МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МИКОЛАЇВСЬКИЙ НАЦІОНАЛЬНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ МЕНЕДЖМЕНТУ

Кафедра економічної кібернетики,
комп'ютерних наук та інформаційних технологій

ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Методичні рекомендації до практичних занять
для здобувачів першого (бакалаврського) рівня вищої освіти
ОПП «Комп'ютерні науки» спеціальності F3 (122) «Комп'ютерні науки»
денної форми здобуття вищої освіти

Миколаїв – 2025

УДК 004.05-048.24
Т36

Друкується за рішенням науково-методичної комісії факультету менеджменту Миколаївського національного аграрного університету (протокол №1 від 28 серпня 2025 року)

Укладачі:

О. Ю. Пархоменко — канд.фіз.-математ. наук, доцент, доцент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

С. І. Тищенко — канд.педагог.наук, доцент, доцент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

С. І. Ємельянов — PhD, старший викладач кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

О. О. Жебко — асистент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

О. Є. Богатєнкова — асистент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету

Рецензенти:

Ю. В. Грицук - канд. техн. наук, доцент кафедри загальної інженерної підготовки Донбаської національної академії будівництва і архітектури

О. С. Садовий - канд. техн. наук, доцент, завідувач кафедри агроінженерії Миколаївського національного аграрного університету

Т36 **Тестування** програмного забезпечення : методичні рекомендації до практичних занять для здобувачів першого (бакалаврського) рівня вищої освіти ОПП «Комп'ютерні науки» спеціальності F3 (122) «Комп'ютерні науки» денної форми здобуття вищої освіти / уклад. О. Ю. Пархоменко, С. І. Тищенко, С. І. Ємельянов, О. О. Жебко, О. Є. Богатєнкова . Миколаїв : МНАУ, 2025. 75 с.

УДК 004.05-048.24

ЗМІСТ

ПЕРЕДМОВА

Тестування програмного забезпечення давно перестало бути просто етапом у життєвому циклі розробки – сьогодні це окрема дисципліна, що потребує глибоких теоретичних знань, розвиненого аналітичного мислення та, найголовніше, практичних навичок. Справжній фахівець із забезпечення якості має не лише розуміти, що таке дефект чи тест-кейс, але й уміти працювати з інструментами, організовувати процеси в команді, створювати документацію та ефективно комунікувати з розробниками й замовниками. Саме формуванню цих прикладних компетенцій присвячений пропонований практикум.

Навчальний посібник, який ви тримаєте в руках, є логічним продовженням теоретичного курсу з тестування програмного забезпечення та містить дванадцять практичних робіт, що охоплюють ключові аспекти діяльності сучасного QA-фахівця. Структура практикуму побудована за принципом "від простого до складного" – від знайомства з методологіями розробки до створення UML-діаграм для генерації тест-кейсів. Такий підхід дозволяє студентам поступово занурюватися в професію, закріплюючи теоретичні знання через виконання конкретних завдань.

Перші практичні роботи присвячені основам Agile-методологій. Ви навчитеся формувати Scrum-команду, створювати беклог спринту, працювати з Kanban-дошкою, а також опануєте роботу в Trello – популярному інструменті для управління проектами. Це дозволить зрозуміти, як тестування інтегрується в сучасні гнучкі процеси розробки та яку роль відіграє тестувальник у крос-функціональній команді.

Окремий блок практичних занять присвячено роботі з вимогами та історіями користувача (user stories). Ви навчитеся створювати епіки, декомпозувати їх на конкретні задачі, визначати пріоритети та оцінювати трудовитрати. Ці навички є критично важливими, адже саме з якісного формулювання вимог починається успішне тестування.

Значну увагу в практикумі приділено різним видам тестування. Ви виконаєте завдання з функціонального тестування, тестування верстки (кросбраузерного), тестування зручності використання (usability), а також технічного тестування з використанням інструментів розробника та програми Screaming Frog SEO Spider. Кожна робота передбачає створення чек-листів, виконання тестів, фіксацію результатів та оформлення баг-репортів – тобто повний цикл діяльності тестувальника.

Особливе місце посідають завдання з тест-дизайну та створення тест-кейсів. Ви навчитеся розробляти позитивні та негативні сценарії для різних форм, дотримуючись стандартів якості та вимог до оформлення. Це дозволить вам створювати документацію, яка буде зрозумілою будь-якому члену команди та придатною для подальшої автоматизації.

Завершальні роботи знайомлять із професійними інструментами – Jira Software для управління задачами та дефектами, а також UML-моделюванням для створення тест-кейсів на основі діаграм. Ви навчитеся будувати Use Case,

Activity, Sequence та State діаграми, що дозволить вам бачити систему з різних точок зору та створювати більш повні тестові набори.

Кожна практична робота містить чітко сформульовану мету, короткі теоретичні відомості, деталізоване завдання, контрольні запитання для самоперевірки та список рекомендованих джерел. Така структура дозволяє використовувати посібник як для аудиторних занять під керівництвом викладача, так і для самостійного опанування матеріалу.

Сподіваємось, що цей практикум стане для вас надійним провідником у світ професійного тестування, допоможе здобути необхідні навички та впевнено почуватися на реальних проектах. Пам’ятайте: тестування – це не просто пошук помилок, це філософія якості, і саме ви творите світ, у якому технології працюють для людей.

Бажаємо успіхів і цікавих відкриттів!

Практичне заняття № 1
Основи роботи зі Scrum, Kanban

Мета заняття:

- ознайомитися з основами Agile-методологій (Scrum і Kanban).
- дослідити ролі, артефакти та події у Scrum.
- навчитися створювати backlog та працювати з Kanban-дошкою.
- розглянути роль тестувальника в Agile-середовищі.

Структура заняття:

1. **Теоретична частина**
   - Ознайомитися з основами Scrum та Kanban за рекомендованими джерелами.
   - Розглянути ролі, артефакти та події Scrum.
   - Вивчити принципи роботи з Kanban-дошкою.
   - Проаналізувати роль тестувальника в Agile-команді.
2. **Практична частина**
   - Сформувати Scrum-команду для уявного проєкту.
   - Створити backlog спринту (мінімум 5 User Stories з Acceptance Criteria).
   - Розробити Kanban-дошку для цього проєкту.
3. **Аналіз та висновки**
   - Описати особливості тестування в Agile-процесах.
   - Проаналізувати виклики тестувальників у Scrum та Kanban.
   - Надати загальний висновок щодо опанованого матеріалу.
   - Q & A.

Короткі теоретичні положення

Scrum

- **Основна ідея:** Scrum – це фреймворк для управління складними проєктами, що працює за принципом ітерацій (спринтів), де команда працює над набором завдань протягом визначеного періоду (зазвичай 1-4 тижні).
- **Основні принципи:**
  - **Прозорість:** Всі аспекти процесу повинні бути відкритими для членів команди.
  - **Інспекція:** Регулярне перегляд результатів роботи для виявлення недоліків.
  - **Адаптація:** Швидке реагування на зміни та коригування планів.

Kanban

- **Основна ідея:** Kanban – це система візуалізації роботи, яка дозволяє керувати потоком завдань без фіксованих ітерацій.
- **Основні принципи:**

- **Візуалізація:** Завдання відображаються на дошці, що допомагає бачити їхній стан.
- **Обмеження роботи в процесі (WIP):** Встановлюються ліміти на кількість завдань, які можуть одночасно перебувати в одній колонці.
- **Управління потоком:** Аналіз та оптимізація проходження завдання від початку до завершення.
- **Безперервне покращення:** Регулярне вдосконалення процесів.

Ролі, артефакти та події Scrum

Ролі Scrum:

- **Product Owner:**
  - **Обов'язки:** Формує та пріоритезує Product Backlog, представляє інтереси замовника та ринку.
- **Scrum Master:**
  - **Обов'язки:** Допомагає команді впроваджувати Scrum, усуває перешкоди, координує процеси та сприяє безперервному покращенню.
- **Розробнича команда (Development Team):**
  - **Обов'язки:** Самоорганізована група фахівців, що відповідає за реалізацію завдань спринту.

Артефакти Scrum:

- **Product Backlog:**
  - Список всіх завдань, функціональних вимог, багів і покращень, які потрібно реалізувати.
- **Sprint Backlog:**
  - Підмножина Product Backlog, завдання якої команда обирає для виконання протягом спринту.
- **Increment:**
  - Сукупність завершених завдань, що відповідають Definition of Done, яку можна демонструвати.

Події Scrum:

- **Sprint Planning (Планування спринту):**
  - Зустріч для вибору завдань зі списку Product Backlog, що будуть виконані у спринті.
- **Daily Scrum (Щоденний stand-up):**
  - Коротка щоденна зустріч (15 хвилин), де кожен розповідає про свій прогрес, план на день та перешкоди.
- **Sprint Review (Огляд спринту):**
  - Зустріч для демонстрації завершеного інкременту та отримання зворотного зв'язку від замовника та зацікавлених сторін.
- **Sprint Retrospective (Ретроспектива спринту):**
  - Зустріч, на якій команда аналізує, що вдалося, що потрібно покращити, та планує вдосконалення наступного спринту.

Основні принципи Kanban:

- **Візуалізація роботи:**

- Завдання представляються у вигляді карток на дошці, що дозволяє всім членам команди бачити поточний стан роботи.

- **Обмеження роботи в процесі (WIP - Work In Progress):**
  - Встановлюються ліміти на кількість завдань в кожній колонці, щоб уникнути перевантаження та забезпечити фокус на завершенні поточної роботи.
- **Управління потоком:**
  - Аналіз проходження завдань через систему для виявлення вузьких місць та оптимізації процесу.
- **Прозорість процесу:**
  - Всі етапи роботи та правила (наприклад, критерії переходу між колонками) мають бути чітко задокументованими та відомими команді.
- **Безперервне покращення:**
  - Регулярне обговорення процесу, виявлення проблем та пошук шляхів для оптимізації роботи.

Роль тестувальника в Agile-команді

Тестувальник (QA) є невід'ємною частиною Agile-команди та бере активну участь у всіх етапах життєвого циклу розробки. Його роль розширюється від пошуку дефектів до забезпечення загальної якості продукту.

Основні обов'язки тестувальника:

- **Участь у плануванні:**
  - Допомога у визначенні Acceptance Criteria для кожної User Story під час планування спринту.
- **Розробка тест-кейсів:**
  - Створення ручних та автоматизованих тестів, що охоплюють основні сценарії використання продукту.
- **Проведення тестування протягом спринту:**
  - Безперервне тестування розроблених функцій, участь у daily stand-up та своєчасне інформування команди про виявлені дефекти.
- **Інтеграція з процесом розробки:**
  - Тісна співпраця з розробниками для швидкого виявлення та усунення дефектів, участь у code review та ретроспективах спринту.
- **Безперервне вдосконалення:**
  - Аналіз причин дефектів, участь у покращенні процесів тестування, впровадження нових методик та інструментів автоматизації.

Переваги інтегрованого підходу:

- **Швидке виявлення проблем:** Завдяки активній участі тестувальника у всіх етапах, дефекти виявляються ще до передачі продукту замовнику.
- **Покращення якості продукту:** Тісна взаємодія тестувальника з розробниками дозволяє оперативно вирішувати проблеми та впроваджувати покращення.
- **Гнучкість у плануванні:** Тестувальник може швидко адаптувати тестові сценарії під нові вимоги, що характерно для Agile-середовища.

Завдання:

1. Теоретичне опрацювання

Ознайомитися з рекомендованими джерелами щодо Scrum та Kanban

2. Практичне завдання

1. **Описати Scrum-команду** для уявного проекту (5-7 учасників, включаючи ролі Product Owner, Scrum Master, команда розробки та тестувальник, орієнтовні предметні області проєктів – див. нижче).

2. **Створити backlog спринту** (мінімум 5 User Stories, кожна з Acceptance Criteria).

3. **Розробити Kanban-дошку** для цього проекту (можна у вигляді таблиці в Excel, Miro тощо).

Приклад у вигляді таблиці:

To Do	In Progress	Testing	Done
Реєстрація через email	Зміна пароля	Перевірка UI-реєстрації	Форма логіну
Відновлення пароля	Авторизація		

3. Аналіз та висновки

1. Опишіть основні виклики тестування в Agile-середовищі.

2. Проаналізуйте переваги та недоліки роботи тестувальника у Scrum та Kanban.

3. Напишіть загальний висновок щодо виконаної роботи (які інструменти були використані, що вдалося дізнатися нового, які складнощі виникли).

Короткий опис предметних областей для проєктів

1. Банківська система

Ви працюєте в банку, що надає широкий спектр фінансових послуг. Система обробляє дані про клієнтів, рахунки, транзакції та кредитні продукти. Для кожного клієнта зберігається інформація (ПІБ, адреса, контактні дані) та дані про відкриті рахунки (тип рахунку, баланс, історія операцій). Також система підтримує кредитування: при оформленні кредитного договору фіксується сума кредиту, процентна ставка, терміни погашення та графік платежів. Тестування має перевіряти коректність обчислень, захищеність транзакцій і правильність обробки даних.

2. Інтернет-магазин книг

Ви працюєте в системі онлайн-продажу книг. Каталог містить інформацію про книги (назва, автор, ISBN, жанр, опис, ціна) та їхню наявність. Клієнти можуть переглядати каталог, додавати книги до кошика, оформлювати замовлення, обирати спосіб доставки та оплати. Система також дозволяє залишати відгуки та рейтинги книг. Тестування повинно охоплювати перевірку пошуку, фільтрації, обчислення вартості замовлення, а також інтеграцію з платіжними системами.

3. Система управління університетом

Ви працюєте в інформаційній системі університету, що об’єднує дані про студентів, викладачів, курси та розклад занять. Для студентів зберігається інформація про ПІБ, студентський квиток, спеціальність та результати навчання. Курси мають унікальний код, назву, опис, кількість кредитів і розклад занять. Викладачі можуть публікувати завдання, оцінювати роботи студентів та вести комунікацію. Тестування має включати перевірку коректності обробки даних, розрахунку середніх балів та роботи пошукових функцій.

4. Медична клініка

Ви працюєте в системі управління медичною клінікою, яка веде облік пацієнтів, лікарів, записів на прийоми та медичної документації. Для кожного пацієнта зберігається особиста інформація, історія хвороб, записи про прийоми та призначення лікарів. Лікарі мають графік прийомів, можуть вносити діагнози та призначати лікування, а також отримувати результати лабораторних аналізів. Тестування повинно перевіряти захищеність даних, правильність обробки записів, інтеграцію з лабораторними системами та зручність інтерфейсу для користувачів.

5. Система бронювання авіаквитків

Ви працюєте в системі бронювання авіаквитків, що забезпечує пошук рейсів, бронювання місць і оплату квитків. Система містить дані про авіакомпанії, рейси (маршрути, дати, час відправлення/прибуття, тип літака) та аеропорти. Кожен рейс має інформацію про кількість вільних місць, класи обслуговування і можливості додаткових послуг (багаж, вибір місця). Тестування має зосередитись на пошукових алгоритмах, правильності розрахунку вартості, резервуванні місць та інтеграції з платіжними системами.

