Міністерство освіти і науки України
Національний університет водного господарства та природокористування
Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії
Кафедра комп'ютерних технологій та економічної кібернетики

04-05-93М

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт і самостійної роботи з дисципліни «Технології тестування програмних продуктів» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Інформаційні системи і технології» спеціальності 126 «Інформаційні системи та технології» денної та заочної форми навчання

Рекомендовано
науково-методичною радою
з якості ННІ КІТІ
Протокол № 1 від 20.10.2025 р.

Рівне – 2025

Методичні вказівки до виконання лабораторних робіт і самостійної роботи з дисципліни «Технології тестування програмних продуктів» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Інформаційні системи і технології» спеціальності 126 «Інформаційні системи та технології» денної та заочної форми навчання [Електронне видання]. Бабич Т. Ю. – Рівне : НУВГП. 2025. – 53 с.

Укладач: Бабич Т. Ю., к.е.н., доцент кафедри комп'ютерних технологій та економічної кібернетики.

Відповідальний за випуск:
Грицюк П. М., завідувач кафедри комп'ютерних технологій та економічної кібернетики.

Керівник (гарант) ОП: Гладка О. М., к.т.н., доцент кафедри комп'ютерних технологій та економічної кібернетики.

Голова науково-методичної ради з якості ННІ КІТІ :
Мартинюк П. М., д.т.н., професор.

ЗМІСТ

ВСТУП

Методичні вказівки до виконання лабораторних робіт і самостійної роботи з дисципліни «Технології тестування програмних продуктів» підготовлені для здобувачів вищої освіти ступеня «бакалавр», які навчаються за освітньо-професійною програмою «Інформаційні системи і технології» спеціальності 126 «Інформаційні системи та технології».

Мета вивчення дисципліни полягає в підготовці здобувачів вищої освіти до здійснення професійної діяльності щодо аналізу вимог та тестування програмного забезпечення, критеріїв вибору тестів, створення звітної тестової документації.

Основні **завдання**: набуття компетентностей з тестування програмного забезпечення, веб-сайтів, мобільних додатків; оформлення відповідної тестової документації; використання спеціалізованого програмного забезпечення для вирішення професійних завдань.

В даних методичних вказівках розроблено комплекс лабораторних робіт, виконання яких є невід'ємною складовою освітнього процесу.

Лабораторні роботи виконуються студентами самостійно під керівництвом викладача. Під час роботи здобувачі працюють індивідуально над тестуванням унікальних програмних продуктів. Це може бути власна розробка, програмний продукт іншого здобувача або ж готовий вебсайт чи мобільний додаток. Головною вимогою є *неповторюваність* тестованих продуктів. Контроль варіантів завдань здійснюється викладачем через спільний документ, де фіксуються назви тестованих продуктів.

За результатами виконання завдань лабораторної роботи здобувачами оформлюється звіт. Орієнтовний зміст звіту викладено у кожній темі лабораторної роботи. Для закріплення практичних навичок здобувачам запропоновано відповісти на контрольні запитання з кожної теми.

В якості підготовки до опитування на лекціях підготовлено термінологічні словники з ключових тем стосовно тестування. Для самостійної роботи запропоновано набір практичних задач по різних підходах до тестування.

ЛАБОРАТОРНІ РОБОТИ

Лабораторна робота 1
Тестування програмного забезпечення: розроблення тестів. Робота в системах управління тестуванням

Мета: набуття умінь проектування та оформлення тестових випадків.

Кількість балів: 9.

Теоретичні відомості

Тест-кейс (тестовий випадок, test case) — це набір умов і/або змінних, за допомогою яких тестувальник визначатиме наскільки повно тестований додаток задовільняє вимоги, що пред'являються до нього.

Базисом для підготовки тест-кейсів повинні бути вимоги (в тому або іншому вигляді), логічні умови, власний досвід, але у жодному випадку не готовий продукт.

Необхідними є наявність відомого вводу (вхідні дані) і очікуваного результату, який досягається після виконання тесту. Вхідні дані називаються передумовами тесту, а очікуваний результат - післяумовами тесту.

о Для того, щоб переконатися, що вимоги повністю задовільняються, може знадобитися декілька тест-кейсів. Для повного тестування всіх вимог, що пред'являються до додатку, повинен бути створений/виконаний щонайменше один тест-кейс для кожної вимоги.

о Якщо вимога має дочірні вимоги, то для кожної такої дочірньої вимоги повинен бути створений/виконаний також принаймні один тест-кейс. Деякі методології (наприклад, RUP) рекомендують створювати щонайменше два тест-кейси для кожної вимоги. Один з них повинен виконувати позитивне тестування, інший - негативне.

Крім того, що тест-кейси повинні перевіряти вимоги, а не повторювати їх, дуже важливим є наступний момент: тест-кейси повинні бути однозначними, тобто повинні бути складені і сформульовані так, щоб вони не допускали двозначного тлумачення, а однозначно розумілися всіма учасниками процесу розробки ПЗ.

Нижче наведено пропонований шаблон для оформлення тестових випадків.

ID № :: TestCase name (назва тестового випадку):

Summary (резюме, короткий опис):	
Steps (Кроки/дії)	Expected Results (Очікувані результати)
1. 2.	1. 2.

Тестовий випадок може мати наступні резолюції чи стани:

- No run (не запускався)
- Passed (пройшов)
- Failed (не пройшов, помилка)
- N/A (недоступний)

Приклад тестового випадку:

Steps (Кроки/дії):	Expected Results (Очікувані результати):
1. додати один запис “123” до списку 2. натиснути кнопку “Clear list”	1. запис доданий до списку 2. “Clear list” кнопка активна і список очищено

Важливою є своєчасність відміток (встановлення стану) про проходження/збою тест-кейсів. Для отримання об'єктивної інформації необхідно, щоб у кожен момент часу дані про виконання тест-кейсів були актуальними. Для цього необхідно, щоб зразу ж після прогону тест-кейсу тестувальник проставляв відповідну відмітку. Це дає змогу у будь-який момент часу відстежувати поточну ситуацію по процесу виконання тест-кейсів, зокрема:

- відсоток виконання від загальної запланованої кількості;
- кількість успішно пройдених тест-кейсів;
- кількість провалених тест-кейсів;
- інші похідні від них метрики.

Сучасне програмне забезпечення є складним, із великою кількістю функціональності, то і кількість розроблюваних тестових випадків є великою (сягає кількох сотень і більше). Тому для зберігання та відслідковування станів тест-кейсів використовуються системи їх обліку, такі як TestRail, TestLink, TestTrack, Mercury QC і т.п.

Інструкція з використання системи TestRail – за посиланням:
https://www.youtube.com/watch?v=sGUbyZtAtXg

У випадку не проходження тест-кейсу (стан: Failed) та переконання в тому, що це дійсно програмна помилка, необхідно оформити звіт про помилку (передбачено виконання під час лабораторної роботи 2).

Індивідуальне завдання

- Вибрати об'єкт для тестування, виділити в ньому модулі (можна взяти результати виконання курсової роботи з програмування або іншого створеного власноруч програмного продукту).

**У випадку відсутності розроблених програмних продуктів (або за бажанням) обрати пункт із завдання відповідно до порядкового номеру в списку групи.*

Розробити та оформити тестові випадки для MS Word:

1. пункт меню Подання -> Подання
2. пункт меню Подання -> Відображення
3. пункт меню Подання -> Масштаб
4. пункт меню Подання -> Вікно
5. пункт меню Макет -> Параметри сторінки
6. пункт меню Макет -> Абзац
7. пункт меню Макет -> Упорядкування
8. пункт меню Посилання -> Зміст
9. пункт меню Посилання -> Виноски
10. пункт меню Посилання -> Посилання та бібліографія
11. пункт меню Посилання -> Підписи
12. пункт меню Посилання -> Покажчик
13. пункт меню Посилання -> Таблиця посилань
14. пункт меню Вставлення -> Сторінки
15. пункт меню Вставлення -> Таблиці
16. пункт меню Вставлення -> Ілюстрації (Зображення)
17. пункт меню Вставлення -> Ілюстрації (Фігури)
18. пункт меню Вставлення -> Ілюстрації (Діаграма)
19. пункт меню Вставлення -> Посилання (Посилання)
20. пункт меню Вставлення -> Посилання (Закладка)
21. пункт меню Вставлення -> Посилання (Перехресне посилання)
22. пункт меню Вставлення -> Колонтитули (Верхній колонтитул)
23. пункт меню Вставлення -> Колонтитули (Нижній колонтитул)
24. пункт меню Вставлення -> Текст (Дата й час)

- Розробити тестові перевірки для обраного об'єкта із зазначенням модуля, що тестується (по 10 тест-кейсів).
- Опис оформити на вибір: в таблицях Excel або в системі TestRail (можна також обрати іншу систему розроблення тестів).

Важливо: *кожен здобувач виконує індивідуальне завдання, тему якого має бути зафіксовано у спільному документі (доступ надається викладачем).*

Контрольні запитання

1. Що таке тестовий випадок?
2. Яким вимогам повинен відповідати тестовий випадок?
3. Які стани може мати тестовий випадок?
4. Які є види тестових випадків?
5. Які атрибути повинен мати тестовий випадок?
6. Які Ви знаєте засоби обліку тестових випадків?

Лабораторна робота 2
Пошук і документування дефектів. Використання систем трекінгу багів

Мета: набуття умінь оформлення знайдених дефектів програмного продукту.

Кількість балів: 9.

Теоретичні відомості

Дефекти, виявлені тестувальником, мають бути коректно і зрозуміло описані, щоб розробник зміг відтворити цей дефект і усунути його. Опис кожного дефекту зберігається в спеціалізованій – багтрекінговій – системі (наприклад, JIRA, Bugzilla, Mantis, Redmine та ін.) або в заздалегідь створеному в програмному середовищі Microsoft Excel файлі (табл. 2.1).

