грн.	-
Обладнання та інструменти	Обладнання (сервери, комп'ютери)	1	100,000 грн.	100,000 грн.	-
Навчання	Тренінги	3	20,000 грн.	60,000 грн.	-
Ризики проєкту	Додаткові витрати на зарплату, Ризик непередбачених витрат	-	-	99,000 грн.	1 місяць затримки
Очікувані переваги	Зменшення витрат на усунення помилок, Покращення якості продукту	-	-	200,000 грн. + Нематеріальні переваги	Складно кількісно оцінити
Альтернативні стратегії	Тестування «білого ящика», Автоматизоване тестування	-	-	Варіативність витрат	Залежить від обраної стратегії

* Ці розрахунки є приблизними і засновані на припущеннях. Реальні витрати можуть відрізнятися в залежності від конкретних умов та вимог проєкту.

Розрахунок повернення інвестицій ROI (Return on Investment) для проєкту виконаємо за формулою:

$$ROI = (ЗД – ЗВ)/ЗВ * 100\%, \quad (3.1)$$

де: ЗД – загальний дохід; ЗВ – загальні витрати.

Припустимо, що загальний дохід від проєкту складає 1500000 грн. Загальні витрати проєкту (як ми раніше розрахували) = 825,000 грн. Тоді, розрахунок ROI

для проєкту за формулою (3.1) показує, що повернення інвестицій складає приблизно 81,82%. Це означає, що на кожну гривню, вкладену в проєкт, буде отримано 0,82 грн. чистого прибутку. Отриманий показник є досить високим, що свідчить про ефективність інвестицій у проєкт. Реальний ROI може варіюватися в залежності від багатьох зовнішніх та внутрішніх факторів.

ВИСНОВКИ

Основні результати роботи:

1. Проведено аналіз історичного розвитку та еволюції методів тестування «чорного ящика», виявлено ключові тенденції та зміни в підходах до тестування програмного забезпечення.

2. Проаналізовано та класифіковано сучасні методики тестування програмного забезпечення, оцінено їх особливості та ефективність на основі різноманітних метрик.

3. Представлено практичну реалізацію алгоритмів методів тестування програмного забезпечення «чорного ящика» з використанням різних мов програмування.

4. Проведено аналіз практичного впровадження методів тестування програмного забезпечення «чорного ящика» на прикладі конкретного експерименту, що включає методологію проведення експерименту, вибір програмного забезпечення та аналіз отриманих результатів.

Елементи наукової новизни:

1. Узагальнення системи класифікації методів тестування «чорного ящика», що дозволяє ефективніше порівнювати та вибирати методики для конкретних проєктів.

2. Систематизація та уніфікація метрик оцінки ефективності тестування, що дозволяє отримувати більш об’єктивну та всебічну картину результатів.

Практична значущість роботи:

1. Використання запропонованих підходів може значно підвищити якість та надійність програмного забезпечення, зменшити кількість помилок та невиявлених проблєм під час розробки програмного забезпечення.

2. Рекомендації щодо вибору методів та інструментів для тестування допоможуть ефективніше планувати та виконувати тестування.

Практичні рекомендації та перспективи подальших досліджень:

1. Розробити автоматизовану систему для вибору оптимальних методів тестування, залежно від специфіки проєкту.

2. Рекомендується подальше вивчення впливу автоматизації на процеси тестування, особливо з точки зору зниження витрат і підвищення швидкості виконання тестів.