6. Готельна система

Ви працюєте в системі управління готелем, що включає бронювання номерів, реєстрацію гостей та розрахунок оплати. Система містить інформацію про номери (тип, вартість, доступність), додаткові послуги (ресторан, спа, трансфер) та дані гостей. При оформленні бронювання гості вказують дати заїзду/виїзду, вибір номера і додаткові побажання. Тестування має охоплювати перевірку бронювання, коректність розрахунку оплати, обробку скасувань та відгуків клієнтів.

7. Система управління складом

Ви працюєте в системі управління складом, яка відслідковує надходження, зберігання та видачу товарів. Для кожного товару зберігаються дані: назва, артикул, опис, кількість на складі, місце зберігання, дата надходження та термін придатності. Система дозволяє вести операції прийому поставок, відвантаження замовлень, інвентаризацію та переміщення товарів між складами. Тестування повинно перевіряти коректність оновлення залишків, точність звітів і належну обробку операцій.

8. Соціальна мережа

Ви працюєте в системі соціальної мережі, що дозволяє користувачам створювати профілі, публікувати пости, обмінюватися повідомленнями та взаємодіяти через коментарі й лайки. Кожен користувач має профіль з особистою інформацією, фото та списком друзів. Система підтримує публічні та приватні повідомлення, а також сповіщення про активність друзів. Тестування має зосередитись на перевірці безпеки даних, коректності публікацій і зручності навігації по платформі.

9. Ресторанна система замовлень

Ви працюєте в системі замовлень для ресторану, яка обслуговує прийом замовлень, управління меню та оплату. Система містить інформацію про страви (назва, опис, ціна, інгредієнти), категорії меню та наявність. Клієнти можуть робити замовлення через веб-сайт або мобільний додаток, вибираючи страви, додавати коментарі до замовлення та здійснювати оплату онлайн. Система також відслідковує статус замовлення від прийому до доставки чи видачі. Тестування має перевіряти коректність формування замовлень, розрахунку сум і інтеграцію з платіжними та кур'єрськими сервісами.

10. Електронна бібліотека

Ви працюєте над системою електронної бібліотеки, яка забезпечує доступ до електронних книг, аудіокниг та журналів. Система містить дані про книги (назва, автор, ISBN, жанр, рік видання, опис), інформацію про користувачів (ПІБ, електронна адреса, історія позичань) та облік операцій – позичання, повернення, бронювання. Також система відстежує наявність копій та штрафи за прострочене повернення. Тестування повинно охоплювати перевірку функціоналу пошуку, бронювання, коректності обліку повернень і нарахування штрафів.

11. Платформа онлайн-навчання

Ця система підтримує проведення дистанційного навчання. Вона об'єднує курси, уроки, відеолекції, завдання та форуми для обговорень. Для кожного курсу зберігається інформація про назву, опис, розклад занять, викладачів та список студентів. Студенти можуть проходити онлайн-тести, завантажувати домашні завдання, брати участь у вебінарах, а також отримувати оцінки. Тестування має перевіряти функціональність системи управління контентом, автентифікацію користувачів, роботу форумів і систему оцінювання.

12. CRM-система для управління продажами

Ви працюєте над CRM-системою, яка допомагає компанії управляти відносинами з клієнтами, відстежувати потенційних клієнтів (ліди), укладати угоди та контролювати взаємодію з клієнтами. Система зберігає інформацію про клієнтів (контактні дані, історія покупок, комунікацій), завдання для менеджерів з продажу, зустрічі та нагадування. Важливими аспектами є

інтеграція з електронною поштою, автоматизація розсилок та генерація звітів про продажі. Тестування повинно включати перевірку бізнес-логіки, роботи пошукових фільтрів та безпеки даних.

13. Система бронювання таксі

Ви розробляєте систему, що дозволяє користувачам замовляти таксі через мобільний додаток або веб-сайт. Система включає інформацію про водіїв (ПІБ, транспортний засіб, рейтинг), замовлення (адреса відправлення та призначення, час, вартість поїздки), інтеграцію з картами для відстеження маршруту та розрахунок тарифів. Також система має повідомляти користувачів про прибуття таксі та дозволяти залишати відгуки. Тестування має зосередитись на інтеграції з GPS, розрахунку вартості, обробці платежів та обробці великої кількості запитів одночасно.

14. Платформа доставки їжі

Система призначена для замовлення їжі з ресторанів. Вона містить інформацію про ресторани (назва, адреса, рейтинг), меню (страви, опис, ціни, інгредієнти), замовлення (список обраних страв, час доставки, адреса замовлення) та платіжні операції. Користувачі можуть робити замовлення онлайн, відслідковувати статус доставки, залишати відгуки та оцінки ресторанам. Тестування повинно включати перевірку роботи фільтрів, розрахунку сум замовлень, інтеграції з платіжними системами та алгоритмів маршрутизації замовлень.

15. Система управління подіями

Ви розробляєте систему, що організовує заходи: конференції, концерти, тренінги тощо. Система містить дані про події (назва, опис, дата, місце проведення, список спікерів/виконавців), реєстрацію учасників, продаж квитків та інтеграцію з системами оплати. Організатори можуть створювати розклади заходів, керувати списком учасників та надсилати повідомлення про зміни в програмі. Тестування має охоплювати перевірку реєстрації, коректність обробки платежів, генерацію квитків та роботу повідомлень.

16. Система розумного дому

Ви працюєте в системі розумного дому, яка дозволяє користувачам керувати різними пристроями будинку через мобільний додаток або веб-інтерфейс. Система інтегрує дані про управління освітленням, опаленням, безпекою та мультимедійними пристроями. Для кожного пристрою зберігається інформація про його назву, тип, місцезнаходження, поточний стан та історію подій. Тестування має включати перевірку автоматичних сценаріїв (наприклад, увімкнення світла при виявленні руху), сумісності пристроїв, безпеки доступу та коректності відображення даних.

17. Мобільний додаток для спільних поїздок

Ви працюєте над мобільним додатком, який дозволяє користувачам організовувати спільні поїздки. Додаток забезпечує пошук маршрутів, бронювання місць, оцінку водіїв та пасажирів, а також інтегрується з платіжними системами для онлайн-оплати. Зберігається інформація про користувачів, їх місцезнаходження, історію поїздок та рейтинги. Тестування має включати перевірку роботи GPS, алгоритмів розрахунку вартості, обробки бронювань та інтеграції з платіжними сервісами.

18. Платформа для торгівлі криптовалютами

Ви працюєте над платформою, що забезпечує купівлю, продаж та обмін криптовалютами. Платформа відображає актуальні курси валют, історію транзакцій, графіки змін, а також інтегрується з API бірж для отримання даних у реальному часі. Зберігається інформація про користувачів, їх баланс, виконані операції та налаштування безпеки. Тестування має включати перевірку точності розрахунків, захищеності транзакцій, стабільності системи під високим навантаженням та коректності інтеграції з API бірж.

19. Система моніторингу навколишнього середовища на базі IoT

Ви працюєте в системі моніторингу навколишнього середовища, яка використовує мережу сенсорів для збору даних про температуру, вологість, рівень шуму та забруднення повітря. Дані передаються на центральний сервер, де вони аналізуються та відображаються у вигляді графіків і сповіщень. Для кожного сенсора зберігається інформація про його тип, місцезнаходження та останні зчитані значення. Тестування має включати перевірку стабільності зчитування даних, надійності передачі інформації, коректності аналітичних алгоритмів та відображення звітів.

20. Мобільний додаток для здоров’я та фітнесу

Ви працюєте над мобільним додатком для здоров’я та фітнесу, який дозволяє користувачам відстежувати фізичну активність, харчування, сон та тренування. Додаток інтегрується з носимими пристроями, зберігає дані про активність користувачів (кількість кроків, спалені калорії, тривалість сну) та надає персоналізовані рекомендації. Тестування має включати перевірку коректності синхронізації даних з носимими пристроями, точності вимірювань, зручності інтерфейсу та інтеграції з соціальними мережами для обміну досягненнями.

Контрольні питання до практичного заняття

1. Що таке Scrum і які його основні принципи?
2. Які основні ролі існують у Scrum і які їхні обов’язки?
3. Що таке Product Backlog і як він формується?
4. Чим відрізняється Sprint Backlog від Product Backlog?
5. Які основні події проводяться в Scrum і яке їхнє призначення?
6. Що таке Kanban-дошка і які колонки вона має містити?

7. Які основні відмінності між Scrum та Kanban?
8. Що таке WIP-ліміт і чому він важливий в Kanban?
9. Як організувати процес пріоритезації завдань у Product Backlog?
10. Яку роль відіграє тестувальник в Agile-команді?

Список джерел для додаткового опанування:

1. Scrum Guide – URL: https://scrumguides.org
2. Kanban метод – URL: https://kanbanize.com/kanban-resources/getting-started/what-is-kanban
3. Книга "Scrum: The Art of Doing Twice the Work in Half the Time" – Jeff Sutherland – URL: https://www.agileleanhouse.com/lib/lib/News/More_Praise_for_Scrum_The_Art_of_Doing_T.pdf
4. Книга "Kanban: Successful Evolutionary Change for Your Technology Business" – David J. Anderson – URL: https://pdfhost.io/v/KwUWPfb6X_Kanban__Successful_Evolutionary_Change_for_Your_Technology_Business_PDFDrive_
5. Coursera курс "Agile with Atlassian Jira" – URL: https://www.coursera.org/learn/agile-atlassian-jira

Практичне заняття № 2
Створення Scrum-дошки у Trello

Мета:

- Ознайомити студентів із практичним використанням Scrum-дошки в Trello.
- Навчити створювати та керувати backlog-ом.
- Визначити структуру Scrum-дошки та її основні елементи.
- Оволодіння практичними навичками в управлінні проєктом у середовищі онлайн-застосунку Trello
- Закріпити навички організації командної роботи за допомогою Trello.

Структура заняття:

1. Теоретична частина

- Огляд Scrum-дошки: які колонки вона повинна містити.
- Відмінності між Scrum та Kanban у Trello.
- Як організувати backlog та спринти.
- Використання міток, дедлайнів, чек-листів та коментарів.

2. Практична частина

- Реєстрація та налаштування акаунта в Trello.
- Створення Scrum-дошки з основними колонками.
- Додавання та управління завданнями (User Stories).
- Робота з тегами, дедлайнами та чек-листами.

3. Аналіз та висновки

- Аналіз отриманого досвіду.
- Обговорення особливостей організації тестування в Scrum.
- Q & A.

Короткі теоретичні положення

1. Що таке Scrum-дошка?

Scrum-дошка – це візуальний інструмент, що допомагає команді керувати спринтом. Вона складається з колонок, які відображають статуси завдань, і карток (завдань), що переміщуються між колонками залежно від їхнього стану.

Основні **елементи Scrum-дошки**:

- **Колонки** (Product Backlog, Sprint Backlog, In Progress, Testing, Done).
- **Картки (tasks)** – окремі завдання, що додаються до дошки.
- **Мітки (labels)** – допомагають класифікувати задачі за типом (наприклад, "тестування", "розробка", "високий пріоритет").
- **Чек-листи** – використовуються для розбиття завдання на дрібніші підзадачі.
- **Дедлайни** – встановлення термінів виконання завдань.
- **Коментарі та вкладення** – дозволяють членам команди взаємодіяти та надавати додаткову інформацію.

2. Відмінності між Scrum та Kanban у Trello

Trello підтримує як Scrum, так і Kanban, але вони мають різні підходи до організації процесу (табл. 2.1).

Таблиця 2.1.

Відмінності між Scrum та Kanban у Trello

Характеристика	**Scrum**	**Kanban**
Орієнтація	На ітерації (спринти)	На безперервний потік завдань
Обмеження роботи	Завдання фіксуються на початку спринту	Встановлюється ліміт WIP (Work in Progress)
Відображення завдань	Sprint Backlog → In Progress → Testing → Done	To Do → In Progress → Testing → Done
Гнучкість	Менше змін під час спринту	Завдання можуть додаватися або змінюватися в будь-який час

В **Scrum** важливо фіксувати завдання на початку спринту, тоді як у **Kanban** робота ведеться в постійному потоці.

3. Налаштування Scrum-дошки в Trello

3.1. Реєстрація та створення дошки

1. Зареєструватися в Trello (якщо ще не маєте акаунта) – https://trello.com.
2. Створити нову дошку: **"Scrum-дошка – Назва проєкту"**.
3. Визначити ключові колонки:
   - **Product Backlog** – список усіх запланованих завдань.
   - **Sprint Backlog** – завдання, що будуть виконані в цьому спринті.
   - **In Progress** – завдання, які наразі виконуються.
   - **Testing** – завдання, що проходять тестування.
   - **Done** – завершені завдання.

3.2. Додавання та управління завданнями

- **User Stories:** створюються як картки в **Product Backlog**.
- **Переміщення карток:** коли спринт розпочинається, завдання перетягуються в **Sprint Backlog**.
- **Назначення виконавців:** у кожній картці можна вказати відповідальних осіб.
- **Додавання чек-листів:** наприклад, для тестування можна додати підзадачі "Функціональні тести", "UI-тести", "Перевірка логування".
- **Встановлення міток:**
  - Жовтий – "В роботі"
  - Червоний – "Блокуюча проблема"

- Синій – "Потребує тестування"

3.3. Робота зі спринтами

- Спринт триває **1-4 тижні**.
- Наприкінці спринту проводиться **Sprint Review та Retrospective**.
- Усі виконані завдання переносяться в **Done**, а невиконані – або в наступний спринт, або повертаються в **Product Backlog**.

4. Роль тестувальника у Scrum

У Scrum тестувальники виконують такі завдання:

- Визначають **Acceptance Criteria** для кожної User Story.
- Створюють тест-кейси та чек-листи.
- Виконують функціональне тестування перед перенесенням картки в "Done".
- Взаємодіють із розробниками для швидкого усунення багів.
- Підтримують автоматизацію тестування (якщо можливо).

5. Виклики тестування в Scrum

- **Швидкі зміни в проєкті** – тестувальники мають оперативно адаптуватися.
- **Обмежений час на тестування** – тестування проводиться в межах спринту.
- **Відсутність детальної документації** – необхідно активно спілкуватися з командою.
- **Автоматизація тестування** – бажано впроваджувати тест-скрипти для регресійного тестування.

6. Принципи управління проєктом у системі Trello

Trello – це система, яка успадкувала принципи японської методології канбан-дошок і є спрямованою на організацію персональної роботи або роботи невеликої команди. Для довідки канбан – це система організації виробництва і постачання, яка дозволяє здійснити принцип "якраз вчасно" на основі теоретичних основ Ф. Тейлора, Г. Форда. Слово "канбан" з японської означає рекламний щит, вивіска. Систему створено компанією "Тойота" 1959 р. Із 1962 р. Тойота почала впроваджувати систему в усі виробничі процеси. Класична канбан-дошка складається з карток і має такий вигляд (рис. 2.1). Картки використовують для організації завдань і розподіляють за типами. Переважно завдання розподіляють на такі типи:

- заплановані (To do);
- поточні (In Progress);
- виконані (Done)

To do	In Progress	Done
Use Kanban Subscribe to Kanban Tool	Learn about Kanban	Get some Sticky Notes

Рис. 2.1. Канбан-дошка (спрощений варіант)

Структура Trello складається з дошок, розподілених на списки, які складаються з карток (рис. 2.2). Кожну з дошок можна виділяти під конкретні проєкти, процеси.