Таблиця 2.1

Приклад опису дефекту *(для зручності відображено в транспонованому вигляді)*

№ з/п	1
Назва дефекту	Адміністраторська частина: Файли: Вибір файлу : Посилання "Скачати файл": Адміністратор не має можливості скачати завантажені студентом файли.
Важливість	Critical
Алгоритм відтворення	Кроки по відтворенню: 1. Входимо на web сайт... 2. Проходимо авторизацію: admin/admin.

	3. Вибираємо в меню категорію "Файли". 4. Вибираємо підкатегорію меню "Вибір файлу". 5. Вибираємо в полі "Огляд файлу" будь-який доступний файл. 6. Натискаємо на посилання "Скачати файл".
Фактичний результат	Перехід до сторінки "HTTP Status 404".
Очікуваний результат	Відбувається скачування вибраного файлу
Додаток	39.png
Примітка	REQ-26

Стовпці таблиці опису дефекту:

1. Headline – назва дефекту.
2. Severity – міра критичності (важливість дефекту).
3. Description – алгоритм відтворення.
4. Result – фактичний результат.
5. Expected result – очікуваний результат.
6. Attachment – прикріплені файли (додаток).

У багтрекінгових системах для кожного дефекту автоматично генерується його унікальний номер, у разі використання Microsoft Excel номер дефекту необхідно привласнювати вручну.

Вимогу специфікації, яка порушує виявлений дефект, можна додатково винести в примітку.

Додатково в описі дефекту може бути вказана Priority – міра терміновості виправлення дефекту розробником.

Розглянемо детально кожну категорію опису дефекту.

Headline (назва дефекту). Мета складання заголовка дефекту – надати коротку , але зрозумілу інформацію про те, де, що і внаслідок чого сталося. Характеристиками якісного заголовка є стисліcть, інформативність, точна ідентифікація проблеми.

Заголовок дефекту повинен відповідати на *три питання*:

1. *Де?* У якому місці інтерфейсу користувача знаходиться проблема. У цій частині заголовка слід також додатково вказати особливості тесту, якщо це допоможе розібратися в проблемі (версія операційної системи, браузеру, сторонніх застосувань, які мають відношення до тестованого програмного засобу).

2. *Що?* Що відбувається або не відбувається згідно із специфікацією або уявленням про нормальну роботу програмного

продукту. При цьому необхідно вказувати на наявність проблеми, а не на її зміст (його вказують в описі). Якщо зміст проблеми варіюється, усі відомі варіанти вказуються в описі.

3. *Коли (за яких умов)?* У який момент роботи програми або по настанню якої події проблема проявляється. Приклад: в додатку є діалог «Перетворити дані» з кнопкою «Перетворити». При натисненні цієї кнопки з'являється повідомлення про помилку – «Помилка 315».

Заголовок дефекту за описаною методикою складається так:

– Де?: Діалог «Перетворити дані».

– Що?: Показується повідомлення про помилку.

– Коли?: При натисненні кнопки «Перетворити».

Підсумковий заголовок матиме наступний вигляд:

Діалог «Перетворити дані» показує повідомлення про помилку при натисненні кнопки «Перетворити».

Прибираємо зайві слова, додаємо код помилки для зручності пошуку:

Діалог «Перетворити дані»: повідомлення про помилку 315 при натисненні кнопки «Перетворити».

Якщо сформулювати заголовок по формулі *Де? Що? Коли (за яких умов)?* важко, то можна скористатися наступним алгоритмом дій :

1. Пропустити заголовок дефекту.
2. Написати опис дефекту, фактичний і очікуваний результати.
3. Виділити ключові моменти, керуючись формулою *Де? Що? Коли (за яких умов)?*
4. Скласти ці ключові моменти разом.

Те, що вийде у результаті, і складатиме заголовок дефекту.

Severity (міра критичності). Міра критичності (серйозності, важливості) показує міру збитку, який наноситься проєкту існуванням дефекту.

У загальному випадку виділяють такі градації критичності дефектів (таблиця 2.2) :

– *Critical* (критичний);
– *Major* (значний);
– *Average* (середній значущості);
– *Minor* (незначний);
– *Enhancement* (пропозиція по поліпшенню).

Таблиця 2.2

Градації критичності дефектів

Рівні критичності дефектів	Описання	Приклади
Critical (критичний)	Функціональна помилка, що блокує роботу частини функціонала або усього застосування. Функціональна помилка, що порушує ключову (з точки зору кінцевого користувача або бізнесу замовника) функціональність додатку.	Заблокована вкладка (категорія меню). Неправильно підраховується підсумкова сума при введенні промокоду на знижку. Розкривається конфіденційна інформація.
Major (значний)	Функціональна помилка, що порушує нормальну роботу додатка, але не блокує роботу частини функціонала в цілому.	Неможливо завантажити відео- файли на персональній сторінці. Не працює система e-mail нотифікації. Не працює інтеграція з соціальними мережами. Необхідно перезапускати додаток при виконанні типових сценаріїв роботи
Average (середньої значимості)	Не дуже важлива функціональна помилка. Критичні дефекти GUI.	Не працює сортування. "Поїхав" текст за межі вікна.
Minor (незначний)	Незначні функціональні дефекти, що рідко зустрічаються. 90 % дефектів GUI.	Введені необов'язкові для заповнення поля, яких немає в специфікації. Граматичні, пунктуаційні помилки.
Enhancement (пропозиції по поліпшенню)	Функціональні пропозиції, раді з поліпшення дизайну (оформлення), навігації та ін.	Додати кнопку "Вгору" на довгих формах. Збільшити розмір шрифту.

	Такий дефект є необов'язковим для виправлення.	

Description (алгоритм відтворення). Мета складання алгоритму відтворення дефекту - послідовно описати кроки для повторення дефекту. Description має бути оформлений у вигляді списку перерахування дій :

Крок #1.

Крок #2.

…

n. Крок #n.

У разі, якщо для відтворення дефекту потрібно ряд початкових умов (наприклад, має бути створений певний набір документів, користувач або група користувачів з особливими правами), то ці передумови мають бути винесені в початок опису :

Передумови.

1. Крок #1.
2. Крок #2.

…

n. Крок #n.

При складанні Description необхідно наслідувати приведені нижче рекомендації.

Description – це чіткий алгоритм, в якому вітаються короткі, зрозумілі фрази і нумерація.

Не можна використати особисті пропозиції формату "Я думаю, що так буде кращий", "Я зайшов на сторінку". і так далі.

Можна використати спеціальні символи "+", "=", "<>" які допоможуть зробити подібність навігації : File > Open, DOC + XLS. Проте не рекомендується писати кроки в рядок через символи переходу : це ускладнює сприйняття дефекту.

Слід вказувати специфічні умови відтворення дефекту, якщо такі є. Наприклад, під яким користувачем ви працюєте (якщо це важливо).

Опис пропозицій щодо поліпшення має бути максимально повним і аргументованим.

Result (фактичний результат). Мета написання Result – чітко описати отриманий результат.

Expected result (очікуваний результат). Мета написання Expected result – навести аргументи розробникам, як саме має працювати додаток.

У Expected result має бути чітке обґрунтування, чому саме так повинен працювати додаток. Найкраще, якщо в ньому наведено посилання на конкретний пункт специфікації.

- Якщо в Expected result наводиться посилання на специфікацію, сам тестувальник додатково цитує текст специфікації, щоб скоротити час розробника на аналіз документа і однозначно вказати на спосіб вирішення проблеми.
- Якщо функція працює, але некоректно, то в Expected Result обов'язково повинно бути опис того, як вона повинна працювати коректно.
- Якщо зроблена помилка в напису або інтерфейсі проекту, необхідно грамотно і повністю вказати, як вона повинна бути виправлена – написати напис без помилки або описати необхідні зміни інтерфейсу.

Expected Result завжди слід заповнювати. Не варто покладатися на очевидність уявлень про правильну поведінку програми.

Текст Expected result рекомендується писати закінченими повними безособовими пропозиціями.

Attachment (додаток). Attachment – це прикріплений до дефекта файл, що доповнює опис: скріншот, файли, необхідні для відтворення дефекту, логи програми, відео помилки і т.д. Attachment є допоміжним засобом передачі інформації про проблему. Для всіх GUI дефектів attachment обов'язковий.

Якщо до дефекта прикріплюється файл, про це обов'язково повинно бути вказано в описі дефекту («See the file / screenshot / log / video attached»). Прикріплений файл не повинен бути надто великим за розміром (особливо це стосується відео: до 10 мегабайт), а також повинен ставитися саме до описаної помилку (наприклад, з балки додатки варто скопіювати в прикріплюється файл тільки дані про помилку). Формат файлу скріншота – PNG. Файл скриншота рекомендується робити числовим, нейтральним – 1.png, 25.png і т.п.

Скріншот повинен містити такі елементи: сама помилка, виділення місця помилки, стрілка до прямокутника, опис помилки: «Спостережуваний результат» і / або «Очікуваний результат». Текст

на скріншоті також необхідно виділити: обвести в прямокутник і набрати шрифтом, помітно відрізняється від шрифту програми. Якісно підготовлений скріншот повинен давати можливість зрозуміти сенс дефекту без необхідності читати його опис (рис. 2.1).

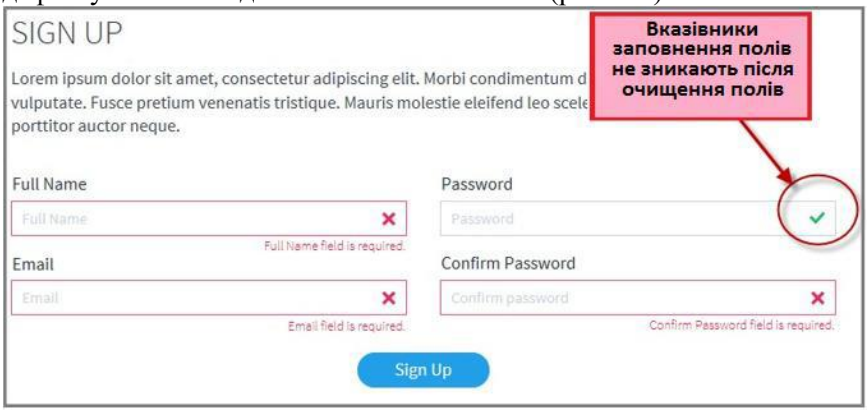

Рис. 2.1. Приклад скріншота, пояснюючий виявлений дефект

Робити знімок всього вікна програми необов'язково: на знімку повинна бути видна помилка і місце, в якому вона знаходиться. Якщо для розуміння дефекту необхідний контекст, то крім власне помилки фіксують необхідну інформацію (браузер, в якому відкрито web додаток, все вікно програми в фоні з назвою діалогового вікна, де ця помилка з'явилася). Якщо необхідно навести знімки декількох сторінок проекту, пов'язаних між собою, краще зробити це на одному скріншоті, поєднавши зображення по горизонталі і при необхідності зазначивши стрілками переходи значень, полів і т.п.