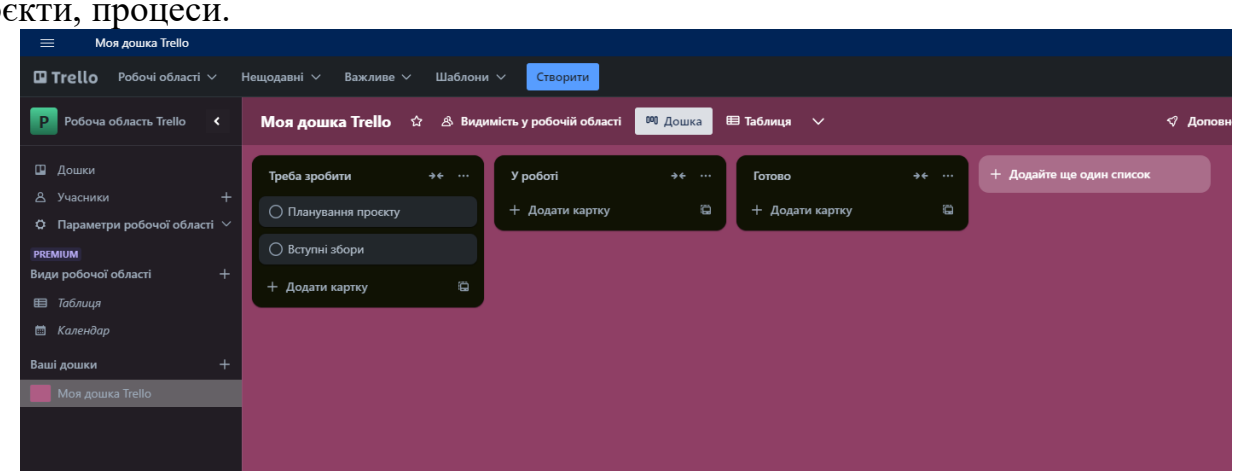

Рис. 2.2. Структура Trello

Дошка (Board): Це основний простір, де організовано всю інформацію про проєкт чи процес. Дошка може представляти конкретний проєкт, команду або робочий процес, і містить усі списки та картки, які допомагають візуально відслідковувати завдання.

Список (List): Список – це стовпчик на дошці, який використовується для групування завдань за певним критерієм або стадією робочого процесу. Наприклад, типові списки можуть бути "To Do" (Завдання до виконання), "In Progress" (В роботі) та "Done" (Виконано). Списки допомагають розбити проєкт на окремі етапи та забезпечують логічну організацію завдань.

Картка (Card): Картка – це окрема одиниця завдання або елемент роботи в межах списку. Вона містить деталі завдання: опис, чек-листи, прикріплені файли, мітки, дедлайни, коментарі та іншу інформацію. Картки можна переміщувати між списками, що дозволяє відслідковувати прогрес виконання завдання.

Trello пропонує безліч корисних можливостей для оформлення, налаштування й управління своїми функціональними елементами.

Списки можна копіювати, переміщати й архівувати. Меню з дошками у Trello можна зробити фіксованим, а самі дошки додавати у "Вибрані" та сортувати.

Є три типи дошок із різним рівнем доступу: приватна (доступна тільки на особисте запрошення власника дошки); командна (доступна всім учасникам команди); публічна (може бути доступна всім).

Закриті дошки та непотрібні списки з картками зберігають в спеціальному архіві. Звідти їх можна повернути назад або остаточно видалити. Можна створювати необмежену кількість завдань, дошок і списків, а також додавати будь-яку кількість учасників.

Картки можуть бути як простим описом завдання, так і складним документом зі списками, чек-листами, укладеннями, термінами, мітками, відповідальними особами тощо.

Картки у Trello можна:

- перейменувати, додати й редагувати опис;
- надати мітки, учасників, строк виконання, додати файл або чек-лист;
- додати коментарі, смайли, укладення, інші завдання, повідомити вибраних учасників;
- змінити положення у списку, переміщати її між списками та іншими дошками;
- копіювати, стежити за змінами, архівувати;
- роздрукувати, експортувати, поділитися посиланням на картку або її поштову адресу;
- переглянути докладний журнал: хто, коли і які дії вчиняв;
- видалити.

Основною перевагою Trello є можливість бачити кілька одночасно запущених проєктів і їхній стан у поточний момент часу. Для керівника проєкту або інших учасників команди, які працюють над проєктами з кінцевою датою виконання або певною метою, онлайн сервіс може відобразити процес роботи над проєктом у будь-який момент часу (тобто в режимі реального часу).

Наявна можливість налаштувати Trello у такий спосіб, щоб кожна дошка показувала стан будь-якого проєкту. Картки до того ж мають безліч можливостей: обговорення, голосування, додавання файлів, установлення дедлайнів, призначення кольорових і текстових міток, розподіл завдань за виконавцями, можливість спостереження за всіма змінами, станами учасників команди в режимі реального часу.

Trello дозволяє працювати як в режимі онлайн (в браузері), так і встановити додаток на комп’ютер.

Індивідуальне завдання:

Завдання:

1. **Зареєструватися в Trello (якщо ще не маєте акаунта).**
2. **Створити Scrum-дошку для уявного проєкту.**
3. **Додати колонки:**
   - **Product Backlog** – загальний список завдань.
   - **Sprint Backlog** – завдання, відібрані для поточного спринту.
   - **In Progress** – завдання, які зараз виконуються.
   - **Testing** – завдання, що перебувають на тестуванні.
   - **Done** – виконані завдання.
4. **Додати мінімум 5 User Stories** у Product Backlog.
5. **Перемістити 2 User Stories до Sprint Backlog** та розподілити між членами команди.
6. **Додати чек-лист до одного з завдань** (наприклад, для тестування).
7. **Додати дедлайн та мітки (labels)** до одного із завдань.

8. **Прикріпити знімок екрана готової Scrum-дошки** у звіт.

Приклад виконання завдання:

1. **Створення дошки в Trello**
   - Назва дошки: *Scrum Project – E-commerce Platform*
   - Колонки: Product Backlog, Sprint Backlog, In Progress, Testing, Done
2. **User Stories у Product Backlog:**

User Story	Acceptance Criteria
Як користувач, я хочу реєструватися через email	Поля для введення email та пароля; перевірка email; повідомлення про реєстрацію.
Як користувач, я хочу змінювати пароль	Поле введення старого та нового пароля; валідація складності пароля; повідомлення про успішну зміну.
Як адміністратор, я хочу бачити список зареєстрованих користувачів	Відображення таблиці з користувачами; можливість пошуку.

3. **Розподіл завдань у Sprint Backlog:**
   - **Реєстрація через email** (Призначено Івану)
   - **Зміна пароля** (Призначено Марії)
4. **Додавання чек-листа:**
   - Наприклад, для задачі *"Реєстрація через email"*:
     - Валідація email
     - Обробка помилок
     - Надсилання листа підтвердження
5. **Додавання дедлайну та міток:**
   - Дедлайн для *"Реєстрація через email"* – 7 днів
   - Мітка: "High Priority" (червона)
6. **Знімок екрана готової Scrum-дошки**

Список джерел для додаткового опанування:

1. Scrum Guide – URL: https://scrumguides.org
2. Офіційний сайт Trello – URL: https://trello.com
3. Основи роботи з дошками Trello – URL: https://trello.com/uk/guide/trello-101
4. Книга "Scrum: The Art of Doing Twice the Work in Half the Time" – Jeff Sutherland – URL: https://www.agileleanhouse.com/lib/lib/News/More_Praise_for_Scrum_The_Art_of_Doing_T.pdf
5. YouTube-курси "How to Use Trello for Project Management"
6. Coursera курс "Agile with Atlassian Jira" – URL: https://www.coursera.org/learn/agile-atlassian-jira

При підготовці використано матеріали:

1. Робота з системою Trello – URL: https://training.qatestlab.com/blog/technical-articles/working-with-the-trello-system/

2. Сучасні методології та середовища розроблення комп’ютерних інформаційних систем [Електронний ресурс]: методичні рекомендації до виконання лабораторних робіт для студентів спеціальності 122 "Комп’ютерні науки" другого (магістерського) рівня / уклад. І. О. Ушакова, І. Б. Медведєва. – Харків : ХНЕУ ім. С. Кузнеця, 2022. – 77 с.

Практична робота №3
Створення історії користувача (User Story)

Мета роботи:

1. Набуття навичок у написанні історій користувача.
2. Створення історії користувача для опису вимог до розроблюваної системи.

Теоретичні положення

Історія користувача (User Story) – це спосіб опису вимог до системи, сформульованих як одна або більше пропозицій на повсякденній або діловій мові користувача.

Стандартизований шаблон опису User Story має такий вигляд:

Як <роль> я можу <дію>, так що <бізнес-цінність>,

де **роль** – визначає того, хто виконує дію або того, хто отримує користь від дії. Це навіть може бути інша система, якщо вона ініціює дію; **дія** – становить виконувану системою активність; **бізнес цінність** – становить користь для бізнесу.

Велику історію користувача називають *епіком*. Відмінності історії користувача та епіка:

епік завжди більше, ніж історія за функціональністю, і може не вміщатися в одній ітерації;

епік має явну цінність для користувача (наприклад, сформувати співтовариство читачів і коментаторів блогу, із якими можна обговорювати цікаві теми), а історії – просто невеличку функціональність.

Для реалізації історій користувачів необхідно визначити їхній пріоритет та оцінити трудовитрати.

Пріоритет (Backlog Priority) – це порядок, у якому *Власник продукту* (Product Owner, PO) хоче отримувати історії. Спочатку рекомендовано робити високорівневу пріоритезацію, щоб сфокусувати увагу команди на тих історіях, які на початковому етапі було вибрано як найважливіші. Для цього можна використовувати просту метрику з декількома такими значеннями, як: **Must** (*обов'язково*), Need (*необхідно*), Want (*бажано*). Замість слів можна використовувати цифри: 1 – Must, 2 – Need, 3 – Want або букви: L – Low (низький), M – Medium (середній) і H – High (високий).

Оцінювання трудовитрат (Estimated Effort) – це витрати праці на реалізацію історії, оцінені командою та виявлені, зазвичай, у Story Points.

Story Point – це абстрактна одиниця розміру, що має сенс тільки для цього проєкту. Для оцінювання у Story Points переважно використовують не абсолютні, а відносні оцінки. Щоб оцінити завдання у Story Points, необхідно з усіх завдань вибрати ту, яка буде передостанньою за тривалістю (майже найкоротшою). Уважають, що це завдання займе 1 Story Point. Для всіх інших завдань визначають, у скільки разів вони більше, і ставлять відповідну оцінку у Story Points. Для переведення оцінок зі Story Points у витрати часу

використовують оцінку декількох завдань в одиницях часу (наприклад, днях чи годинах). Цю оцінку можуть поставити як члени команди, так і експерти. Потім на підставі цих оцінок обчислюють, скільки приблизно знадобиться реального часу, щоб зробити 1 Story Point.

Спочатку для оцінювання історій користувача рекомендовано використовувати систему високорівневого оцінювання, яку буде зручно застосовувати. На цьому етапі можна використовувати просту шкалу: **S** – **Small** *(маленька*), **M** – **Medium** (*середня*) та **L** – **Large** (*велика*). Ці оцінки не є остаточними, і в майбутньому буде необхідно більш точно оцінити трудовитрати.

Методичні рекомендації до виконання завдань 3.1 – 3.3

Завдання 3.1. Опишіть історії користувача. У завданні 3.1 потрібно описати епічні історії користувача.

Порядок виконання

1. Вивчіть теоретичний матеріал про історії користувача.
2. Проведіть із командою мітинг для створення історій користувача, відповідно до раніше створених бачень і портретів персонажів.
3. Визначте по одній великій історії користувача (епіки) на кожного члена команди. Опишіть історії користувача, використовуючи шаблон.
4. Для кожної історії визначте назву (заголовок) і надайте ID.
5. Обговоріть із замовником / обговоріть у команді пріоритет User Story. Розставте пріоритети для історій, використовуючи найпростішу шкалу: L – Low (низький), M – Medium (середній) і H – High (високий).
6. Обговоріть із командою початкові високорівневі оцінки трудовитрат, необхідні на реалізацію User Story. Поставте оцінки, використовуючи найпростішу шкалу: S – Small (маленька), M – Medium (середня) та L – Large (велика).
7. Складіть перелік історій користувача у вигляді табл. 3.1.

Таблиця 3.1

Перелік історій користувача

Ідентифікатор (ID)	Заголовок (Title)	Історія користувача (User Story)	Пріоритет (Backlog Priority)	Оцінювання трудовитрат (Estimated Effort)

Завдання 3.2. Декомпозуйте історію користувача.

У завданні 3.2 потрібно декомпозувати великі історії користувача, використовуючи патерни.

Порядок виконання

1. Виконайте декомпозицію великих історій користувача за такими патернами: послідовністю дій; варіацією бізнес-правил; основними трудовитратами; простотою / складністю; варіацією даних; методами введення даних; відстроченням якостей системи; операціями (наприклад: CRUD); сценаріями використання. Результати наведіть у таблиці (табл. 3.2).

Таблиця 3.2

Декомпозиція історії користувача

Патерн декомпозиції	ID	Велика історія користувача (Epic)	ID	Історія користувача (User Story)

Завдання 3.3. Деталізуйте опис історій користувача.

У завданні 3.3 потрібно зробити детальний опис історій користувача.

Детальний опис історії користувача допомагає розробникам і тестувальникам бачити основні проблеми, які має бути вирішено в межах історії.

Детальний опис історії користувача містить:

критерії приймання для того, щоб зрозуміти, коли історію завершено; приймальні тести. В ідеальному випадку історії користувача використовують як легковажну тестову документацію, у якій фіксуються тест-кейси, наприклад, у вигляді сценаріїв або специфікацій;

технічні нотатки. Сюди часто потрапляє інформація про обмеження системи, наприклад, про необхідність підтримувати певний формат даних;

опрацювання помилок. Наводять рекомендації, які необхідно виконати в разі виникнення помилок.

Порядок виконання

1. Додайте таку інформацію для кожної історії користувача:

критерії приймання;

нотатки.

До нотаток у разі потреби можуть додавати: угоди; приймальне тестування; технічні нотатки; опрацювання помилок.

2. Результати детального опису історії користувача подайте у вигляді таблиці Excel (табл. 3.3).

Таблиця 3.3

Детальний опис історії користувача

ID	
Заголовок	
Опис	
Пріоритет	
Розмір	
Критерії приймання	
Нотатки*	
Угоди	
Тестування	
Технічні нотатки	
Опрацювання помилок	

*Нотатки можуть мати посилання на інші документи.

Зміст звіту

1. Мета роботи.
2. Опис епічних історій користувача. Таблиця "Перелік історій користувача".
3. Декомпозиція історій користувача. Таблиця "Декомпозиція історій користувача".
4. Деталізація історій користувача. Таблиця "Детальний опис історії користувача" (по дві таблиці на кожного члена команди).
5. Висновки.

Контрольні запитання до лабораторної роботи 4

1. Що таке історія користувача?
2. Наведіть шаблон опису історії користувача.
3. Що таке епік?
4. Що таке пріоритет історії користувача?
5. Як оцінити трудовитрати в Story Point?
6. Яку шкалу рекомендовано використовувати для високорівневого оцінювання трудовитрат для історії користувача?
7. Що вносять до детального опису історії користувача?

При підготовці використано матеріали:

1. Сучасні методології та середовища розроблення комп’ютерних інформаційних систем [Електронний ресурс]: методичні рекомендації до виконання лабораторних робіт для студентів спеціальності 122 "Комп’ютерні науки" другого (магістерського) рівня / уклад. І. О. Ушакова, І. Б. Медведєва. – Харків : ХНЕУ ім. С. Кузнеця, 2022. – 77 с.

Практична робота №4.
Види тестування

Короткі теоретичні відомості

Види тестування програмного забезпечення

1. За ступенем автоматизації

- **Ручне тестування (manual testing):** Виконується тестувальником, який самостійно проводить перевірку функціональності програми згідно з тест-кейсами.
- **Автоматизоване тестування (automated testing):** Тести виконуються за допомогою спеціальних інструментів, що дозволяє економити час та знижувати кількість помилок, пов'язаних з людським фактором.
- **Напівавтоматизоване тестування (semiautomated testing):** Поєднує елементи як ручного, так і автоматизованого тестування, де певні кроки виконуються автоматично, а інші – вручну.

2. За ступенем підготовленості до тестування

- **Тестування по документації (formal testing):** Планується і виконується згідно з детальною документацією, що містить тест-плани, тест-кейси, вимоги та специфікації.
- **Тестування ad hoc або інтуїтивне тестування (ad hoc testing):** Відбувається без заздалегідь підготовленого плану, спираючись на досвід тестувальника та інтуїцію щодо можливих помилок.

3. За знанням системи

- **Тестування чорної скриньки (black box):** Тестувальник не має доступу до внутрішньої логіки або коду системи, тестування базується на зовнішніх функціональних характеристиках.
- **Тестування білої скриньки (white box):** Вимагає знання внутрішньої логіки, структури коду та алгоритмів, що дозволяє перевірити кожну деталь реалізації.
- **Тестування сірої скриньки (grey box):** Поєднує підходи чорної та білої скриньки, коли тестувальник має часткове розуміння внутрішніх процесів системи.

4. За ступенем ізольованості компонентів

- **Компонентне (модульне) тестування (component/unit testing):** Перевірка окремих модулів або компонентів програми ізольовано від інших частин.
- **Інтеграційне тестування (integration testing):** Фокусується на перевірці взаємодії між різними модулями, щоб виявити проблеми, що виникають при їх взаємодії.
- **Системне тестування (system/end-to-end testing):** Оцінка роботи всієї системи як єдиного цілого, де тестуються як внутрішні, так і зовнішні взаємодії.

5. За часом проведення тестування

- **Альфа-тестування (alpha testing):** Проводиться розробниками або спеціальною командою тестування на ранніх стадіях розробки перед офіційним запуском продукту.
- **Тестування при прийманні або Димове тестування (smoke testing):** Перевірка базової функціональності системи після внесення змін, щоб переконатися, що продукт готовий для подальшого тестування.
- **Тестування нової функціональності (new feature testing):** Окрема перевірка нових можливостей, що були додані до продукту.
- **Регресивне тестування (regression testing):** Перевірка, чи не вплинули внесені зміни або виправлення на існуючий функціонал.
- **Тестування при здачі (acceptance testing):** Проводиться замовником або кінцевим користувачем для підтвердження відповідності продукту вимогам.
- **Бета-тестування (beta testing):** Продукт тестується групою кінцевих користувачів у реальному середовищі перед остаточним релізом.

6. За об'єктом тестування

- **Функціональне тестування (functional testing):** Перевірка відповідності роботи програми вимогам і специфікаціям.
- **Тестування продуктивності (performance testing):** Оцінка швидкодії та ефективності роботи системи під певним навантаженням.
- **Навантажувальне тестування (load testing):** Визначення поведінки системи при збільшенні навантаження до максимальної межі її можливостей.
- **Стрес-тестування (stress testing):** Аналіз стабільності роботи програми за умов екстремального навантаження або збоїв ресурсів.
- **Тестування стабільності (stability/endurance/soak testing):** Перевірка, як система функціонує протягом тривалого часу під нормальним або підвищеним навантаженням.
- **Тестування зручності використання або Юзабіліті-тестування (usability testing):** Оцінка зручності та інтуїтивності інтерфейсу для кінцевих користувачів.
- **Тестування інтерфейсу користувача (UI testing):** Перевірка графічного інтерфейсу, елементів дизайну та їх взаємодії з користувачем.
- **Тестування безпеки (security testing):** Виявлення вразливостей системи для запобігання несанкціонованому доступу та іншим загрозам.
- **Тестування локалізації (localization testing):** Перевірка адаптації продукту для роботи в різних регіонах та мовних середовищах.
- **Тестування сумісності (compatibility testing):** Оцінка роботи програми на різних платформах, пристроях, браузерах та операційних системах.

7. За ознакою позитивності сценаріїв

- **Позитивне тестування (positive testing):** Перевірка, що система працює згідно з очікуваннями при коректних даних та умовах.
- **Негативне тестування (negative testing):** Перевірка поведінки системи при некоректних або неочікуваних вхідних даних, що допомагає виявити помилки або недоліки в обробці виключних ситуацій.

Індивідуальне завдання

1. Перелічити і описати не менше 10 видів тестування, про які ви знаєте. Кожен з видів тестування необхідно описувати своїми словами з поясненням. Важливо описати не тільки види тестування, а ще й вказати критерії класифікації, до яких відноситься той чи інший вид тестування (наприклад критерій по ступеню автоматизації: автоматизоване, напівавтоматизоване, ручне тестування). Опис видів тестування додати у звіті.

2. Створити схему з класифікацією видів тестування, що були описані згідно завдання п. 1, та застосувати кожен з видів до одного з предметів за відповідним варіантом (*варіант обирається згідно номера студента в журналі*).

Варіанти предметів для завдання2.

1. Ліфт	10. Холодильник	17. Електронний годинник
2. Кружка	11. Тарілка	18. Навушники
3. Стілець	12. Виделка	19. Картина
4. Стіл	13. Ноутбук	20. Зубна щітка
5. Вікно	14. Принтер	21. Книга
6. Підвіконня	15. Комп’ютерна миша	22. Аркуш паперу
7. Праска	16. Телевізор	23. Шкарпетки
8. Кавоварка		24. Шапка
9. Мікрохвильова піч		

Тести необхідно вигадувати самостійно, достатньо одного тесту для кожного виду тестування. Загалом має бути не менше 10 видів тестування та не менше 10 тестів для предмету.

Рекомендовані програми для оформлення схем: MS Word, Visio, XMind. Схему необхідно вставити у звіт. Приклади схем наведено на рисунках 4.1 та 4.2.

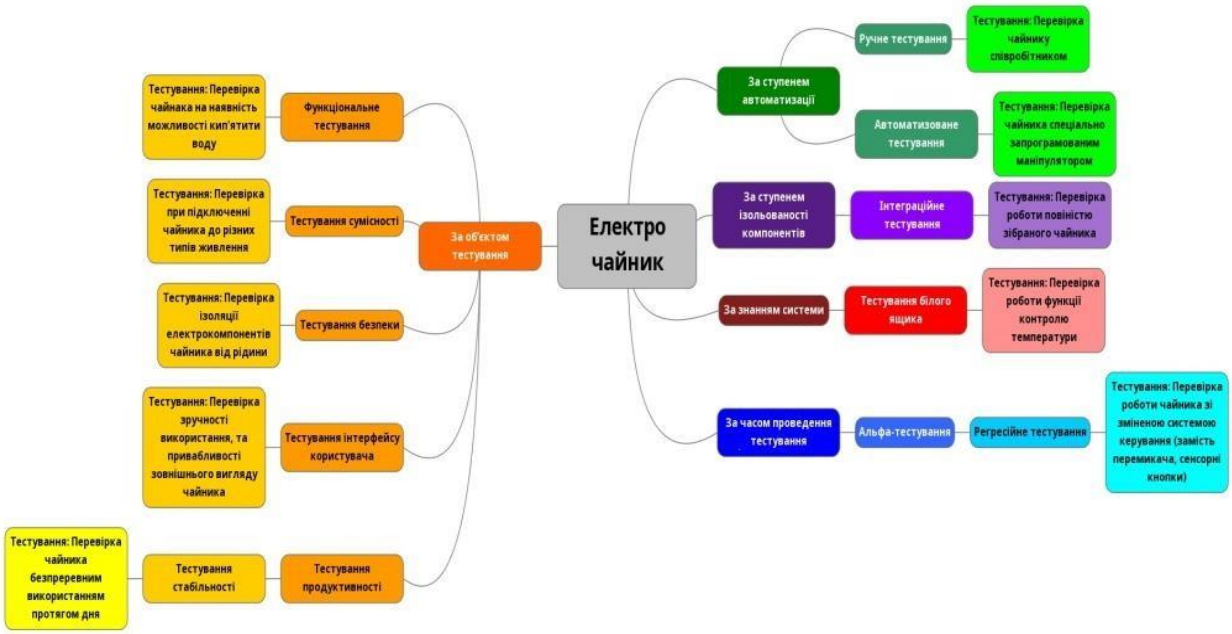

Рис. 4.1 Схема з класифікацією видів тестування електрочайника.

Рис. 4.2 – Схема з класифікацією видів тестування електромобіля.

Зміст звіту

1. Тема та мета роботи.
2. Завдання до роботи.
3. Виконати завдання до лабораторної роботи.
4. Висновки, що відображують результати виконання роботи та їх критичний аналіз.
5. Перелік використаних джерел.

Практична робота №5
Створення баг-репортів

Мета лабораторної роботи: отримання базових теоретичних та практичних навичок в розпізнаванні етапів тестування та створенні баг-репортів

Короткі теоретичні відомості

Баг **полягає в основній ідеї тестування** та характеризується як *невідповідність фактичної роботи програмного забезпечення очікуваному результату*, тобто технічному завданню чи специфікації від замовника. Перевірка на баги відбувається не тільки безпосередньо вже на етапі тестування готового продукту, а й на етапі аналізу техзавдання. Під тестування може потрапляти як інтерфейс у зручності користування, так і точність дизайну продукту, або функціональна складова бекенду програми. **Кожна помилка системи має бути задокументована.** І звіт, в якому вказані всі баги, називається баг-репорт. У ньому вказується детальний опис помилки, що сталася на етапі тестування та всі дії, що до неї призвели.

Кожен баг-репорт складається з кількох важливих атрибутів, які обов'язково вказуються у ньому:

- ***Саммарі*** (стислий опис помилки в одному рядку) - будується за принципом: що не працює? де не працює? коли? чи за яких умов не працює? Наприклад,- "Не здійснюється оплата, після натискання на кнопку "Купити" на сторінці купівлі товару".
- ***Опис помилки*** *(кроки відтворення помилки),* має бути коротким і ясним програмісту. Тут вказуються всі кроки, тобто умови, виникнення бага, фактичний результат та очікуваний результат роботи програми.
- ***Проєкт*** - вказується найменування програми, сайту чи іншої програми, до якої належить наш баг, для подальшого відстеження усунення цього дефекту.
- ***Компонент*** - під ним слід розуміти частину програми, або ділянку розробки, в якій було допущено баг. Якщо у різних частинах продукту є схожі ділянки коду, рекомендується перевірити один баг на кількох ділянках розробки. Наприклад, каталог інтернет-магазину передбачає наявність фільтрів для деталізації пошуку певного товару, але також на різних категоріях товару є ймовірність ділянки розробки коду, що повторюється, для фільтрів, які можуть бути просто скопійовані розробником. Якщо є баг в одній частині, він неодмінно буде і в іншій ділянці.
- ***Серйозність*** - тобто критичність помилки, або наскільки дана помилка впливає на працездатність нашого програмного забезпечення. Найчастіше це - несуттєвий (Minor), значний (Major), критичний (Critical) чи блокуючий (Blocker).
- ***Пріоритет*** – послідовність виправлення багів, виходячи з попередньої атрибуту бага. Зазвичай це високий (High), середній (Medium) і низький (Low).

Важливо відзначити, що пріоритет тестувальник не проставляє - цим займається менеджер або той, хто займається розподілом завдань на програмістів.

- ***Статус*** - це фіксація етапу усунення бага у його життєвому циклі. Наприклад, відкритий (Open), закритий (Closed), новий (New) або перевідкритий (Reopen).
- ***Версія білда***, тобто версія нашого продукту, в якій припущена помилка. Іноді цей атрибут може бути не вказаний, якщо немає версії продукту або бага відтворюється у всіх версіях програмного забезпечення.
- ***Автор*** - той учасник команди, який вносить інформацію про баг.
- ***Оточення*** - під це поняття може потрапляти ОС, розряд системи, браузер, у якому було виявлено помилку.

Найважливішими атрибутами кожної помилки в баг-репорті є **серйозність і пріоритет**. І якщо рівень серйозності вказує тестувальник, то ***за пріоритетність їх усунення відповідає PO (Product Owner) з боку замовника або проджект менеджер (PM)***. Оскільки є незначні помилки, які не ламають додаток, наприклад, найменування інтернет-магазину, в якому допущена помилка, але їх усунення першочергово важливе для престижу компанії замовника.

Класифікації ступеня серйозності багів та його вплив на працездатність програмного забезпечення.

- ***Blocker*** – та помилка, яка призводить до непрацездатності нашої програми. Тобто її подальше використання користувачем стає неможливим.
- ***Critical*** – є кілька варіантів: відхилення від логіки роботи програмного забезпечення, задуманого замовником; неробочий функціонал окремих елементів ПЗ (наприклад, відсутність переходу в кошик після натискання кнопки "Купити" на сайті інтернет-магазину); не збереження даних сеансу або користувача.
- ***Major*** - є схожість з критичними помилками, але в даному випадку програма працює, але не відповідає запитам замовника.
- ***Minor*** – незначний дефект, що не порушує працездатність ПЗ. Наприклад, помилка в дизайні програми.
- ***Trivial*** - найменш незначний баг програмного забезпечення, наприклад, текстова друкарська помилка. Вона ніяк не впливає на працездатність, але не відповідає бажаному результату технічного завдання.

Пріоритети відповідають за порядок виправлення багів на проєкті. Існує три найпоширеніші пріоритети: ***High, Medium*** **і** ***Low*** (іноді їх можуть класифікувати цифрами за тією ж логікою). Баг High повинен виправитися в першу чергу, зазвичай цей пріоритет поширюється на Blocker і Critical ступені серйозності, які можуть зашкодити програмному забезпеченню. Відповідно, після виправляються Medium і Low пріоритети, які критично не впливають на працездатність функцій софту. Для правильної назви заголовка бага в атрибуті опису можна використовувати золоте правило **"що?", "де?" і "коли?"**. Що - що не працює коректно, Де – місце, в якому виявлено помилку, коли – за яких

умов. Чим коректнішим і зрозумілішим буде опис бага, тим швидше він буде усунений. Оскільки, якщо баг буде не відтворюваний, то найімовірніше програміст відхилить цю помилку і вона не буде виправлена. **Усі баг-репорти вносяться до баг-трекінгових систем.** По суті, багтрекінгові системи створені не тільки для фіксації багів. В них також вносяться і User-стори, технічні вимоги від PM реалізації у продукті та інша документація. Найпоширенішими баг-трекінговими системами є Jira (найбільш використовувана), Asana, Redmine, Trello и Bugzilla.