Рекомендації по опису дефектів

Часто повідомлення про помилку перетворюється в скорочений запис тільки основних дій, необхідних для відтворення помилки, опускаючи все несуттєві. Але, будучи незнайомим з внутрішньою структурою додатку, тестувальник не може знати, які з виконаних ним дій найбільш істотні для діагностування цієї помилки. Нехтуючи діями, які здаються незначним, підвищується ризик втрати важливої інформації. Кращий спосіб уникнути цієї проблеми полягає в тому, щоб просто перерахувати всі дії, які необхідні для відтворення

помилкової поведінки, починаючи з відкриття потрібної форми в проекті.

Якщо є підозра на повторення дефекту в декількох модулях проекту, цей факт потрібно досліджувати ще до внесення дефекту і при його описі вказати всі місця, де дефект відтворюється.

Дефект не повинен містити фразу: «Це не працює», дефект повинен показати, що і за яких умов не працює.

Щоб оформити дефект, його слід відтворити мінімум два рази, причому починаючи з самих нейтральних умов відтворення, і тільки після гарантованого повторення описати послідовність дій.

Не можна не описувати дефекти тільки тому, що їх не виходить відтворити. Факт неможливості з'ясувати причину дефекту в такому випадку обов'язково повинен бути зазначений в описі дефекту.

Дефекти доцільно групувати, однак робити це необхідно відповідно до наведених нижче **правил**:

- GUI дефекти можуть групуватися в один за ознакою форми, на якій вони знаходяться, тобто якщо одна форма містить кілька GUI дефектів з однаковим рівнем Severity (критичності), то їх можна об'єднати.
- Функціональні дефекти групуються в тому випадку, якщо мова йде про однотипні дефекти, які відтворюються в різних модулях, сторінках або полях (наприклад, динамічне оновлення не працює в модулях 1, 2 і 4 або відсутній валідації на спецсимволи на всіх полях сторінки).
- Групування функціональних дефектів за ознакою форми, на якій вони знайдені, не застосовується.
- Неприпустимо поєднувати в один дефекти різного типу, наприклад, функціональні і GUI. В такому випадку пишуться кілька дефектів на кожен тип.

Рекомендації по хорошому опису дефектів

1. Кроки відтворення, фактичний і очікуваний результати повинні бути докладно описані.
2. Дефект повинен бути зрозуміло описаний (з використанням загальнодоступної лексики, точних назв програмних засобів).
3. Необхідно давати посилання на відповідну вимогу, до порушення якого призводить фактичний результат роботи програмного засобу.

4. Якщо існує будь-яка інформація, яка допоможе швидше виявити або виправити дефект, необхідно повідомити цю інформацію.
5. Оточення (ОС, браузер, настройки тощо), під яким виникла помилка, має бути чітко вказано.
6. Створювати дефект і описувати його необхідно відразу ж, як тільки він був виявлений. Відкладання «на потім» призводить до ризику забути про дефект або й будь-яких деталях його відтворення. Несвоєчасне створення дефекту не дозволяє проектній команді реагувати на її виявлення в реальному часі.
7. Після опису дефекту необхідно ще раз перечитати його: переконатися, що всі необхідні поля заповнені, і все написано вірно.

Крім власне опису дефектів, результати тестування вносять до робочої тестової документації (Acceptance Sheet, Test Survey, Test Cases). Для цього навпроти виконаної перевірки вказують ступінь критичності виявленого дефекту, його номер і заголовок. Якщо за результатами конкретної перевірки виявлено кілька дефектів, то перелічують номери всіх дефектів, а в якості міри критичності і заголовка вказують найбільш серйозний дефект (рис. 2.2).

Function	Result	Issues	Comments
Форма реєстрації			
Поле "Ім'я"	Critical	15, 16	Українськомовний додаток не підтримує введення українських літер
Поле "Прізвище"	Critical	15, 16	Українськомовний додаток не підтримує введення українських літер
Поле "Ел.адрес або номер моб.телефона"	Major	21	Відсутня перевірка унікальності даних, що вводяться
Поле "Повторно введіть ваш ел.адрес або номер моб.телефона"	Average	24	Допускається залишити поле, обов'язкове для заповнення, порожнім
Поле "Новий пароль"	OK		

Рис. 2.2. Фрагмент Acceptance Sheet з внесеними після тестування дефектами

Індивідуальне завдання

1. Прослухати коротку інструкцію з користування Confluence + Jira: https://www.youtube.com/watch?v=FunG1zQhD0c (16 хвилин).
2. Основою для виконання пошуку дефектів може стати опис тест-кейсів із попередньої лабораторної роботи 1.
3. Для розроблених 10 тест-кейсів із зазначенням модуля, що тестується, знайти (або сформулювати) по два можливих дефекти на

кожен рівень Severity (Critical, Major, Average, Minor, Enhancement) для об'єкта тестування.

4. Описати по одному дефекту на кожен рівень Severity (Critical, Major, Average, Minor, Enhancement).
5. Описати всі знайдені дефекти в звіті про дефекти в середовищі Microsoft Excel.
6. У звіті про дефекти вказати номер тестованої збірки, назву програми, період часу тестування, ПІБ тестувальника, тестове оточення (операційна система, браузер).
7. Для кожного дефекту вказати його порядковий номер, заголовок, важливість, алгоритм відтворення, фактичний результат, очікуваний результат, додаток, примітку.
8. Для кожного дефекту обов'язково зробити скріншоти.
9. Результати тестування із зазначенням навпроти відповідної перевірки ступеня критичності виявленого дефекту, його номера та заголовка оформити на вибір або в таблицях Excel або через Confluence + Jira (згідно інструкції).
10. Оформити звіт та захистити лабораторну роботу.

Зміст звіту:

1. Мета роботи.
2. Назва вибраного об'єкта тестування.
3. Звіт про результати тестування вибраного об'єкта з перерахуванням тестових перевірок, сформульованих дефектів на кожен рівень Severity, опису дефектів.
4. Звіт про знайдені дефекти.
5. Робоча тестова документація з внесеними дефектами.
6. Висновки по роботі.

Важливо: *кожен здобувач виконує індивідуальне завдання, тему якого має бути зафіксовано у спільному документі (доступ надається викладачем).*

Контрольні запитання

1. Що таке дефект (баг) у програмному забезпеченні? Як він відрізняється від помилки?
2. Які ключові елементи повинні міститися у звіті про дефект (bug report)?
3. Які популярні системи трекінгу дефектів ви знаєте?

4. Яка інформація допомагає розробнику ефективно відтворити та виправити дефект?
5. Що означають терміни “severity” (серйозність) і “priority” (пріоритет) у контексті дефекту? Як вони відрізняються?
6. Які етапи проходить дефект у системі трекінгу: від виявлення до закриття?
7. Як можна перевірити, що виправлений дефект дійсно усунуто і не викликав нових помилок?

Лабораторна робота 3
Колективне інспектування програмного коду

Мета: Набуття навичок аналізу, виявлення та документування помилок в реальних проєктах.

Кількість балів: 9.

Теоретичні відомості

Завдання і цілі проведення формальних інспекцій

Не завжди можлива розробка автоматичних або чітко формалізованих ручних тестів для перевірки функціональності програмної системи. В деяких випадках виконання тестованого програмного коду неможливе в умовах, що створюються тестовим оточенням. Така ситуація можлива у вбудованих системах, якщо програмний код призначений для обробки виняткових ситуацій, що створюються тільки на реальному устаткуванні.

У тих випадках, коли верифікується не програмний код, а проектна документація на систему, яку не можна "виконати" або створити для неї окремі тестові приклади, також зазвичай вдаються до методу експертних досліджень програмного коду або документації на коректність або несуперечність.

Такі експертні дослідження зазвичай називають інспекціями або оглядами. Існує два типи інспекцій - *неформальні* і *формальні*.

При *неформальній інспекції* автор деякого документа або частини програмної системи передає його експертові, а той, ознайомившись з документом, передає авторові список зауважень, які той виправляє. Сам факт проведення інспекції і зауваження, як правило, окремо не

зберігаються, стан виправлень по зауваженнях також ніде не відстежується.

Формальна інспекція, навпаки, є чітко керованим процесом, структура якого зазвичай чітко визначається відповідним стандартом проекту. Таким чином, всі формальні інспекції мають однакову структуру і однакові вихідні документи, які потім використовуються при розробці.

Факт початку формальної інспекції чітко фіксується в загальній базі даних проекту. Також фіксуються документи, що піддаються інспекції, і списки зауважень, відстежуються внесені по зауваженнях зміни.

Бланк інспекції. Бланк інспекції - основний документ, який заповнюється в ході проведення інспекцій. Зазвичай він розробляється в ході розробки стандартів проекту. Для кожного типу об'єктів інспекції в проекті має бути розроблений свій бланк інспекції.

Бланк інспекції складається з трьох основних частин:

- титульний лист;
- список контрольних питань;
- список невідповідностей.

Крім того, рекомендується на всіх сторінках бланку, крім першої, поміщати колонтитул, що включає як мінімум номер бланка інспекції.

Формальні інспекції програмного коду. Процес формальної інспекції програмного коду підкорюється всім правилам, визначеним для абстрактної формальної інспекції, проте, має деякі особливості, зв'язані, насамперед, із структурою інспектованого програмного коду, а також з тим, що зазвичай інспектуються ділянки коду, які неможливо перевірити за допомогою автоматизованого тестування, базованого на тестових прикладах.