Після початкових усунень багів, розроблений **продукт тестується повторно**, окремо на відтворення бага, і окремо за функціоналом, який міг бути порушений під час виправлення помилок. Потрібно розділяти повторне тестування бага і повторне тестування суміжного компонента, щоб не дублювати фіксацію однієї помилки. Перед написанням баг-репорту ***обов'язково відтворіть помилку кілька разів***. Оскільки вона могла виникнути у зв'язку з нестабільною роботою мережі, або через неполадки ПК та інші причини, не пов'язані з працездатністю самого програмного забезпечення. Якщо виникнення бага повторюється, необхідно відразу зафіксувати помилку та повідомити про неї програміста. Також краще заздалегідь перевірити помилку з різних браузерів та з мобільної версії, щоб максимально точно локалізувати місця, де щось не працює. Дуже часто, можна спостерігати помилки тільки в браузері Chrome, у свою чергу, як у Firefox все буде працювати коректно. І навпаки. Перед надсиланням баг-репорту програмістам необхідно перечитати його кілька разів, перевірити на орфографічні помилки, ще раз оцінити стисліть і ясність написаного звіту. Баг-репорт є одним із найважливіших документів тестувальника, він показує вашу грамотність та кваліфікованість.

Заповнення баг-репортів.

Поля:

• Поле "Summary" (короткий опис дефекту) потрібно описувати за принципом "Що? Де? Коли

• У темі, описі та результатах повинна дотримуватися послідовність "Що? Де? Коли?". Відповідати на запитання необхідно саме в такій послідовності.

• Тему, опис та результати необхідно описувати у теперішньому часі, відповідаючи на питання: "Що відбувається?".

• Допускається формулювання теми, опису та результатів в баг-репортах за принципом "Що?, Де?" без вказівки "Коли?", в разі, якщо баг відображається на сторінці завжди без будьяких додаткових дій/подій/умов (наприклад, в багах верстки).

• Тема бага повинна повністю вміщатись в рядку "Summary".

Поле "Description" потрібно описувати одним реченням за принципом "Що? Де? Коли?" згідно "Summary". Можна додати 1 додаткове речення, якщо це є важливою інформацією і не вміщується в "Summary" через обмеження за кількістю символів. Кроки;

• В першому кроці необхідно вказати посилання на головний домен, а в наступних кроках описувати послідовність дій.

• Посилання необхідно вказувати тільки в першому кроці. (Див. детальний опис в статті.

• У кроках необхідно відповідати на запитання: "Що необхідно зробити?", без звернень до третьої особи. Наприклад: Відкрити ..., Натиснути ..., Звернути увагу ...

• У кроках необхідно коротко писати, що зробити, куди натискати, не уточнюючи назви сторінки, поп-ап вікна у кожному кроці.

• У кроках необхідно вказувати які дані вводяться в поле (валідні або невалідні). У разі невалідних повинні бути наведені приклади (наприклад: "Ввести спецсимволи в поле" Name "(*, @ ,, /, <,>)").

• Необхідно скоротити кількість кроків. Як правило, в багрепорті описується не більше 7-8 кроків. Частина інформації можна винести у передумови.

• В останньому кроці не потрібно детально описувати помилку і дублювати фактичний результат. Потрібно акцентувати увагу на сам елемент з дефектом (Наприклад: Pay attention to the ..., Звернути увагу на ...).

• Кілька кроків варто обєднати в один (наприклад, "Заповнити обовязкові поля форми валідними даними"). Як правило, в баг-репорті описується не більше 7-8 кроків.

• Не варто об’єднувати багато кроків в один (наприклад: Click the "Login" button and fill the "Facebook Account" form with valid data and then click the "OK" button). Необхідно розділити складовий крок на кілька окремих кроків.

• У кроках не зазначено інформацію про роль користувача або його авторизаційні дані.

Результати :

Очікуваний результат слід вказувати після фактичного результату

• Результати необхідно описувати згідно теми за принципом "Що? Де? Коли?", відповідаючи по порядку на ці питання.

• Результати не потрібно нумерувати та не варто писати кілька результатів в одному дефекті.

• Результати в поточному баг-трекері необхідно описувати відразу після кроків в блоці "Steps To Reproduce".

• Краще уникати формулювань очікуваного і фактичного результатів, які відрізняються лише негативною формою. Очікуваний результат необхідно детально описати, недостатньо просто додати заперечення в одне з речень.

Скріншот/відео: Розширення скріншота має бути: ".png", ".bmp", ".jpeg".

На скріншоті потрібно відобразити курсор (вказівник миші).

На скріншоті достатньо одного прямокутника і однієї стрілки (прямокутником потрібно виділяти місце з помилкою, а не весь блок).

• На скріншоті/відео не повинна відображатися панель закладок браузера або інша зайва інформація (задній фон робочого столу, вікно програвача відеофайлів, сторонні повідомлення та ін.).

• На скріншоті/відео необхідно показати вікно браузера з адресним рядком.

На скріншоті необхідно додавати червону стрілку і прямокутник, використовуючи вбудовані засоби графічного редактора (наприклад, Snagit).

• На скріншоті проблемне місце потрібно виділити червоним прямокутником і вказати на нього червоною стрілкою.

• На скріншоті червоним прямокутником і червоною стрілкою виділяється проблемне місце, а зеленим прямокутником і зеленою стрілкою - правильний приклад або макет.

• Необхідно прикріпити посилання на відео, яке демонструє описаний дефект на додаток до скріншоту. Скріншот видаляти не потрібно.

Відео необхідно завантажувати формату ".avi", ".mp4", ".mkv".

Оформлення Інше:

• При тестуванні сайтів на десктопних пристроях в поле "OS" пишеться назва операційної системи (наприклад, Windows), в поле "OS Version" - версія операційної системи (наприклад, 10x64), а в полі "Platform" - назва браузера і його версія.

• Необхідно вказати браузер і його версію в полі "Platform". Якщо баг повторюється в різних браузерах, їх потрібно вказати в розділі "Additional information".

• Необхідно вказати версію браузера, так як поява багатьох дефектів залежить саме від версії браузера.

• Пропущені деякі атрибути в баг-репорті (тема, опис, кроки, результати, скріншот/відео, тестове оточення). (Див. детальний опис в

• У темі та результатах потрібно вказати, яка саме помилка відображається без застосування слів "некоректно/коректно", "невірно/вірно", "правильно/неправильно", "incorrect/correct", "wrong/right". Наприклад: Зображення товару розтягується (Що?) у формі "Надіслати другу" (Де?)

• Необхідно перефразувати тему або результати, вказавши яка саме помилка відображається. Не варто писати "нічого не відбувається", "не працює", "ламається верстка", "nothing happens", "the layout is broken". Наприклад: Вікно завантаження зображення не з’являється на сторінці редагування профілю ... The menu items are shifted to the left.

• В баг-репорт не варто використовувати слова на кшталт "можна/не можна", "можливо/неможливо", "possible/impossible", "can/can not". Необхідно точно описувати суть бага, вказуючи, в чому саме невідповідність.

• Особисті займенники ("ви", "ми", "я"), а також слова "користувач/user" краще не використовувати, перефразувавши опис баг-репорта від третьої особи, акцентуючи увагу на об’єкт дії: "сайт", "кнопка "," посилання " та ін.

Тему та результати слід описувати повними реченнями з підметом і присудком.

• Поряд з назвою елемента потрібно писати його тип (кнопка, посилання, поле і т.д.), так як на сайті може бути декілька елементів з однаковими назвами.

• Не потрібно перекладати назви елементів сайту на мову оформлення звіту.

Назви кнопок, полів, посилань, повідомлень необхідно вказувати саме так, як вони відображені на сайті.

• Коли баг відображається після скоєної дії правильно писати "після" замість "при". Наприклад, "після натискання ...", "після наведення ...".

• Не потрібно створювати дублікати з описом одного і того ж дефекту, який відтворюється на різних сторінках або для різних елементів. У розділі "Additional Information" вказується перелік сторінок та елементів з однотипними багами.

• Тему, опис, кроки, назви результатів необхідно писати з великої літери.

• Необхідно використовувати лапки в назвах елементів сайту. Приклад наведено на рис. 5.1.

	A	B	C	D	E	F
1	номер ID	Проєкт	Категорія	Рівень доступу	Дата реєстрації	Дата зміни
2	1					
3	Тестувальник		Пріорітет		Серйозність	
4	Відтворюваність		Статус		Резолюція	
5	Платформа		Операційна система		Версія ОС	
6	Тема					
7	Опис					
8	Кроки для відстворенн					
9	Фактичний результат					
10	Очікуваний результат					
11	Додаткова інформація					
12	Прикріплені файли					

Рис. 5.1. Створення баг-репорту

Завдання до практичної роботи

1. Відкрити сайт, наприклад: http://prestashop.qatestlab.com.ua/en/.

2. Знайти 3 баги, та оформити баг-репорти, див. рис 5.1 (для створення баг-репорту можна використовувати файл Excel).

3. Додати скріншоти оформлених баг-репортів до звіту.

4. Оформити звіт.

Зміст звіту

1. Тема та мета роботи.

2. Завдання до роботи.

3. Короткі теоретичні відомості.

4. Виконати завдання до практичної роботи.

5. Висновки, що відображують результати виконання роботи та їх критичний аналіз.

6. Перелік використаних джерел.

Контрольні запитання

1. Визначити основні властивості та відмінності встановлених браузерів: Mozilla Firefox, Opera, Google Chrome, Edge. Обґрунтувати розповсюдженість використання встановлених браузерів.

2. Для чого призначені та які існують системи відслідковування помилок?

3. Що повинен містити якісний опис помилки?

4. Що відносять до атрибутів дефекту?

Практична робота №6
Веб-тестування, чек-листи та кросбраузерне тестування

Короткі теоретичні відомості

Чек-лист – це зручний і структурований інструмент, який допомагає тестувальникам у проведенні перевірки програмного забезпечення. Він являє собою список завдань, кроків і критеріїв, які необхідно виконати для ретельної перевірки функціональності або інших аспектів ПЗ.

Використання чек-листа в процесі тестування ПЗ надає кілька переваг. Ось деякі з них:

- **Нічого не пропустити!** Уявіть, що у вас є величезний застосунок із безліччю функцій і можливостей. Без чек-листа, ви можете забути перевірити якісь важливі аспекти і упустити помилки. Чек-лист допомагає не загубитися і бути впевненими, що покриті всі основні перевірки. Він нагадує про кожен крок, який потрібно виконати, щоб нічого не пропустити.
- **Структурованість і організація.** У роботі тестувальника дуже важливо мати структуру й організацію. Чек-лист це і надає: допомагає розбити процес на більш дрібні завдання і слідувати за порядком. Це особливо корисно, коли у є великий обсяг роботи або обмежений час. Чек-лист дає нам чітке уявлення про те, що потрібно робити і в якій послідовності.
- **Уніфікація та консистентність.** У команді QA може працювати кілька тестувальників, і кожен із них може мати свої вподобання та методи роботи. Чек-лист допомагає уніфікувати процес і встановити єдині стандарти для всіх. Коли кожен дотримується одного й того самого чек-листа, ми можемо бути впевнені, що всі основні перевірки будуть виконані. Це підвищує консистентність і якість роботи команди.

З яких ключових моментів складається чек-лист тестування? Їх кілька і вони допомагають організувати та впорядкувати процес перевірки. Ось деякі з них:

- **Завдання та функціональність.** Чек-лист має містити список завдань і функціональності, які слід перевірити. Це може бути щось на кшталт “перевірити, що кнопка ‘Відправити’ надсилає дані форми” або “переконатися, що функція пошуку знаходить усі результати, які повинна”. Простіше кажучи, це список дій і перевірок, які слід виконати, щоб переконатися, що все працює належним чином.
- **Критерії оцінки** або очікувані результати, які порівнюються з фактичними результатами перевірок. Наприклад, якщо перевіряється функція пошуку, критерієм оцінки може бути те, що за правильного запиту мають бути видані всі відповідні результати. Якщо це відбувається, відзначається завдання як виконане успішно. Якщо ні, значить є недолік, якому потрібно приділити увагу.
- **Пріоритети.** У чек-листі також можна вказати пріоритети для кожного

завдання. Це допомагає визначити, які з них слід виконати першими, щоб зосередитися на найважливіших і критичних аспектах ПЗ. Пам'ятайте, може бути обмежений час або ресурси, тому вміння пріоритизувати – ключова навичка!

При проходженні чек-листів тестувальник зазначає статус навпроти кожного тестованого пункту. Можливі такі варіанти статусів:

•«Passed» - перевірка пройдена успішно, багів не знайдено;

•«Failed» - знайдений один або більше багів;

• «Blocked» - неможливо перевірити, тому що один з багів блокує поточну перевірку;

• «In Progress» - поточний пункт, над яким працює тестувальник;

•«Not run» - ще не перевірено;

•«Skipped» - не буде перевірятися з будь-якої причини.

Зразок чек-листа наведено на рис. 6.1.

	A	B	C	D	E	F	G	H
1		*Сайт http:// . . .*	*Браузер версія*	*Браузер версія*	*Браузер версія*	*Примітки/Посилання на баг-репорт*	Failed	Failed (Для "Failed" в коментарях/примітках вказати посилання на баг-репорти. Баг-репорт оформити у баг-трекері та зберігти на гугл-диску у вигляді скріншоту)
2							Passed	Passed (пункт пройдений, багів не знайдено)
3							Skipped	Skipped (якщо немає можливості пройти пункт)
4	1	**Наявність елементів сторінок**						
5	a	Перевірити коректне відображення сторінок згідно файлів дизайну	Skipped	Skipped	Skipped			
6	b	Перевірити наявність логотипу	Failed	Failed	Passed			
7	c	Перевірити наявність головного меню	Passed	not run	not run			
8	d	Перевірити наявність випадаючого меню з підкатегоріями	not run	not run	not run			
9	e	Перевірити наявність підвалу сайту	not run	not run	not run			
10	f	Перевірити наявність рядка пошуку	not run	not run	not run			
11	g	Перевірити наявність повідомлень при додаванні товару до кошика/списку бажаних	not run	not run	not run			
12	2	**Відображення елементів**						
13	a	Перевірити стиль та коректне відображення шрифтів тексту	not run	not run	not run			
14	b	Перевірити коректне відображення кнопок, блоків меню та ін.	not run	not run	not run			
15	c	Перевірити відображення кольорової гами усіх елементів	not run	not run	not run			
16	d	Перевірити коректне вирівнювання елементів (кнопки, посилання, заголовки)	not run	not run	not run			
17	e	Перевірити коректне розміщення банерів	not run	not run	not run			
18	3	**Активні елементи**						
19	a	Перевірити наявність підказок для активних елементів, призначення яких не є очевидним	not run	not run	not run			
20	b	Перевірити зміну курсору на pointer при наведенні на клікабельний елемент	not run	not run	not run			
21	c	Перевірити реагування активних елементів на наведення курсора (підсвічування)	not run	not run	not run			
22	4	**Зміст сторінок**						
23	a	Перевірити наявність посилань на соц. мережі в підвалі сайту	not run	not run	not run			
24	b	Перевірити відповідність посилання на соц. мережі сторінці сайта	not run	not run	not run			
25	c	Перевірити наявність повідомлення про авторські права в підвалі сайту	not run	not run	not run			
26	d	Перевірити орфографію/граматику контенту сайту	not run	not run	not run			
27	e	Перевірити відповідність характеристик товару його назві	not run	not run	not run			
28	f	Перевірити адаптацію елементів сайту для роздільної здатності 1280x800	not run	not run	not run			
29	g	Перевірити коректне відображення валюти	not run	not run	not run			
30	h	Перевірити відсутність горизонтальної прокрутки після збільшення масштабу сторінки до 150%	not run	not run	not run			

Рис. 6.1 – Чек-лист

Вимоги до розмітки (верстки):

• чи немає помітних оку невідповідностей: поламані блоки, не стикування кольору, некоректне відображення тексту навколо зображень;

• якщо дизайн згідно ТЗ розрахований на певну ширину, то, незалежно від браузера, не повинна з'являтися горизонтальна прокрутка;

• під час зменшення розміру вікна менше мінімального згідно ТЗ не повинно нічого ламатися, фони не повинні «розпливатися»;

• сайт повинен виглядати однаково у всіх стандартних розширеннях екрану (1024x600, 1024x768, 1152x864, 1280x800, 1280x1024, 1440x900, 1680x1050, 1920x1080): не повинно нічого ламатися, не повинні різко обриватися фони при великих розширеннях;

• при відключених зображеннях, написи (особливо логотип та головне

меню сайту) повинні залишатися читабельними, на всіх інформаційних картинках повинні бути підписані акуратним, невеликим, сірим шрифтом (відключення картинок: Chrome → Settings → Search settings → JavaScript → Site settings→ Images → вибрати опцію «Don`t allow sites to show images»; Firefox→ about:config → permissions.default.image → 2);

• працездатність при вимкненому JavaScript. Увесь критично важливий функціонал сайту повинен бути доступний без JS (відключення JavaScript: Chrome → Settings → Search settings → JavaScript → JavaScript (Content) → Don`t allow sites to use JavaScript; Firefox→ about:config → Accept the Risk and Continue → javascript.enabled → Toggle the «javascript.enabled» preference).

Перевірка однотипності/симетричності в інтерфейсі:

• однотипність/симетричність використання відступів;

• ширина колонок та контентного поля;

• позиціонування елементів верхнього та нижнього колонтитулів на всіх сторінках;

• позиціонування банерів, правого та лівого блоку;

• розмір текстів;

• відступи між абзацами;

• відстані між рядками;

• • елементи управління (кнопки, чекбокси, випадаючі списки);

• розмір/колір/тип шрифту;

• наявність стилів оформлення заголовків першого, другого та більше рівнів;

• коректне відображення тексту навколо зображень;

• напрямки тіней у всіх елементів управління;

• назви однакових елементів у різних місцях;

• чи мають елементи, що призначені для натискання, вказівник «pointer»; елементи, які можна перетягувати – вказівник «move» або «crosshair»; неактивні/недоступні елементи – вказівник «default» (рис. 6.2).

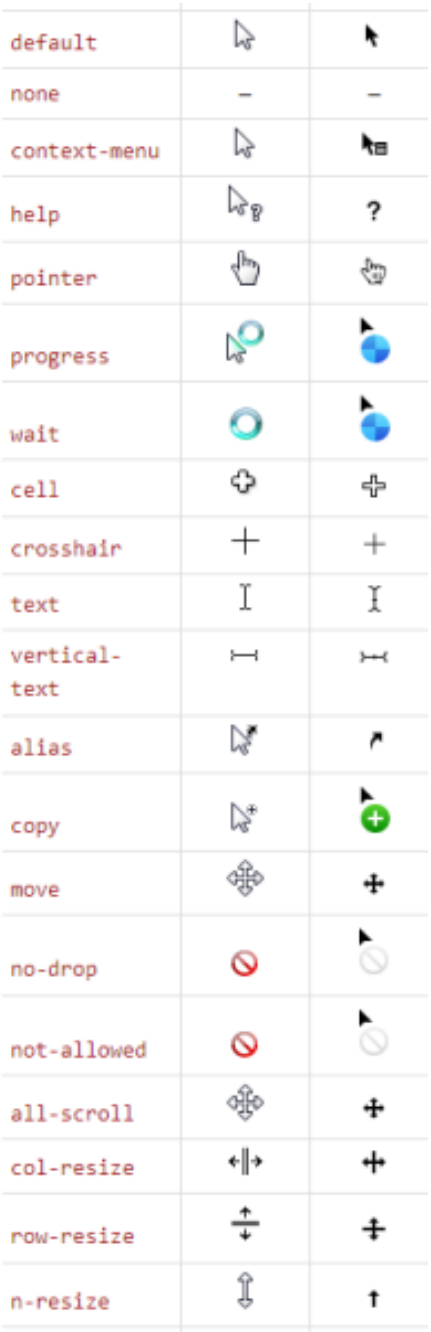

Рис. 6.2. Форма вказівника при наведенні на відповідний елемент

Кросбраузерність – властивість сайту відображатися та працювати у всіх браузерах ідентично. Під ідентичністю розуміється відсутність розвалів верстки та здатність відображати матеріал з однаковим ступенем читабельності. Поняття «кросбраузерність» дуже часто плутають з попіксельною відповідністю, що насправді є різними поняттями. Так як веб-технології весь час розвиваються, прийнятну кросбраузерність можна забезпечити тільки для останніх версій різних браузерів. На практиці зазвичай обмежуються тільки найпопулярнішими браузерами, що суттєво може скоротити час на розробку сайту.

Правила кросбраузерності:

• під час застосування будь-яких змін в дизайні сайту (чи то новий інформер, рекламний блок, новий шаблон або ділянка шаблонів) – необхідно

звертати увагу, який має вигляд дизайн в інших браузерах;

• потрібно тестувати сайти самостійно за допомогою встановлених на комп'ютері браузерів, інколи використовуючи спеціальні онлайн сервіси;

• необхідно здійснювати тестування кросбраузерності на всіх доступних пристроях (комп'ютер, ноутбук, планшет та ін.);

• завжди повинен бути встановлений на комп'ютері комплект найбільш популярних браузерів, які не вичерпують ресурси жорсткого диска, але якщо буде потрібно перевірити кросбраузерність сайту – вебмастер запускає їх та проводить необхідний аналіз;

• різні версії одного й того ж браузера будуть відображати сайт теж по-різному і це потрібно враховувати, але намагатися використовувати найостанніші версії.

Основні проблеми кросбраузерності сайту лежать на витоках його створення – під час створення шаблону. Існує багато готових шаблонів, з яких можна вибрати той єдиний, який буде однаково (або з незначними змінами) відображатися у всіх браузерах, а потім доопрацювати його під свій проєкт, але з постійним орієнтиром на різні браузери.

Інструменти тестування кросбраузерності верстки:

1. ***Browsershots***, ймовірно, володіє найширшим набором браузерів серед безкоштовних інструментів тестування, включає в себе браузери, що працюють в Linux, Windows та BSD. Серед них є такі, про які взагалі ніколи не чули (наприклад, Galeon, Iceape, Kazehakase або Epiphany). Browsershots дозволяє тестувати як в останніх версіях кожного браузера, так і в застарілих версіях. Хоча Browsershots дозволяє тестувати у величезній кількості браузерів, варто пам'ятати, чим більший набір браузерів для тестування, тим довше доведеться чекати результату. Так що варто зупинитися на основних браузерах.

2. ***Browser Sandbox*** – інструмент, який буде корисним тільки користувачам Windows. Він підтримує такі браузери, як Firefox, Chrome, ChromiumCanary, Firefox Mobile, Safari, Opera та FirefoxNightly. В безкоштовному варіанті існує можливість тестування тільки в останній версії певного браузера. Для тестування в старих версіях необхідно користуватися платною версією.

3. ***Microsoft Edge*** - це справжня платформа для тестування сайтів в IE. В ньому можна отримати скріншот сайту в багатьох браузерах, в тому числі і мобільних, але на основі платформи Microsoft Edge можна розгорнути віртуальну машину тільки для тестування в IE7 і новіших.

4. ***CrossBrowserTesting*** - сервіс, який використовує реальні пристрої для тестування сайту. Сервіс платний, але надає можливість тестувати більш як в 900 браузерах і приблизно в 40 операційних системах. Ще одна його особливість полягає в режимі live testing, в якому можна тестувати сайт в реальному оточенні, перевіряючи дієздатність AJAX, HTML-форм, JavaScript, Flash тощо.

5. ***Sauce Labs*** (безкоштовна та комерційна версії) - надає доступ до безлічі браузерів в різних ОС та встановлює з'єднання браузера з налагодженою віртуальною машиною. Також записується відео всієї сесії тестування. Sauce

надає 200 хвилин безкоштовного тестування на місяць і дозволяє створювати тести автоматизованого тестування в браузерах (використовується Selenium).

6. ***Browsera*** (безкоштовна та комерційна версії) - забезпечує автоматизацію тестування сумісності. Він автоматично визначає відмінності в відображенні сторінок браузерами, тим самим спрощуючи процес тестування. Також визначаються помилки JavaScript, а в комерційній версії дозволяє тестувати сторінки за передплатою та сторінки, що вимагають авторизації. Також можна протестувати динамічні сторінки. Безкоштовна версія включає в себе достатньо обмежене число браузерів та низький дозвіл. Існують різні комерційні версії, які підтримують більшу кількість браузерів, забезпечують високий дозвіл та дозволяють тестувати «закриті» сторінки.

7. ***BrowserStack*** - ще один з найвідоміших сервісів для кросбраузерного тестування. Він також надає реальні пристрої для тестування і підтримує більш як 700 браузерів. Існує можливість локального тестування та швидкого отримання скріншотів на різних розширеннях екранів від 800-600 до 2048-1536. Сервіс платний, але для деяких open source проєктів пропонуються безкоштовні послуги.

Кросбраузерне тестування верстки–це процес перевірки того, як вебсайт або вебзастосунок відображається та функціонує в різних браузерах, їхніх версіях, операційних системах і на різних пристроях. Для цього використовують спеціальні інструменти, які допомагають автоматизувати або полегшити перевірку.

Популярні інструменти для тестування кросбраузерності

1. BrowserStack

Хмарний сервіс для тестування на реальних пристроях та віртуальних машинах. Підтримує різні версії браузерів, операційних систем та мобільних пристроїв. Дозволяє тестувати інтерактивність сайту, а не тільки його зовнішній вигляд. Має можливості автоматизації тестування через Selenium, Cypress, Playwright.

2. Sauce Labs

Аналог BrowserStack із широкою підтримкою середовищ. Дозволяє тестувати веб- та мобільні застосунки в різних комбінаціях ОС та браузерів. Підтримує автоматизоване тестування з Selenium, Appium, Cypress тощо.

3. LambdaTest

Хмарний сервіс для тестування вебдодатків у понад 3000 середовищах. Можливість інтеграції з CI/CD (Jenkins, GitHub Actions). Підтримка тестування локальних серверів через SSH-тунель.

4. CrossBrowserTesting (від SmartBear)

Дозволяє тестувати на реальних пристроях і віртуальних середовищах. Має функцію автоматичних скріншотів для порівняння рендерингу верстки. Підтримує інтерактивне тестування, автоматизоване тестування з Selenium.

5. Comparium

Генерує скріншоти вебсторінок у різних браузерах і на різних пристроях. Підтримує перевірку відтворення шрифтів, кольорів, елементів інтерфейсу.

6. Blisk

Окремий браузер, розроблений для тестування кросбраузерності та адаптивності. Має функцію одночасного перегляду сайту на мобільному та десктопі. Автоматично оновлює сторінку при зміні коду.

Інструменти для візуального порівняння верстки

Якщо потрібно перевірити відповідність верстки між різними браузерами, можна використовувати:

- **Percy** – робить скріншоти та виконує візуальне порівняння.
- **Applitools Eyes** – використовує AI для пошуку візуальних відмінностей.
- **Pixel Perfect** – плагін для Chrome, який дозволяє накладати зображення макета на сайт.

Автоматизоване тестування кросбраузерності

Якщо потрібно регулярно перевіряти сайт на різних браузерах, можна використовувати:

- **Selenium Grid** – дозволяє запускати тести в різних браузерах одночасно.
- **Cypress + Percy** – автоматичне тестування + візуальна перевірка змін.
- **Playwright** – підтримує тестування в Chromium, Firefox і WebKit.

Завдання до роботи

1 Завантажити шаблон чек-листа з навчальної системи Moodle. Вказати прізвище та ім’я власника у назві контрольного списку.

2 Провести тестування верстки сайту http://prestashop.qatestlab.com.ua/en/, перевіривши всі пункти контрольного списку на вкладці «Верстка» у 3 браузерах (Firefox, Google Chrome, Edge, Opera або інший) в останній або передостанній версії.

3 Результат перевірки відзначити «Passed/Failed». Для «Failed» вказати в примітці або нотатках посилання на баг-репорти (2 шт.), якщо багів буде більше 2, то додати в примітках тему багу за принципом «Що? Де? Коли?».

4 Створити 2 баги-репорти для багів, знайдених під час проведення тестування верстки, оформити їх та додати до звіту.

5 Скриншот пройденого чек-листа додати до звіту.

6 Оформити звіт з лабораторної роботи та надати його на перевірку.

Зміст звіту

1 Тема та мета роботи.

2 Завдання до роботи.

3 Короткі теоретичні відомості.

4 Хід роботи з виконання завдання до лабораторної роботи.

5 Висновки, що відображують результати виконання роботи та їх критичний аналіз.

6 Перелік використаних джерел.

Контрольні запитання

1. Що відносять до дефектів розмітки (верстки)? Навести приклади та

обґрунтувати відповідь.

2. За яким принципом були додані нові пункти до чек-листа? Обґрунтувати важливість нових пунктів при тестуванні сайту http://prestashop.qatestlab.com.ua/en/.

3. Для чого в примітці до пункту чек-листа вказується посилання на звіти у системі відслідковування помилок? Обґрунтувати необхідність посилання на звіт в чек-листі.

4. Обґрунтувати необхідності створення пунктів та підпунктів в чек-листі.

5. Які існують переваги використання чек-листів порівняно з іншою подібною документацією для проведення тестування? Навести приклади та обґрунтувати відповідь.

6. Обґрунтувати необхідність та важливість кросбраузерного тестування.

7. За якими чинниками визначають, в яких браузерах проводити тестування?

8. Перерахувати існуючі розширення у різних браузерах для тестування вебінтерфейсів, вказати доцільність та ефективність застосування.

9. Для чого здійснюється та як впливає на роботу очистка історії (включаючи cookies, cache та ін.) у браузері? Навести приклад роботи браузера до очистки та після.

10. Назвати програму, яка була використана для тестування кросбраузерного тестування у лабораторній роботі, обґрунтувати вибір даної програми.

Практична робота №7
Тестування зручності використання

Мета роботи: отримання базових теоретичних та практичних навичок в визначенні, застосуванні тестування зручності використання та складанні простого контрольного списку.