Особливості етапу огляду інспектованого коду. Виділення ключових точок і побудова або використання таблиць трасування. Перед початком огляду початкового коду рекомендується відзначити пункти вимог, на відповідність яким перевіряється початковий код, а також записати обґрунтування того, чому ці вимоги не можуть бути перевірені в автоматичному режимі. Після цього можна переходити до перегляду власне початкового коду. Всі позначки, які доведеться вносити в ході інспекції в початковий код, необхідно робити не у файлі, що зберігається в базі даних проекту, а в його копії, яка потім

буде підшита до матеріалів інспекції. Копія може існувати в тому ж форматі, що і початковий файл, або може бути роздрукована або виведена у формат DOC, PDF або аналогічний, який допускає коментування.

За допомогою таблиць трасувань в початковому коді визначаються інспектовані функції або методи, які відповідають необхідним вимогам. Ділянки коду виділяються і відзначаються міткою або номером відповідної вимоги. Якщо ділянка коду відповідає вимогам, то необхідно відзначити цей факт або виділенням кольором, або відповідною текстовою приміткою. Якщо ділянка коду має проблеми, цей факт має бути відображений або виділенням кольором, або посиланням на відповідний пункт списку зауважень в бланку інспекції.

У разі відсутності таблиць трасувань вимог на початковий код рекомендується робити позначки, що пояснюють, чому саме дана ділянка коду реалізує вказані вимоги. Такі позначки допоможуть на етапі обговорення документа.

Питання для інспектування програмного коду

1. Простота розуміння і відповідність стандартам кодування

1.1.Чи зрозумілі специфікації елементу?

1.2.Чи зрозумілий код елементу? Чи є проблеми в розумінні?

1.3.Чи достатньо код супроводжується коментарями? Чи допомагають коментарі зрозуміти процедури і функції елементу?

1.4.Чи можуть бути труднощі при модифікації елементу?

1.5.Чи встановлені в проекті стандарти кодування?

1.6.Чи відповідає код встановленим стандартам кодування?

1.7.Чи розроблені тести для перевірки дотримання стандартів?

2. Коректність звертання до даних

2.1.Чи використовуються непроініціалізовані змінні?

2.2.Чи не виходить значення кожного з індексів за межі, визначені для відповідного виміру при всіх звертаннях до масиву?

2.3.Чи приймає кожен індекс цілі значення при всіх звертаннях до масиву? Не цілі індекси не обов'язково є помилкою, але представляють практичну небезпеку.

2.4.Чи для всіх звертань за допомогою показників або змінних-посилань пам'ять, до якої проводиться звертання, розподілена (чи є "підвішені" звернення)?

2.5.Чи коректні атрибути при всіх псевдонімах? Якщо одна і та ж область пам'яті має декілька псевдонімів (імен) з різними атрибутами, то чи мають значення даних в цій області коректні атрибути при звертанні по одному з цих псевдонімів?

2.6.Чи відповідають атрибути запису і структури?

2.7.Чи відрізняються типи або атрибути змінних величин від тих, які передбачались компілятором? Це може відбутися у тому випадку, коли програма зчитує запис із пам'яті і звертається до них як до структур, але фізичне представлення записів відмінне від опису структури.

2.8.Чи обчислювані адреси бітових рядків? Чи передаються бітові рядки як аргументи?

2.9.Якщо до структури даних звертаються з декількох процедур або підпрограм, то чи визначена ця структура однаково в кожній процедурі?

2.10. Чи не перевищені межі рядка при індексації в ньому?

2.11.Чи існують які-небудь інші помилки в операціях з індексацією або при звертанні до масиву по індексу?

3. Коректність опису даних

3.1.Чи всі змінні описані явно?

3.2.Якщо початкові значення присвоюються змінним в операторах опису, то чи правильно ініціалізувалися ці значення?

3.3.Чи правильно для кожної змінної визначені довжина, тип і клас пам'яті?

3.4.Чи узгоджується ініціалізація змінної з її типом пам'яті?

3.5.Чи є змінні зі схожими іменами?

4. Коректність обчислень

4.1. Чи є обчислення, що використовують змінні неприпустимих типів?

4.2.Чи є обчислення, що використовують дані різного типу (змішані обчислення)? Чи можливе відкидання дробової частини?

4.3.Чи існують обчислення, що використовують змінні одного типу, але різної довжини?

4.4.Чи не менша довжина результату, ніж довжина обчислюваного значення?

4.5.Чи можливе переповнення або втрата проміжного результату під час обчислення?

4.6.Чи можливо, щоб дільник в операторі дорівнював нулю?

4.7.Чи можливий вихід значення змінної за межі її діапазону ?

4.8.Чи правильно використовується цілочисельна арифметика, особливо ділення.

5. Коректність порівняння даних

5.1. Чи порівнюються величини несумісних типів (наприклад, рядок символів з адресою)?

5.2.Чи порівнюються величини різних типів?

5.3.Чи коректні відношення порівняння?

5.4.Чи коректні булеві вирази?

5.5.Чи об'єднуються порівняння і булеві вирази?

5.6.Чи порівнюються дробові величини, представлені в двійковій формі?

5.7.Чи зрозумілий порядок проходження операторів?

6 . Коректність передачі управління

6.1.Чи може значення індексу в операторі вибору перевищити кількість переходів?

6.2.Чи буде завершений кожен цикл?

6.3.Чи буде завершена програма?

6.4.Чи можливо, що через вхідні умови цикл ніколи не зможе виконуватися?

6.5.Чи коректні можливі занулення в циклі?

6.6.Чи є помилки відхилення кількості ітерацій від норми?

6.7.Чи відповідають один одному оператори початку і кінця блоку ({ і } чи begin і end)?

7. Коректність інтерфейсу між модулями

7.1.Чи є невідповідність сигнатури параметрів у викликаючому модулі(підпрограмі) і тому, що викликається ?

7.2.Якщо модуль має декілька точок входу, чи передається параметр завжди не залежно від точки входу?

7.3.Чи не змінює підпрограма параметр, який повинен використовуватися тільки як вхідна величина?

7.4.Чи всі глобальні змінні використовуються правильно?

7.5.Чи передаються константи в якості аргументів?

8. Коректність вводу-виводу

8.1.Чи немає помилок в описах атрибутів файлів і операторах вводу-виводу?

8.2.Чи відповідає розмір буфера розміру запису?
8.3.Чи відкриті файли перед їх використанням?
8.4.Чи виявляються ознаки кінця файлу?
8.5.Чи виявляються помилки вводу-виводу?
8.6.Чи існують текстові помилки у вихідній інформації?

Індивідуальне завдання

- Ознайомитися з теоретичними відомостями щодо інспекції коду.
- Об'єднатися у групи по 3-4 студенти.
- Обрати проєкт для перевірки (кожна команда перевіряє чужий код).
- Зафіксувати об’єкт для інспекції коду **(без повторень)**, склад команд і розподіл ролей в спільній таблиці (доступ надається викладачем).
- Визначити ключові критерії оцінки (читабельність, стиль, безпека, тестування, тощо).
- Для проведення інспекції:
  - Ознайомитися з проєктом (структура, основна логіка).
  - Перевірити:
    - відповідність стилю;
    - наявність критичних помилок;
    - оптимізацію алгоритмів;
    - безпеку (захист від SQL-ін'єкцій, XSS тощо);
    - покриття тестами.

При проведенні перевірки спробуйте дати відповіді на питання для інспектування програмного коду.

- Документування помилок
- оформити звіти про знайдені проблеми.
- запропонувати можливі виправлення.
- автору коду проаналізувати фідбек і виправити помилки.

Шаблон звіту про інспекцію коду

1. Загальна інформація

- Назва проєкту: [Курсова робота студента]
- Автор коду: [Ім'я]
- Дата інспекції: [Дата]
- Рецензенти: [Імена, ролі]

2. Приклад опису виявлених проблем

№	Опис проблеми	Критичність	Рекомендоване виправлення
1	Відсутні тести на граничні значення	Висока ●	Додати юніт-тести
2	SQL-запити не захищені від ін'єкцій	Висока ●	Використати параметризовані запити
3	Невідповідність стилю коду (PEP8)	Середня ◍	Виправити форматування
4	Використання `print()` для логування	Низька ◍	Замінити на `logging`

3. Висновки

- Що було знайдено?
- Як покращили код?
- Що взяли для себе?

Контрольні запитання

1. Що таке колективне інспектування коду (code review)? Яка його мета?
2. Які ролі зазвичай беруть участь у процесі колективного інспектування коду?
3. Які типові етапи включає процес колективного перегляду коду?
4. Які переваги надає колективне інспектування у порівнянні з індивідуальним?
5. Які інструменти або платформи можна використовувати для організації code review?
6. Які типові помилки або проблеми найчастіше виявляються під час інспектування коду?
7. Як можна оцінити ефективність проведеного колективного інспектування?

Лабораторна робота 4
Тестування веб-додатку

Мета: набуття умінь виявлення помилок в інтерфейсі, функціональності, адаптивності та безпеці веб-застосунків.

Кількість балів: 9.

Теоретичні відомості

Тестування веб-додатків – це процес перевірки функціональності, надійності, безпеки, доступності та сумісності веб-ресурсу з метою виявлення помилок і забезпечення його якісної роботи для кінцевого користувача. Основною метою є забезпечення того, щоб веб-додаток:

- працював коректно на різних пристроях і браузерах;
- виконував усі заявлені функції;
- мав зручний та інтуїтивний інтерфейс;
- був захищеним від зовнішніх атак;
- швидко завантажувався та ефективно працював під навантаженням.

Перевіряються такі аспекти:

1. Функціональність: Чи правильно працюють форми, кнопки, посилання, логіка?
2. Інтерфейс користувача (UI): Чи виглядає все так, як повинно? Чи немає зсувів, поломок стилів?
3. Крос-браузерність: Як сайт виглядає та функціонує у Chrome, Firefox, Safari, Edge тощо.
4. Адаптивність: Чи сайт підлаштовується під екрани телефонів, планшетів, ноутбуків?
5. Безпека: Захист від SQL-ін'єкцій, XSS, CSRF та інших атак.
6. Продуктивність: Час завантаження, швидкість відповіді, поведінка під навантаженням.
7. Доступність: Чи зручно користуватися сайтом людям з обмеженими можливостями.

Автоматизоване тестування – це процес виконання тестів за допомогою спеціального ПЗ, що дозволяє:

- скоротити час перевірки змін у коді,
- забезпечити повторюваність тестів,
- покрити більшу кількість сценаріїв без ручної роботи.

Серед найпопулярніших інструментів варто виділити Selenium і Cypress. *Selenium* працює з автоматизацією браузера (Chrome, Firefox, Edge) і дає змогу виконувати тестування UI, перевірку форм, кліків, переходів. Особливістю *Cypress* є те, що він працює безпосередньо в браузері, має зручний інтерфейс.

Індивідуальне завдання

Важливо: *кожен здобувач виконує індивідуальне завдання, тему якого має бути зафіксовано у спільному документі (доступ надається викладачем).*

1. Аналіз вимог

Ознайомтесь із тестовим веб-додатком (використовуйте один із варіантів: власний проєкт, демонстраційний сайт, наприклад: https://demoqa.com , https://the-internet.herokuapp.com).

Визначте основні функції та інтерфейсні елементи, які потребують перевірки.

2. Функціональне тестування

Складіть чек-лист або тест-кейси для перевірки:

- Функціоналу форм (вхід, реєстрація, пошук).
- Навігації між сторінками.
- Відображення повідомлень про помилки.

3. UI та кросбраузерне тестування

Перевірте:

- Відображення інтерфейсу у різних браузерах (Chrome, Firefox, Edge).
- Роботу на різних розмірах екранів (використовуючи DevTools → Toggle Device Toolbar).
- Зафіксуйте виявлені недоліки (скріни, опис, браузер).

4. Адаптивність та доступність

- Оцініть, як виглядає сайт на мобільних пристроях.
- Перевірте, чи можна керувати елементами з клавіатури (Tab, Enter).

5. Безпека (базове)

- Виконайте спробу SQL-ін’єкції в полях форми (' OR 1=1 --).
- Спробуйте XSS-ін’єкцію в текстове поле (<script>alert('XSS')</script>).
- Зробіть висновок про наявність/відсутність вразливостей.

6. Автоматизоване тестування (базово, за бажанням)

Напишіть скрипт на Python з бібліотекою Selenium, який:

- Відкриває сайт.
- Знаходить форму входу/пошуку.

- Вводить дані та натискає кнопку.
- Перевіряє наявність результату або повідомлення.

Що здати:

- Тест-кейси (у вигляді таблиці або окремим файлом).
- Скріншоти з помилками/виявленими дефектами.
- Короткий звіт (до 1 сторінки): які тести проводились, що виявлено, які рекомендації.
- (Опціонально) — скрипт автоматизації + відео/гіфка роботи.

Орієнтовний шаблон звіту з тестування веб-додатку

Звіт

Тема: Тестування веб-додатку

Студент: _____

Група: _____

Дата: _____

Тестований веб-додаток: [URL або назва сайту]

1. Опис веб-додатку

Коротко опишіть призначення та функціонал веб-додатку (1-3 речення):

Приклад: Веб-додаток дозволяє користувачам створювати акаунт, авторизуватись і публікувати повідомлення у блозі.

2. Проведені види тестування

Позначте типи тестування, які були виконані:

- Функціональне тестування
- UI-тестування
- Кросбраузерне тестування
- Тестування адаптивності
- Перевірка доступності
- Тестування безпеки (SQL/XSS)
- Автоматизоване тестування

3. Таблиця тест-кейсів

№ Назва тесту Кроки тестування Очікуваний результат Фактичний результат Статус (OK/Bug)

1 Тест логіну з вірними даними Ввести email+пароль, натиснути Login Користувача авторизовано Успішно OK

2 SQL-ін’єкція у полі логіну Ввести ' OR 1=1 -- Відмова в доступі або помилка Авторизовано BUG

…. (Додайте потрібну кількість рядків).

4. Виявлені дефекти

Вкажіть 2–5 основних виявлених помилок:

Приклад:

- У Firefox форма реєстрації обрізається на малих екранах.
- Сайт дозволяє XSS-ін’єкцію у полі «Коментар».

5. Докази (скріншоти)

Вставте скріншоти або додайте посилання на них (Google Drive, GitHub тощо):

6. Результати автоматизації (за наявності)

Якщо ви писали скрипт (Selenium тощо), опишіть його дію:

- Скрипт відкриває головну сторінку, вводить дані у форму, перевіряє, чи з’явилось повідомлення «Welcome».
- Посилання на код / відео / GitHub (опціонально):

7. Висновки

- Що працює добре?
- Які проблеми найпоширеніші?
- Що можна покращити?

Контрольні запитання

1. Які основні типи тестування застосовуються до веб-додатків?
2. Чим відрізняється тестування адаптивності від кросбраузерного тестування?
3. Які інструменти можна використовувати для автоматизованого тестування веб-додатків?
4. Які загрози безпеки характерні для веб-додатків? Як їх можна виявити під час тестування?
5. Що потрібно перевірити під час тестування форм на веб-сторінках?
6. Опишіть послідовність дій для перевірки працездатності посилань на веб-сайті.

Лабораторна робота 5
Тестування мобільного додатку

Мета: набуття навичок створення і оформлення тест-кейсів для мобільного застосунку.

Кількість балів: 9.

Теоретичні відомості

Основні моменти, на які необхідно звертати особливу увагу саме при тестуванні додатків на мобільних пристроях.

Розмір екрана та touch-інтерфейс:

- розміри всіх елементів графічного інтерфейсу користувача;
- перевірка можливості використання усіх активних елементів (кнопок, посилань та ін.);
- швидкість відгуку активних елементів повинна бути досить високою;
- перевірка того, що багаторазове швидке натискання на кнопку не викличе екстрене завершення програми;
- підтримка мультитач – одночасне натискання кількох кнопок;
- підтримка горизонтального (landscape) та вертикального (portrait) положення;
- перевірка використання в додатку спеціальних жестів (double tap, swipe, pinch in/out та ін.).

Ресурси телефону:

- необхідно проконтролювати можливі витоки пам'яті. Часто це трапляється в додатках з вікнами, що містять велику кількість інформації. Також витоки пам'яті можуть бути присутніми під час тривалої роботи програми;
- перевірка обробки ситуацій нестачі пам'яті для функціонування операційної системи, під час роботи програми в активному та фоновому режимах;
- недолік місця для установки або роботи програми;
- установка на SD-карту;
- перевірка роботи батареї (акумулятора) пристрою при запущеному додатку, роботи у фоновому режимі, при включеному Wi-Fi, 3G Інтернеті, без підключення до мережі та ін.

Різні роздільні здатності екрану та версії ОС:

- необхідно перевірити роботу програми на пристроях з різними роздільними здатностями екрану. На екранах з високою роздільною здатністю (наприклад, ретіна-екран) елементи інтерфейсу та текст відображаються дрібніше, при роботі додатка на пристрої з екраном більш низькою роздільною здатністю – елементи інтерфейсу можуть стати занадто великими;
- необхідно переконатися, що додаток не може бути встановлений на непідтримувані пристрої. При цьому обов'язково тестування програми на усіх заявлених підтримуваних пристроях.

Реакція програми на зовнішні переривання:

- вхідні та вихідні SMS та MMS;
- вхідні та вихідні дзвінки;
- нагадування, будильник та ін.;
- відключення та підключення Wi-Fi. Наприклад, додаток може екстрено завершувати свою роботу під час запуску авіарежиму при відключеному Wi-Fi. При втраті сигналу та одночасному виконанні операції може відображатися нескінченне завантаження програми, замість коректного повідомлення про відсутність інтернет-підключення;
- часто виникають проблеми з переходом в онлайн-режим після офлайн режиму;
- відключення та підключення SD-карти;
- відключення/підключення мобільного пристрою до зарядного пристрою;
- робота з фізичною клавіатурою (якщо в списку підтримуваних моделей є такі);
- робота в фоновому (background) режимі. Для відправки у фоновий режим – запускається додаток, а потім натискається кнопка home. В результаті при повторному запуску програми бувають помилки, екстрені завершення роботи, неправильне відображення останнього відкритого вікна. Також при тривалій бездіяльності додатка виникає екстрене завершення роботи або помилки;
- робота додатка після виходу зі сплячого режиму;
- сумісність з іншими додатками.

Індивідуальне завдання

1. Виберіть мобільний додаток для тестування. Зафіксуйте свій вибір у спільному документі (**без повторень**, доступ надається викладачем).
2. Встановіть тестовий мобільний додаток на пристрій або запустіть на емуляторі.
3. Визначте основні функції, які потрібно протестувати.
4. Складіть 5-7 тест-кейсів для перевірки функціональності додатку.
5. Виконайте тестування згідно з тест-кейсами.
6. Зафіксуйте результати тестування та виявлені дефекти.

Рекомендація послідовності виконання

1. Розробити тест-план (короткий опис того, що буде перевірятися).
2. Створити таблицю з тест-кейсами (опис, вхідні дані, очікуваний результат, фактичний результат, статус).
3. Виконати тестування мобільного додатку.
4. Зафіксувати щонайменше 2 знайдених дефекти (оформити їх опис: назва, кроки відтворення, очікуваний результат, фактичний результат).