Короткі теоретичні відомості

Тестування зручності використання (usability testing) – метод тестування, спрямований на встановлення ступеня зручності використання, здатності до навчання користувача, зрозумілості та привабливості для користувачів розроблювального продукту в контексті заданих умов.

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

- ***продуктивність, ефективність (efficiency)*** **–** скільки часу та кроків знадобиться користувачеві для завершення основних завдань програми, наприклад, розміщення новини, реєстрації, покупка та ін.? (*менше - краще*);
- ***правильність (accuracy)*** - скільки помилок зробив користувач під час роботи з додатком? (*менше - краще*);
- ***активізація в пам'яті (recall)*** - як багато користувач пам'ятає про роботу програми після припинення роботи з нею на тривалий період часу? (*повторне виконання операцій після перерви має проходити швидше ніж у нового користувача*);
- ***емоційна реакція (emotional response)*** - як користувач себе почуває після завершення завдання - розгублений, в стані стресу? Чи порекомендує користувач систему своїм друзям? (*позитивна реакція - краще*).
- простоту використання сайту або інтерфейсу;
- ефективність використання;
- запам'ятовуваність;
- помилки, їх кількість та серйозність;
- задоволення користувача *(суб'єктивне).*

Правильне уявлення про тестування зручності використання *(usability testing)* та застосування в роботі приходить з досвідом та знаннями структури проєкту, а також вимог до проєкту. Також слід звернути увагу на те, що саме підлягає тестуванню:

- веб додаток;
- мобільний додаток;
- десктоп-додаток.

У кожного з цих видів додатків можуть бути різні цілі, використовувані ресурси (наприклад пам'ять, енергія та ін.), оточення, ступінь досвіду користувача та багато іншого.

Розглянемо приклади:

1. Вебдодаток (інтернет-магазин).
2. Спеціалізований вебдодаток для формування бухгалтерських звітів.

Інтернет-магазин:

Мета - користувачі повинні багаторазово користуватися сайтом, весь цикл повинен бути простий, містити якомога менше кроків, інтерфейс повинен бути інтуїтивно зрозумілий для будь-якого користувача.

Критично важливо перевірити простоту використання (основного функціоналу) та зрозумілість використання інтерфейсу без додаткового вивчення яких-небудь документів або правил використання.

***Веб-додаток для формування бухгалтерських звітів*:**

Мета - користувач повинен мати можливість скласти звіти з усіма можливими даними, що зберігаються в БД в різній послідовності за певний період часу. 30

Не так важлива простота використання ***як*** функціональні можливості та правильність даних, тобто скоріше всього в складній системі працюватимуть користувачі, які заздалегідь навчені всім можливим алгоритмам роботи та знають де й що натискати.

Завдання до роботи

1. Завантажити приклад чек-листа з навчальної системи Moodle.
2. Вказати прізвище та ім’я власника у назві контрольного списку.
3. Додати мінімум 3-5 пунктів до існуючих пунктів контрольного списку на вкладці «Usability», не повторюючись з вже доданими, та відмітити їх будь-яким кольором.
4. Провести тестування зручності використання сайту http://prestashop.qatestlab.com.ua за чек-листом. В чек-листі вказати назву та версію браузеру. Результат перевірки відзначити «Passed/Failed». Для «Failed» вказати в примітці або нотатках посилання на баг-репорт (2 шт.), якщо багів набагато більше 2, то додати в примітках тему багу за принципом «Що? Де?Коли?».
5. Створити 2 баги-репорти для багів, знайдених під час проведення тестування зручності використання сайту http://prestashop.qatestlab.com.ua, оформити їх та додати до звіту.
6. Скріншот доповненого та пройденого чек-листа додати до звіту.
7. Оформити звіт з лабораторної роботи та надати його на перевірку.

Зміст звіту

1. Тема та мета роботи.
2. Завдання до роботи.
3. Короткі теоретичні відомості.
4. Хід роботи з виконання завдання до лабораторної роботи.
5. Висновки, що відображують результати виконання роботи та їх критичний аналіз.
6. Перелік використаних джерел.

Контрольні запитання

1. Що відносять до дефектів зручності використання?
2. Навести приклади та обґрунтувати відповідь.
3. За яким принципом були додані нові пункти до чек-ліста? Обґрунтувати важливість нових пунктів при тестуванні сайту http://prestashop.qatestlab.com.ua/en/.
4. Якого принципу необхідно дотримуватися при поліпшенні зручності користування?
5. Які основні пункти та стовпці повинен містити контрольний список з тестування зручності використання (usability testing)?
6. Обґрунтувати важливість тестування зручності використання.

Практична робота №8
Технічне тестування

Мета роботи: отримання базових теоретичних та практичних навичок з технічного тестування, ознайомлення з функціональними можливостями Інструментів розробника та *Screaming Frog SEO Spider*.

Короткі теоретичні відомості

Технічне тестування - група видів тестування, що проводиться на завершальному етапі готовності програмного продукту (сайту) й спрямована на перевірку технічних характеристик, таких як:

- здатність сервера витримувати навантаження;
- швидкість завантаження сторінок;
- трастовість (такі показники як вік, PR);
- цілісність посилань;
- індексація сторінок, наявність в каталогах пошукових машин;
- мета-теги, HTML-теги;
- відповідність стандартам W3C і багато іншого.

Основні програми для технічного тестування:

1. **Інструменти розробника (Developer Tools)** - це потужний інструмент для налагодження коду веб-сайтів, який за замовчуванням встановлений в браузерах Firefox, Google Chrome та в інших браузерах (зазвичай доступ через натискання F12). Крім веб розробки, даний інструментарій, а також накопичені знання з HTML, CSS будуть дуже корисні тестувальникам при тестуванні веб-сайтів і створенні баг-репортів. Визначити колір, погратися зі шрифтами, виміряти піксельну позицію елементу на сторінці або протестувати адаптивність сайту для різних пристроїв - тепер все це можна зробити не встановлюючи окремих додатків.

До основних можливостей Інструментів розробника відносять:

- зручний перегляд HTML-коду сторінки: функція Inspect дозволяє точно визначити місцезнаходження тега того чи іншого елемента, переглянути всі «прив'язані» до нього властивості та стилі;
- редагування HTML та CSS безпосередньо в браузері: можна змінювати атрибути тегів та значення властивостей для того, щоб спостерігати зміни. Це зручно для тих випадків, коли потрібно шляхом експериментів знайти найбільш прийнятний варіант оформлення створеної сторінки;
- налагодження JavaScript;
- відстеження процесу завантаження сторінки. Існує можливість урізати швидкість з'єднання, щоб подивитися як буде поводити себе контент сторінки при повільному інтернеті або коли взагалі не буде інтернету;
- перегляд HTTP-заголовків звичайних та AJAX-запитів;
- редагування або видалення cookie. Вкладка Sources;
- зміна даних геолокації – це досить зручно при тестуванні різних сервісів з урахуванням карт і маршрутів. У цьому режимі можна й провести тестування

юзабіліті мобільної версії сайту. А підтримка емуляції відразу пари десятків пристроїв на адаптивність, зробить перевірку ще більш надійною.

2. **Xenu Link Sleuth** – це один з корисних інструментів (варто враховувати, що з 2010 року не оновлюється) в пошуковій оптимізації (https://home.snafu.de/tilman/xenulink.html).

До основних завдань, які виконує програма відносять:

- шукати биті (непрацюючі) посилання на заданому ресурсі: своєчасне виявлення несправних посилань дозволяє не тільки поліпшити показники сайту в пошукових системах, але й вочевидь підвищити інтерес та лояльність користувачів сайту;
- складати карту сайту: для динамічних сайтів скласти карту не складає проблеми, однак, для статичних HTML ресурсів створювати карту сайту вручну вельми довго й трудомістко. *Xenu* вирішує цю задачу за кілька хвилин в залежності від розміру сайту та швидкості інтернет-з'єднання;
- шукати сторінки з великим часом віддачі: вимірювання швидкості завантаження веб сторінок – дуже важливий процес тестування. Але заміряти вручну кожну сторінку багатосторінкового сайту занадто затратно по ресурсам. За допомогою *Xenu* можна побачити картину відгуку всіх сторінок в цілому та визначити проблемні місця;
- знаходження неунікальних заголовків: кожен заголовок на сторінці повинен бути унікальним, тоді жоден з них не буде знаходитися в додаткових результатах пошуку та фільтруватися, як дубльований контент.
- знаходження сторінок з великим рівнем вкладеності: всі сторінки на сайті по можливості повинні знаходитися не далі, ніж у двохтрьох рівнях від головної. Чим далі знаходиться сторінка, тим складніше до неї добратися як користувачам, так і пошуковим системам. Якщо знайшлися подібні сторінки, які являються достатньо важливими, але знаходяться далі, ніж в 3-х рівнях від головної, варто прийняти які-небудь заходи для поліпшення навігації;
- виявляти, які зі сторінок мають найбільшу та найменшу кількість внутрішніх посилань: перевірити внутрішню перелінковку в числовому вигляді. Які з сторінок заслужили більше уваги, а які менше (виходячи з внутрішніх посилань);
- знаходження картинок з відсутнім атрибутом «alt»: атрибут «alt» є важливим при оптимізації сайту або окремих сторінок під певні запити. Перевірити, можливо відсутня важлива інформація чи опис зображень на вашому сайті.

3. **Програма Screaming Frog SEO Spider** – це професійний десктопний інструмент, який сканує веб сайти так само, як це робить пошуковий робот (наприклад, Googlebot). Вона широко використовується SEO-фахівцями, тестувальниками та розробниками для аналізу структури сайту, виявлення технічних помилок і перевірки якості контенту (https://www.screamingfrog.co.uk/seo-spider/). В безкоштовній версії має обмеження на 500 url-записів.

Основні завдання та можливості Screaming Frog SEO Spider

1. Пошук битих (непрацюючих) посилань

- Виявляє помилки 404 (not found), 500 (server error), перенаправлення 301/302.
- Допомагає усунути проблеми з навігацією та підвищити якість індексації сайту.
- Створює звіти для команди розробників або тестувальників.

2. Створення карти сайту (XML Sitemap)

- Автоматично генерує XML Sitemap на основі сканування.
- Можна налаштувати частоту оновлення, пріоритетність, виключення окремих сторінок.
- Особливо корисно для статичних HTML-сайтів, де Sitemap не створюється динамічно.

3. Аналіз швидкості відгуку сторінок

- Фіксує час відповіді кожної сторінки (Response Time).
- Допомагає знайти повільні сторінки, що впливають на загальну продуктивність сайту.

4. Аналіз метаданих (Title, Description, H1, Canonical тощо)

- Виявляє неунікальні заголовки або мета-описи.
- Показує надто короткі, довгі або відсутні метадані.
- Це важливо для SEO та уникнення дублювання контенту.

5. Аналіз структури сайту та рівнів вкладеності

- Визначає, на якій глибині знаходиться кожна сторінка.
- Допомагає знайти важливі сторінки, сховані глибоко в структурі.
- Полегшує оптимізацію внутрішньої навігації.

6. Перевірка внутрішньої перелінковки

- Показує кількість внутрішніх і зовнішніх посилань на кожну сторінку.
- Виявляє сторінки з недостатньою увагою (які мають мало посилань з інших сторінок).
- Дозволяє оптимізувати "лінк-джус" всередині сайту.

7. Пошук зображень без атрибуту alt

- Визначає всі зображення з відсутнім або порожнім alt.
- Важливо для доступності (accessibility) та SEO-оптимізації (особливо Google Images).
- Допомагає уникнути пропущених описів у візуальному контенті.

4. **Тестування навантаження (Load Testing)** – дозволяє зрозуміти чи здатен сайт/додаток стабільно працювати з навантаженням в допустимих межах і трохи перевищуючи ці межі. При цьому тестуванні перевіряється поведінка з високим навантаженням. Навантаженням може бути, наприклад, певна кількість одночасно працюючих користувачів додатка в певний проміжок часу. Такий тип тестування дозволяє визначити час відгуку важливих бізнес-транзакцій. Спостерігаючи за базою даних, сервером додатка і мережею можна визначити слабкі місця програми.

Навантажувальне тестування найчастіше застосовується для дослідження багатокористувацьких ресурсів і різноманітних загальних систем, але інші види програмних продуктів також можуть бути протестовані цим методом.

Наприклад, можна перевірити, чи зможе графічний редактор впоратися з відкриттям зображення великого обсягу. Також можна з’ясувати, чи здатна система сформувати фінансовий звіт, проаналізувавши дані декількох місяців, або навіть років. Якісно створений навантажувальний тест забезпечить найбільш достовірними результатами.

Основною ідеєю навантажувального тестування є створення певного навантаження за допомогою однакових апаратних і програмних забезпечень, щоб відстежити індекс продуктивності продукту. Найефективнішим цей метод буде на ранніх етапах розробки, тому що такий підхід дозволить отримати оптимальні результати вимірювання показників продуктивності системи.

Основні принципи навантажувального тестування:

1. Унікальність запитів. Під час складання сценарію слід враховувати реальні статистичні дані, вимоги, очікувану поведінку системи.

2. Час відгуку системи. Керуючи певним собі числом вимірів, можна визначити до якого інтервалу часу потрапить той чи інший запит.

3. Розподіл системи залежить від часу відгуку. Кількість вузлів впливає на розкид часу відгуку системи, кожен з яких збільшує величину затримки при скануванні запитів. Цей факт варто врахувати під час складання вимог до продуктивності продукту.

4. Коректність відтворення навантажувальних профілів. Складність програмного забезпечення вимагає значних витрат часу і ресурсів на проєктування, програмування і подальшу підтримку. Розробка тестів і покриття функціоналу системи повинні бути збалансованими для отримання найбільш

Системи управління контентом (CMS – Content Management System)

Системи управління контентом (CMS – Content Management System) – це програмні платформи, які дозволяють створювати, редагувати, організовувати та публікувати цифровий контент, головним чином для веб-сайтів, без потреби глибоких знань у програмуванні.

Основні функції CMS:

- надання інструментів для створення наповнення (даних), організації спільної роботи над наповненням;
- управління даними: зберігання, контроль версій, дотримання режиму доступу, управління потоком документів та ін.;
- публікація наповнення;
- представлення інформації у вигляді, зручному для навігації, пошуку.

Joomla! – система управління даними (CMS), написана на мовах PHP і JavaScript, що використовується в якості сховища бази даних СУБД MySQL або іншої індустріально-стандартної реляційної СУБД. Є вільним програмним забезпеченням, що поширюється під ліцензією GNU GPL.

CMS Joomla! включає в себе мінімальний набір інструментів при початковій установці, який доповнюється в міру необхідності. Це знижує заповнення адміністративної панелі непотрібними елементами, а також знижує навантаження на сервер і економить місце на хостингу.

Drupal – система управління вмістом, що також використовується як каркас для вебдодатків (CMF). Система написана на мові PHP та використовує як сховище даних реляційну базу даних (підтримуються MySQL, PostgreSQL та інші). Drupal є вільним програмним забезпеченням,

Завдання до практичної роботи

1 Дослідити 3 зображення на будь-яких вебсайтах, визначити розмір цих зображень за допомогою Інструментів розробника. У звіті описати проведену роботу та додати скріншот з посиланнями на сторінки цих сайтів, вказати розміри зображень.

2 Описати та додати до звіту 2 скріншота з помилкою в консолі Web Developer при перегляді різних сайтів.