У звіті відобразити:

- Назву мобільного додатку
- Скріншоти тест-плану і тест-кейсів
- Скріншоти процесу тестування (за бажанням)
- Опис виявлених дефектів
- Висновки: які труднощі виникли при тестуванні, що вдалося виявити

Приклад таблиці для тест-кейсів

№	1
Назва тест-кейсу	Перевірка додавання нового завдання
Передумови	Додаток відкритий
Кроки	1. Натиснути кнопку "Додати". 2. Ввести текст завдання. 3. Натиснути "Зберегти"
Очікуваний результат	Нове завдання відображається у списку
Фактичний результат	Завдання відображається
Статус (Pass/Fail)	Pass

№	2
Назва тест-кейсу	Перевірка видалення завдання
Передумови	У списку є завдання
Кроки	1. Провести пальцем по завданню вліво 2. Натиснути "Видалити"
Очікуваний результат	Завдання зникає зі списку
Фактичний результат	Завдання не видалилось
Статус (Pass/Fail)	Fail

№	3
Назва тест-кейсу	Перевірка роботи додатку без інтернету
Передумови	Відключено Wi-Fi і мобільні дані
Кроки	1. Запустити додаток 2. Спробувати додати нове завдання
Очікуваний результат	Додаток працює офлайн, завдання додається
Фактичний результат	Завдання додається
Статус (Pass/Fail)	Pass

Шаблон для опису дефекту:

Дефект № __

Назва дефекту: Видалення завдання не працює

Опис: При спробі видалити завдання зі списку шляхом свайпу вліво, завдання не видаляється.

Передумови: У списку є хоча б одне завдання.

Кроки для відтворення:

1. Відкрити мобільний додаток.
2. Додати нове завдання (за потреби).
3. Провести пальцем по завданню вліво.
4. Натиснути кнопку "Видалити".

Очікуваний результат: Завдання зникає зі списку.

Фактичний результат: Завдання залишається у списку.

Скріншоти (за наявності): *(додати зображення)*

Статус: Open (відкритий)

Контрольні запитання

1. Які основні відмінності між тестуванням веб-додатків і мобільних додатків?
2. Які види тестування є критично важливими для мобільних додатків?
3. Що таке тестування на різних пристроях і чому воно важливе?
4. Які особливості потрібно враховувати під час тестування мобільного додатку в офлайн-режимі?
5. Які є популярні інструменти для тестування мобільних додатків.
6. Що таке gesture testing (тестування жестів), і як його проводять?
7. Які проблеми можуть виникати при тестуванні мобільного додатку на різних операційних системах (iOS та Android)?

Лабораторна робота 6
Використання штучного інтелекту у тестуванні програмного забезпечення

Мета: набуття навичок застосування інструментів штучного інтелекту для автоматизованого тестування власного ПЗ, зокрема для генерації тестів, пошуку помилок, виявлення вразливостей і покращення якості коду.

Кількість балів: 8.

Теоретичні відомості

Зі зростанням складності програмних систем традиційні методи тестування часто виявляються недостатньо ефективними, трудомісткими й затратними. Саме тому дедалі більше компаній звертають увагу на використання штучного інтелекту (ШІ) для автоматизації та оптимізації процесів тестування.

Штучний інтелект здатен аналізувати великі обсяги даних, виявляти закономірності, прогнозувати помилки та самостійно створювати тестові сценарії. Наприклад, за допомогою машинного навчання можна побудувати моделі, що виявляють потенційно

вразливі місця в коді або функціональність, яка найчастіше викликає збої. Це дозволяє зосередити увагу тестувальників на найважливіших ділянках системи.

Інструменти, що базуються на ШІ, також використовуються для генерації тестових даних, виявлення дублікатів дефектів, пріоритезації тест-кейсів та прогнозування ризиків релізу. Наприклад, такі системи можуть в автоматичному режимі оновлювати тестові сценарії після змін у програмному коді, що значно скорочує час на регресійне тестування. Популярні платформи, як-от Testim, Applitools, Functionize або mabl надають змогу виявляти UI-зміни, прогнозувати поведінку користувачів, автоматично фіксувати помилки та адаптуватися до змін у програмі без ручного втручання.

Проте використання ШІ у тестуванні не позбавлене викликів. Воно потребує високоякісних даних, відповідної інфраструктури та розуміння принципів роботи алгоритмів. Крім того, повна автоматизація не може повністю замінити людську інтуїцію й критичне мислення, особливо в контексті UX-тестування або тестування нестандартних сценаріїв.

Ефективне використання технології ШІ можливе лише за умов грамотної інтеграції в загальний процес розробки та тестування, а також при активній участі людини як контролера і аналітика.

Індивідуальне завдання

1. Оберіть фрагмент або модуль вашої курсової роботи (або іншого програмного продукту), який ви хочете протестувати (не менше ніж одна функція/клас/частина UI/логіки). Зафіксуйте свій вибір у спільному документі (**без повторень**, доступ надається викладачем). Підготуйте код до аналізу: додайте коментарі, структуруйте його (якщо потрібно).
2. Використання ШІ для тестування.

Крок 1: Генерація тест-кейсів.

- За допомогою ChatGPT, GitHub Copilot або подібного ШІ-інструменту згенеруйте юніт-тести для обраного фрагменту.
- За потреби доопрацюйте згенеровані тести вручну.

Крок 2: Виявлення помилок та покращення коду.

- Проаналізуйте ваш код за допомогою інструменту зі штучним інтелектом (наприклад, Snyk Code, Codacy AI, SonarLint, Codeium або ChatGPT).
- Зверніть увагу на:
  – логічні помилки,
  – потенційні вразливості,
  – повторення коду,
  – порушення стилю або принципів SOLID/DRY.

Крок 3 (додатково, для бажаючих): Візуальне тестування.

Якщо ваше ПЗ має інтерфейс, протестуйте UI за допомогою Applitools або аналогічного інструменту.

Оформлення звіту:

- Назва проєкту/курсової роботи.
- Фрагмент коду, який тестувався.
- Скріни сформульованих запитів до ШІ.
- Згенеровані тести (з коментарями).
- Результати запуску тестів.
- Помилки, виявлені ШІ, та зміни, які ви внесли.
- Висновки щодо ефективності використання ШІ у вашому випадку. Наскільки згенеровані ШІ тести були коректними та повними? Які саме помилки ШІ допоміг вам виявити? Чи виявили ви обмеження у використанні ШІ-інструментів?

Контрольні запитання

1. Які основні переваги використання штучного інтелекту в тестуванні ПЗ порівняно з традиційними методами?
2. У чому полягає застосування машинного навчання в автоматизованому тестуванні? Наведіть приклади.
3. Які типи тестування можуть бути оптимізовані за допомогою ШІ?
4. Які дані зазвичай використовуються для навчання моделей ШІ в контексті тестування ПЗ?
5. Назвіть популярні інструменти або фреймворки, що поєднують автоматизацію тестування з AI/ML.
6. Які обмеження та виклики пов’язані із впровадженням ШІ у процес тестування ПЗ?

ТЕРМІНОЛОГІЧНІ СЛОВНИКИ

Словник основних термінів з тестування

Українська/ Англійська	Опис
Тестування Testing	Процес перевірки програмного забезпечення на відповідність вимогам.
Дефект (баг) Defect (Bug)	Помилка в програмному коді, яка призводить до некоректної роботи системи.
Помилка Error	Дія або невірне рішення, яке призводить до дефекту в коді.
Збій Failure	Невідповідність очікуваної та фактичної поведінки системи під час виконання.
Тест-кейс Test Case	Набір умов і кроків для перевірки конкретної функції.
Тест-план Test Plan	Документ, що описує стратегію, обсяг і підхід до тестування.
Тестовий сценарій Test Script	Набір інструкцій для виконання тесту (може бути автоматизованим або ручним).
Автоматизоване тестування Automated Testing	Виконання тестів за допомогою спеціальних програмних засобів.
Ручне тестування Manual Testing	Виконання тестів без використання автоматизації.
Функціональне тестування Functional Testing	Перевірка відповідності функцій ПЗ вимогам.
Нефункціональне тестування Non-functional Testing	Перевірка параметрів, що не стосуються функцій (швидкість, безпека тощо).
Тестування "чорної скриньки" Black Box Testing	Метод тестування без аналізу внутрішньої структури коду.
Тестування "білої скриньки" White Box Testing	Метод тестування, що враховує внутрішню структуру коду.

Регресійне тестування Regression Testing	Повторне тестування для перевірки, чи не з’явилися нові помилки після змін у коді.
Інтеграційне тестування Integration Testing	Перевірка взаємодії між різними модулями системи.
Модульне тестування Unit Testing	Тестування окремих модулів або функцій програми.
Системне тестування System Testing	Перевірка всієї системи як єдиного цілого.
Навантажувальне тестування Load Testing	Перевірка роботи системи під навантаженням.
Стресове тестування Stress Testing	Перевірка системи на витривалість у критичних умовах.
Юзабіліті тестування Usability Testing	Оцінка зручності використання системи для кінцевого користувача.
Альфа-тестування Alpha Testing	Початкове тестування всередині компанії перед випуском продукту.
Бета-тестування Beta Testing	Тестування кінцевими користувачами перед офіційним релізом.
Приймальне тестування Acceptance Testing	Остаточна перевірка перед впровадженням продукту.
Тестове середовище Test Environment	Налаштоване оточення для виконання тестування.
Звіт про дефект Bug Report	Документ, що містить опис знайденої помилки.
Покриття коду Code Coverage	Відсоток коду, що перевірений тестами.
Димове тестування Smoke Testing	Початкове тестування, яке перевіряє, чи працюють основні функції ПЗ.

Саніті-тестування Sanity Testing	Швидка перевірка після незначних змін у коді.
Дослідницьке тестування Exploratory Testing	Неформальне тестування, під час якого тестувальник досліджує систему.
Ad-hoc тестування Ad-hoc Testing	Спонтанне тестування без підготовки тест-кейсів.

Словник термінів за моделями розробки ПЗ

Українською Англійською	**Опис**
Водоспадна модель Waterfall Model	Послідовна модель, де кожен етап починається після завершення попереднього.
Ітеративна модель Iterative Model	ПЗ розробляється через послідовні ітерації з поступовим удосконаленням.
Agile модель Agile Model	Гнучка модель, що орієнтується на швидке постачання продукту та адаптацію.
Канбан Kanban	Метод візуального керування завданнями без чітких спринтів.
Спіральна модель Spiral Model	Комбінація ітераційної моделі з оцінкою ризиків.
V-модель V-Model	Розширення водоспадної моделі з чітким зв’язком між розробкою та тестуванням.
RAD (швидка розробка додатків) Rapid Application Development (RAD)	Модель з акцентом на швидке створення прототипів і мінімізацію планування.
Побудова прототипів Prototyping	Створення функціонального шаблону для демонстрації ідеї.
Компонентна розробка Component-based Development	Розробка з використанням готових модулів.

Словник термінів з тестів та тест-кейсів

Українська Англійська	Опис
Тест Test	Одиниця перевірки програмного забезпечення, що має вхідні дані, очікуваний результат і критерії виконання.
Тест-кейс Test Case	Документований набір умов, кроків і очікуваних результатів для перевірки конкретної функції або сценарію.
Тестовий сценарій Test Script	Автоматизований або ручний набір інструкцій для виконання тестів.
Тестовий набір Test Suite	Група тест-кейсів, що об’єднані за певними критеріями.
Тестовий план Test Plan	Документ, що описує мету, стратегію, обсяг і ресурси для тестування.
Тестовий звіт Test Report	Документ, що містить результати тестування та висновки.
Очікуваний результат Expected Result	Правильний (очікуваний) вихідний результат виконання тесту.
Фактичний результат Actual Result	Реальний вихідний результат тесту після виконання.
Тестовий крок Test Step	Окремий крок у тест-кейсі, що визначає дію та очікуваний результат.
Попередні умови Preconditions	Початкові умови, що мають бути виконані перед запуском тесту.
Постумови Postconditions	Стан системи після виконання тесту.
Критерії проходження тесту Pass Criteria	Умови, за яких тест вважається успішним.
Критерії провалу тесту Fail Criteria	Умови, за яких тест вважається проваленим.
Регресійний тест-кейс Regression Test Case	Тест-кейс, що перевіряє, чи не з’явилися нові помилки після змін у коді.
Саніті тест-кейс Sanity Test Case	Тест-кейс, що перевіряє коректність основного функціоналу після змін.

Смоук тест-кейс Smoke Test Case	Початковий тест, що перевіряє, чи працює критичний функціонал системи.
Позитивне тестування Positive Testing	Перевірка системи за правильними вхідними даними.
Негативне тестування Negative Testing	Перевірка системи на некоректні або непередбачені вхідні дані.
Граничне тестування Boundary Testing	Перевірка роботи системи на граничних значеннях вхідних даних.
Динамічне тестування Dynamic Testing	Виконання тестів на працюючому програмному забезпеченні.
Статичне тестування Static Testing	Аналіз тест-кейсів, вимог та коду без виконання програми.
Тестування на основі вимог Requirement-Based Testing	Створення тест-кейсів на основі вимог до ПЗ.
Тестування на основі ризиків Risk-Based Testing	Створення тестів для критичних частин системи, що можуть містити дефекти.
Виконання тесту Test Execution	Процес запуску тесту та аналізу його результатів.
Статус тесту Test Status	Поточний стан тесту (наприклад, "Виконано", "Не виконано", "Помилка").
Трасування тест-кейсів Test Traceability	Зв’язок між тест-кейсами та вимогами, що перевіряються.
Пріоритизація тест-кейсів Test Case Prioritization	Визначення важливості тест-кейсів для ефективного тестування.
Дублікат тест-кейсу Duplicate Test Case	Тест-кейс, що дублює існуючий тест без додаткової користі.
Тест-кейс з високим пріоритетом High-Priority Test Case	Найважливіший тест, що перевіряє критичний функціонал.

Тест-кейс з низьким пріоритетом Low-Priority Test Case	Тест, що перевіряє менш критичні сценарії.

Словник термінів з багів

Українська Англійська	Опис
Баг (дефект, помилка) Bug (Defect, Issue)	Некоректна поведінка програмного забезпечення через помилку в коді або логіці.
Баг-репорт Bug Report	Опис знайденої помилки, що містить деталі для її відтворення та виправлення.
Дефект Defect	Відхилення фактичного результату від очікуваного через проблеми в коді або логіці.
Вразливість Vulnerability	Недолік у безпеці програмного забезпечення, який може бути використаний зловмисниками.
Відтворюваний баг Reproducible Bug	Баг, який можна стабільно повторити за певних умов.
Невідтворюваний баг Non-reproducible Bug	Баг, який з’являється випадково або не має стабільних умов відтворення.
Критичний баг Critical Bug	Помилка, що робить систему непридатною для використання.
Блокуючий баг Blocking Bug	Баг, що унеможливлює подальше тестування або роботу з системою.
Фатальний баг Fatal Bug	Помилка, що призводить до аварійного завершення роботи програми.
Мінорний баг Minor Bug	Незначний дефект, що не впливає на основний функціонал.
Косметичний баг Cosmetic Bug	Дефект, пов’язаний із візуальним оформленням або UI (шрифти, кольори, вирівнювання тощо).
Логічна помилка Logical Error	Баг, спричинений помилковою логікою або алгоритмом.

Функціональний баг Functional Bug	Дефект, що впливає на функціональність системи.
Регресійний баг Regression Bug	Помилка, яка з'явилася після внесення змін у код або оновлення ПЗ.
Прихований баг Latent Bug	Помилка, яка залишається непомітною в нормальних умовах, але може проявитися за певних сценаріїв.
Баг UI UI Bug	Помилка, що стосується інтерфейсу користувача (наприклад, некоректне розташування кнопок).
Баг UX UX Bug	Дефект, що погіршує зручність використання (наприклад, незрозумілий навігаційний потік).
Баг продуктивності Performance Bug	Проблема, що впливає на швидкість або ефективність роботи системи.
Випадковий баг Intermittent Bug	Дефект, який виникає рідко або непередбачувано.
Відтворюваність Reproducibility	Можливість повторного виклику помилки за тих самих умов.
Регресія Regression	Ситуація, коли виправлення одного багу призводить до появи нового або повернення старого.

Словник термінів з тестування вебпроєктів

Українською Англійською	**Опис**
Функціональне тестування Functional Testing	Перевірка відповідності функціоналу вимогам.
Юзабіліті Usability	Оцінка зручності інтерфейсу для користувача.
Тестування сумісності Compatibility Testing	Перевірка роботи вебсайту на різних пристроях, браузерах, ОС.
Кросбраузерне тестування Cross-browser Testing	Перевірка коректного відображення у різних браузерах.
Адаптивність Responsive Design Testing	Перевірка адаптації сайту до екранів різного розміру.

Тестування безпеки Security Testing	Виявлення вразливостей (XSS, SQL-ін’єкції тощо).
Продуктивність Performance Testing	Оцінка швидкодії, навантаження та стабільності.
Тестування навантаження Load Testing	Імітація великої кількості користувачів для перевірки стійкості.
Тестування локалізації Localization Testing	Перевірка правильності перекладу та формату локалізованого контенту.
Тестування посилань Link Testing	Перевірка робочих (небитих) внутрішніх і зовнішніх посилань.
Валідація HTML/CSS HTML/CSS Validation	Перевірка коду на відповідність стандартам.
Автоматизоване тестування Automated Testing	Виконання тестів за допомогою скриптів (наприклад, Selenium).
Ручне тестування Manual Testing	Тестування без використання автоматизації.
Тест-кейси Test Cases	Набір дій та очікуваних результатів для перевірки функціоналу.
Бекенд Backend	Серверна частина веб-додатку.
Фронтенд Frontend	Інтерфейсна частина веб-додатку, яку бачить користувач.
API-тестування API Testing	Перевірка взаємодії між клієнтом і сервером через API.
CI/CD CI/CD	Інтеграція та доставка змін у коді з автоматизованим тестуванням.

Словник термінів з тестування мобільних додатків

Українською Англійською	**Опис**
Функціональне тестування Functional Testing	Перевірка, чи мобільний додаток працює згідно з вимогами.
UI-тестування UI Testing	Перевірка інтерфейсу користувача (відображення, елементи, стилі).

UX-тестування UX Testing	Оцінка зручності користування додатком.
Тестування сумісності Compatibility Testing	Перевірка на різних пристроях, ОС та розмірах екранів.
Кросплатформне тестування Cross-platform Testing	Тестування додатку на Android та iOS.
Адаптивність Responsiveness Testing	Перевірка адаптації UI до різних екранів і орієнтацій.
Тестування продуктивності Performance Testing	Оцінка швидкодії, споживання ресурсів (CPU, RAM, батарея).
Тестування навантаження Load Testing	Перевірка стійкості при великій кількості користувачів.
Тестування переривань Interruption Testing	Реакція додатку на дзвінки, SMS, повідомлення, зміни підключень.
Тестування офлайн/онлайн режимів Offline/Online Testing	Перевірка поведінки додатку без мережі та при відновленні з'єднання.
Геолокаційне тестування Geolocation Testing	Перевірка функцій, що залежать від місця розташування.
Тестування жестів Gesture Testing	Тестування свайпів, натискань, масштабування тощо.
Тестування оновлень Update Testing	Перевірка правильності оновлення додатку без втрати даних.
Інсталяційне тестування Installation Testing	Перевірка встановлення, запуску, деінсталяції додатку.
Безпекове тестування Security Testing	Виявлення вразливостей мобільного додатку.
Тестування споживання енергії Battery Usage Testing	Аналіз впливу додатку на заряд акумулятора.
Тестування push-повідомлень Push Notification Testing	Перевірка надсилання, доставки та дій після натиснення.
Емулятор Emulator	Програмний інструмент для симуляції пристрою.
Фізичний пристрій Real Device	Тестування на реальному смартфоні чи планшеті.

Автоматизоване тестування Automated Testing	Виконання тестів за допомогою фреймворків (Appium, Espresso).

ПРАКТИЧНІ ЗАВДАННЯ ДЛЯ САМОСТІЙНОГО ВИКОНАННЯ

Тестування методом "чорної скриньки"

Задача 1. Перевірка форми логіну.

Ситуація: Веб-додаток має форму входу з полями *логін* і *пароль*. Пароль приховується (••••), є кнопка "Увійти".

Завдання: скласти набір тест-кейсів для перевірки правильної та неправильної поведінки системи:

- правильні дані
- неправильний пароль
- порожні поля
- логін без @
- спеціальні символи

Очікування: результати у вигляді таблиці тест-кейсів (вхідні дані → очікуваний результат).

Задача 2. Перевірка калькулятора.

Ситуація: є проста веб-форма для додавання двох чисел. Користувач вводить числа у два поля й натискає "Обчислити".

Завдання: протестувати:

- два позитивних числа
- від'ємні значення
- одне з полів порожнє
- введено текст замість числа
- десяткові числа

Очікування: результати з прикладами вхідних даних і очікуваних відповідей.

Задача 3. Перевірка форми реєстрації.

Ситуація: реєстраційна форма з обов'язковими полями: Ім'я, Email, Пароль (мінімум 8 символів), Підтвердження паролю.

Завдання: Виявити граничні умови й описати, які комбінації введення слід протестувати методом чорної скриньки.

Задача 4. Перевірка фільтра товарів за ціною

Ситуація: у мобільному застосунку є фільтр: користувач вводить **мінімальну** і **максимальну** ціну товару.

Завдання: скласти тест-кейси для перевірки:

- правильних меж (100–500)
- меж у зворотному порядку (500–100)
- однакових значень (200–200)
- нечислових значень
- порожніх полів

Задача 5. Перевірка поля вибору дати

Ситуація: у формі бронювання є поле вибору дати. Дата не може бути в минулому.

Завдання: протестувати:

- сьогоднішню дату
- дату вчорашню
- дату на 1 рік уперед
- некоректні формати (напр. 32.13.2025)
- поле порожнє

Метод тестування “еквівалентного поділу”

Задача 1. Введення віку користувача.

Умова: форма реєстрації вимагає ввести вік користувача. Допустимий вік: від 18 до 99 років включно.

Завдання: Знайдіть **класи еквівалентності** для введення віку та створіть мінімум 1 тест-кейс для кожного класу.

Задача 2. Поле для введення email.

Умова: система вимагає, щоб електронна адреса відповідала формату: *username@domain.com.*

Завдання: виділіть класи еквівалентності для введення email та сформуйте приклади тест-кейсів для кожного.

Задача 3. Замовлення квитків.

Умова: користувач може замовити від 1 до 6 квитків за раз.

Завдання: здійсніть еквівалентний поділ діапазону введених чисел. Створіть по одному тест-кейсу для кожного класу.

Задача 4. Веб-додаток: поле “Пароль”.

Умова: пароль має бути довжиною від 8 до 20 символів.

Завдання: виділіть класи еквівалентності для довжини пароля. Напишіть тест-кейси.

Задача 5. Мобільний додаток: вибір дати народження.

Умова: Користувач може вибрати дату народження в межах: **01.01.1920 – 31.12.2010**.

Завдання: визначте класи еквівалентності та створіть відповідні тест-кейси.

Задача 6. Мобільний додаток: введення номера телефону.

Умова: номер телефону має складатися з **10 цифр**, без літер чи спецсимволів.

Завдання: розділіть вхідні дані на класи та наведіть приклади.

Тестування методом аналізу граничних значень

Задача 1. Вік користувача для реєстрації.

Умова: користувач може зареєструватися, якщо його вік — **від 18 до 65 років включно**.

Завдання: визначити граничні значення, тестові значення.

Задача 2. Сума замовлення для отримання знижки.

Умова: Знижка 10% діє при замовленні **від 1000 до 5000 грн включно**.

Завдання: визначити граничні значення, тестові значення.

Задача 3. Довжина текстового поля “Ім’я” у формі.

Умова: Поле "Ім’я" дозволяє вводити **від 2 до 30 символів**.

Завдання: визначити граничні значення, тестові значення.

Техніка тестування “Причина–Наслідок”

Задача 1. Увімкнення обігрівача.

Умова: обігрівач ввімкнеться, якщо:

`C1`: Температура нижча за 18°C

`C2`: Користувач увімкнув режим "Авто"

Наслідок (`E1`): Обігрівач вмикається.

Завдання: описати можливі тестові випадки, очікувані наслідки.

Задача 2. Авторизація користувача.

Умова: користувач може увійти в систему, якщо:

C1: Введено правильний логін.
C2: Введено правильний пароль.
C3: Аккаунт активний.
Наслідок (E1): Доступ дозволено.
Завдання: описати можливі тестові випадки, очікувані наслідки.

Задача 3. Відображення повідомлення про помилку.
Умова: помилка відображається, якщо:
C1: Поле "Email" порожнє
C2: Поле "Пароль" порожнє
C3: Email не відповідає формату
Наслідок (E1): Показати повідомлення про помилку.
Завдання: описати можливі тестові випадки, очікувані наслідки.

Техніка тестування "Передбачення помилки"

Задача 1: Поле введення дати народження.
Опис: Перевірка поля введення дати народження у форматі дд.мм.рррр.
Завдання: навести приклади 2-3 передбачуваних помилок користувача.

Задача 2: Завантаження файлу.
Опис: Користувач повинен завантажити документ PDF розміром до 5 МБ.
Завдання: навести приклади 2-3 передбачуваних помилок користувача.

Задача 3: Поле вводу e-mail.
Опис: Перевірка коректності введення e-mail у формі реєстрації.
Передбачувані помилки:
Завдання: навести приклади 2-3 передбачуваних типових помилок користувача.

Повне тестування

Задача 1: Перемикач (checkbox) з 2 параметрами.
Опис: Є форма з двома прапорцями:
Прапорець 1: "Підписатися на розсилку"
Прапорець 2: "Отримувати сповіщення"

Мета: Перевірити **усі можливі комбінації** прапорців ($2^2 = 4$ варіанти).

Задача 2: Вибір типу доставки.

Опис: У користувача є **3 варіанти доставки**:

- Кур'єр
- Самовивіз
- Пошта

Мета: Перевірити **всі варіанти вибору** (3 комбінації), і переконатися, що:

- Вибирається лише один варіант
- Відповідний метод активує потрібне поле введення (наприклад, адреса для кур'єра)

Задача 3: Калькулятор логічної операції AND.

Опис: Користувач вводить два бінарні значення (0 або 1), які обробляються логічною операцією AND.

Мета: Перевірити **всі можливі комбінації** вхідних значень ($2 \times 2 = 4$ комбінації).

Позитивне тестування

Задача 1: Авторизація користувача.

Опис: Користувач вводить правильний логін і пароль.

Сформулювати очікуваний результат.

Задача 2: Додавання товару в кошик.

Опис: Користувач натискає кнопку "Додати до кошика" на сторінці товару.

Сформулювати очікуваний результат.

Задача 3: Відправлення форми зворотного зв'язку.

Опис: Користувач заповнює всі обов'язкові поля (ім'я, email, повідомлення) та натискає "Відправити".

Сформулювати очікуваний результат.

Негативне тестування

Задача 1: Авторизація з неправильним паролем.

Опис: Користувач вводить правильний логін, але **невірний пароль**.

Сформулювати очікуваний результат.

Задача 2: Заповнення форми з помилкою в email.

Опис: Користувач вводить некоректну адресу email, наприклад: `username@` або `example.com` (без @).

Сформулювати очікуваний результат.

Задача 3: Додавання в кошик без вибору розміру (одяг).

Опис: На сторінці товару (наприклад, футболка) користувач натискає "Додати до кошика", **не обравши розмір**.

Сформулювати очікуваний результат.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. Авраменко А. С., Авраменко В. С., Косенюк Г. В. Тестування програмного забезпечення : навчальний посібник. Черкаси : ЧНУ імені Богдана Хмельницького, 2017. 284 с.
2. Багрій Р. О., Петровський С. С. Особливості сучасного тестування веб-додатків. *Вісник Хмельницького національного університету*. №3, 2022 (309). С.70–74.
3. Гамільтон Т. Підручник з тестування програмного забезпечення. URL : https://www.guru99.com/uk/software-testing.html
4. Глосарій тестувальника. URL: https://beetroot.academy/glossary-qa-manual
5. Документація Cypress. URL: https://docs.cypress.io .
6. Дідковська М. В. Тестування. Основні визначення, аксіоми та принципи. Текст лекцій. Частина І. URL: https://www.quality-assurancegroup.com/book/testuvannya-osnovni-viznachennya-aksiomi-ta-printsipi-tekstlektsij-chastina-i.
7. Котенко Н. О., Жирова Т. О., Кулеба М. Б. Дослідження особливостей тестування мобільних додатків. *Управління розвитком складних систем.* 2020. № 41. С. 55–60. DOI: 10.32347/2412-9933.2020.41.55-60.
8. Навчальний ресурс з тестування програмного забезпечення URL: https://qalearning.com.ua .
9. Підручник з TestLink. URL: https://uk.myservername.com/testlink-tutorial-layman-s-guide-testlink-test-management-tool .
10. Повний посібник з автоматизації тестування програмного забезпечення. URL: https://www.zaptest.com/uk/повний-посібник-з-автоматизації-тест .
11. Попелюха. Тестування ПЗ. URL: https://www.youtube.com/@Popeliuha .
12. Сучасні технології автоматизованого проєктування та верифікації програм. Тестування програмного забезпечення : методичні вказівки до виконання

лабораторних робіт. [Електронне видання] / Уклад.: Я. Ю. Дорогий, О. О. Дорога-Іванюк. Київ : НТУУ «КПІ», 2021. 87 с.
13. Тестування програмного забезпечення систем : методичні вказівки / уклад. Є. Є. Шабала. Київ: КНУБА, 2022. 28 с.
14. Якість програмного забезпечення та тестування: базовий курс : навчальний посібник для бакалаврів галузі знань 12 «Інформаційні технології» спеціальності 121 «Інженерія програмного забезпечення». / За ред. Крепич С. Я., Співак І. Я. Тернопіль : ФОП Паляниця В.А., 2020. 478 с.
15. Beizer B. Black-box testing. New York: Wiley, 1995. 324 p.
16. Kaner Cem, Bach James, Pettichord Bret. Lessons Learned in Software Testing: A Context-Driven Approach. John Wiley & Sons, 2011. 320 p.