3 Змінити властивості шрифту на сторінці будь-якого сайту (колір та розмір), ширину поля. У звіті описати проведену роботу та додати до звіту скріншот з посиланнями на сторінки.

4 Встановити програму *Screaming Frog SEO Spider* (рекомендовано) або *Xenu Link Sleuth* для пошукової оптимізації.

5 Провести технічне тестування пошукової оптимізації будь-якого сайту за допомогою *Screaming Frog SEO Spider*. У головному вікні потрібно ввести адресу сайту (рисунок 8.1).

6 Після закінчення перевірки сайту за допомогою *Screaming Frog SEO Spider*, в головному вікні виводиться детальна таблиця, в якій можна відсортувати по кожному із стовпців: адреса посилання, статус, mime- тип, розмір, заголовок, рівень вкладеності, кількість зовнішніх та внутрішніх посилань, час віддачі сторінки.

Отримати звіт:

Весь звіт про сторінки (All URLs):

• **Меню → File → Export → All URLs**

або

Правий клік на **"Internal" → Export**.

• Обери формат файлу (CSV, Excel XLSX).

• Збережи звіт на диск.

🔹 Окремі типи звітів:

Можна зберегти звіт по конкретному типу помилок або елементів:

Що потрібно	Як зберегти
Биті посилання (404)	Reports → Response Codes → Client Error (4xx)
Редиректи (301/302)	Reports → Redirects → All Redirects
Title tags / Meta descriptions	Bulk Export → Page Titles / Meta Descriptions
Зображення без ALT	Bulk Export → Images → Missing Alt Text
Сторінки з глибокою вкладеністю	Reports → Site Structure
Внутрішні посилання / перелінковка	Bulk Export → Links → All Inlinks / Outlinks

Описати проведену роботу та додати скріншот звіту *Xenu* (рисунок 5.7).

Зміст звіту

1 Тема та мета роботи.

2 Завдання до роботи.

3 Короткі теоретичні відомості.

4 Хід роботи з виконання завдання до лабораторної роботи.

5 Висновки, що відображують результати виконання роботи та їх критичний аналіз.

6 Перелік використаних джерел.

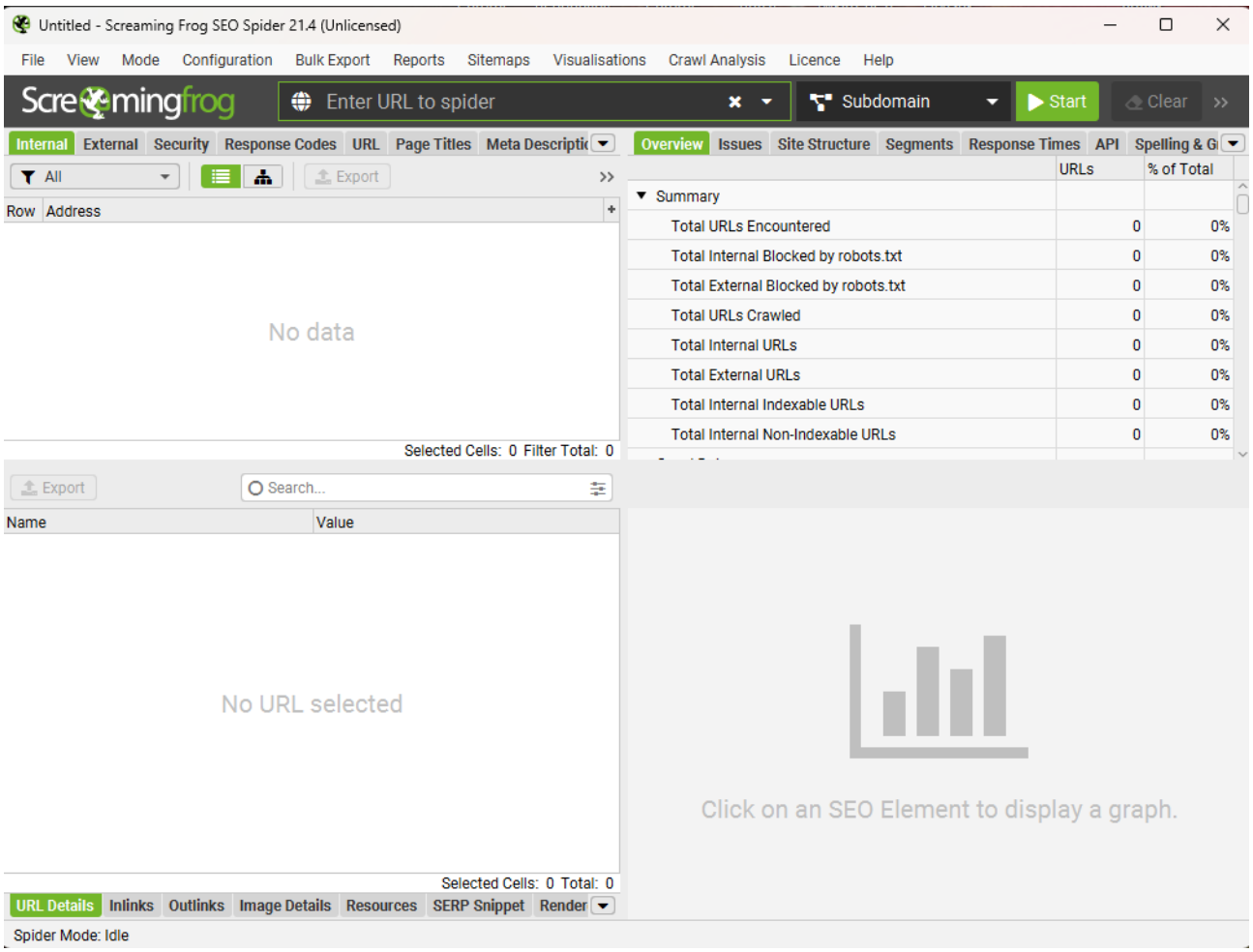

Рис. 8.1. Стартове вікно програми *Screaming Frog SEO Spider*

Контрольні запитання

1 Яким чином можна оцінити проблемні частини сайту за допомогою Dev Tools? Відповідь обґрунтувати та навести приклади.

2 За допомогою якого програмного інтерфейсу можливо змінити «JavaScript» сценарії у реальному часі на сайті?

3 Як визначити за допомогою Dev Tools, коли почалося та закінчилося завантаження файлів на сайті? Відповідь обґрунтувати та навести приклади.

4 В якій послідовності завантажується «JavaScript»? Відповідь обґрунтувати та навести приклади.

5 Перерахувати, які помилки на сайті можна знаходити за допомогою інструментів розробника, навести приклади.

6 Навести приклади інструментів для навантажувального тестування, коротко розповісти про них.

7 Розповісти для чого проводиться навантажувальне тестування?

Практична робота №9
Функціональне тестування

Короткі теоретичні відомості

Функціональне тестування (functional testing) **–** вид тестування, при якому виявляється некоректна/неправильна робота функціоналу програми або процес перевірки відповідності поведінки системи першочергово заявленим функціональним вимогам.

Також, можна сказати, що ***функціональне тестування*** **–** це тестування програмного забезпечення з метою перевірки реалізованості функціональних вимог, тобто здатності програмного забезпечення в певних умовах вирішувати завдання, потрібні користувачам. Функціональні вимоги визначають, що саме робить програмне забезпечення, які завдання вирішує.

При функціональному тестуванні перевіряється коректність роботи системи, яка включає перевірку кожної з функцій програми, адекватність збережених і вихідних даних, методи їх обробки, обробка даних, що вводяться, методи зберігання даних, методи імпорту та експорту даних і т.д. залежно від специфіки додатку. При перевірці проєктів проводиться документація функціональних вимог, що спрощує перевірку.

При функціональному тестуванні проводиться перевірка наступних модулів сайту:

1. Реєстрація (registration, logging in);
2. Авторизація (authorization);
3. Новинний модуль (news);
4. Пошук по сайту (search);
5. Зворотній зв’язок (feedback);
6. Банери (banners);
7. Фотоальбоми (photo album);
8. Форум (forum);
9. Інтернет-магазин (online shop, e-shop);
10. Список, що випадає (drop-down list);
11. Коментарі, поширення в соціальних мережах (sharing);
12. Відео (video);
13. Модуль розсилки (mailout module);
14. Форми (forms).

Під час роботи з даними важливу роль відіграє валідація даних *(data validation).* Перш ніж використовувати отримані від користувача дані, необхідно переконатися, що вони введені правильно і показують коректні значення.

Валідація (validation) **–** це процес перевірки даних на відповідність певним, заздалегідь відомим правилам (форматам, вимогам).

Під час тестування необхідно перевіряти узгодженість валідаторів вхідних даних з логікою обробки цих даних додатком. Для тестування затвердження даних, треба розуміти, як вони повинні працювати, що можна вважати

правильним, а що ні.

«Невалідні» дані, що не задовольняють певним обмеженням, можуть викликати збій у роботі програми.

Валідація даних здійснюється наступним чином:

1. Посимвольна перевірка.
2. Перевірка окремих значень.
3. Сукупність вхідних значень.
4. Перевірка стану системи після обробки даних.

Завдання до роботи

1. Завантажити приклад чек-ліста «*Чек_ліст Функціонал.xlsx*» (з навчальної системи Moodle).
2. Вказати прізвище та ім’я власника у назві контрольного списку.
3. Додати мінімум 3-5 пунктів до існуючих пунктів контрольного списку на вкладці «Функціонал», не повторюючись з вже доданими, та відмітити їх будь-яким кольором.
4. Провести функціональне тестування сайту http://prestashop.qatestlab.com.ua/en/, перевіривши всі пункти контрольного списку на вкладці «Функціонал» мінімум в 3х браузерах (Firefox, Google Chrome, Edge, Opera або інший) в останніх або передостанніх версіях.
5. Результат перевірки за чек-лістом відзначити «Passed/Failed». Для «Failed» вказати в примітці або нотатках посилання на баг-репорти (2 шт), якщо багів буже більше 2, то додати в примітках тему багу за принципом «Що? Де?Коли?».
6. Створити 2 баг-репорти для багів, знайдених під час функціонального тестування http://prestashop.qatestlab.com.ua (або іншого, який тестувався в попередній роботі),оформити їх та додати до звіту.
7. Скриншот доповненого та пройденого чек-ліста додати до звіту.
8. Оформити звіт з лабораторної роботи та надати його на перевірку.

Зміст звіту

1. Тема та мета роботи.
2. Завдання до роботи.
3. Короткі теоретичні відомості.
4. Хід роботи з виконання завдання до лабораторної роботи.
5. Висновки, що відображують результати виконання роботи та їх критичний аналіз.
6. Перелік використаних джерел.

Контрольні запитання

1. Що відносять до дефектів функціонального тестування? Навести приклади та обґрунтувати відповідь.
2. За яким принципом були додані нові пункти до чек-ліста? Обґрунтувати важливість нових пунктів при тестуванні сайту http://prestashop.qatestlab.com.ua/en/.

3. Які основні пункти повинен містити контрольний список (checklist) з тестування функціоналу сайту?

4. Надати пояснення твердженню, що функціональне тестування є трудомістким процесом та може займати до 80% всього бюджету проєкту з тестування.

5. Тестування яких модулів сайту http://prestashop.qatestlab.com.ua/en/ здійснювалось в першу чергу при функціональному тестуванні, навести приклади та обґрунтувати відповідь

Додаткова інформація

Альтернативні платформи для проведення тестування:

1. https://demo-opencart.com/

- **Тип**: Демонстраційний магазин на базі CMS OpenCart.
- **Можливості для тестування**:
  - Пошук товарів
  - Додавання до кошика
  - Оформлення замовлення
  - Реєстрація користувача
  - Робота з формами
- **Типові баги**: Через часте тестування – часто з'являються конфлікти, некоректні відповіді або зламані UI-елементи.

2. https://testpages.eviltester.com/

- **Мова**: Англійська, але чудовий ресурс для практики.
- **Можливості**:
  - Створений спеціально для тестувальників.
  - Є сторінки з формами, HTTP-помилками, таблицями, що генеруються динамічно, pop-up вікнами тощо.
  - Добре підходить для написання тест-кейсів, баг-репортів, перевірки edge-case сценаріїв.

Практична робота №10
Тест-дизайн та тест-кейси

Короткі теоретичні відомості

Тест-дизайн (test design) - це етап процесу тестування програмного забезпечення (ПЗ), на якому проєктуються і створюються тестові випадки (тест-кейси), відповідно з певними раніше визначеними критеріями якості і цілями тестування.

Тестовий випадок або тест-кейс (test case) - це сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації функції, що тестується, або її частини.

Тестовий набір (test suite):

- набір тестів, що реалізують бізнес-завдання, виконуване системою, що тестується;
- ▪ тестовий набір включає тестові сценарії, тестові дані або правила їх генерації.

До стандартних атрибутів тест-кейса відносять:

- ***ID (Test Case ID) –*** унікальний ідентифікатор необхідний для зручної організації зберігання і навігації у групі тестів (Test Suite), переважно генерується автоматично;
- ***назва*** - основна тема чи ідея тест-кейса, короткий опис його суті, який дозволяє зрозуміти призначення тест-кейса. Може повторювати тему. Поле обов'язкове для заповнення;
- ***тема (Summary) –*** поле опису основної теми, яка лаконічно і точно описує його суть. Для уніфікованого підходу до викладу теми використовують конструкцію, яка відповідає на питання «Що перевіряємо? Де перевіряємо? З якими даними?». Поле обов'язково для заповнення;
- ***попередні умови (Preconditions) -*** список всіх необхідних
- підготовчих дій (налаштування програми, середовища тестування) для виконання даного тест-кейса;
- ***кроки для відтворення (Step actions)*** – описують
- послідовність дій для відтворення тестового випадку, які повинні привести до очікуваного результату. Повинні бути короткими і зрозумілими;
- ***очікувані результати (Expected results)*** – визначають
- правильну реакцію програми на виконання даних кроків. Повинні бути зрозумілими, однозначними, простими;
- ***історія редагування -*** лаконічний журнал змін, де
- відображено ким, як і коли був змінений тест-кейс;

Тестові випадки поділяються за очікуваним результатом на ***позитивні*** та ***негативні***.

Позитивний тест-кейс використовує тільки валідні дані і перевіряє, що додаток правильно виконав функцію, що викликається.

Негативний тест-кейс оперує як валідними, так і невалідними даними (мінімум 1 некоректний параметр) і ставить за мету перевірку виняткових

ситуацій (спрацьовування валідаторів).

Тестові випадки зручно об'єднувати ***за призначенням***:

1. позитивні кейси:
- перевірка функціоналу;
- перевірка дизайну/UI;
- перевірка безпеки.

2. негативні кейси.

При складанні тест-кейсів необхідно дотримуватися наступних принципів:

- тему тест-кейса необхідно описувати за принципом «Що
- перевіряється? Де перевіряється? З яким типом даних (валідні, невалідні)?»;
- наявності опису тест-кейса;
- наявність попередніх умов, де має бути вказано, яка
- форма, сторінка відкрита, але посилання слід вказувати тільки на головний домен сайту – не на конкретну сторінку;
- на кожен крок повинен бути очікуваний результат;
- в кроках має бути зазначений тип даних, що вводяться (валідні/ невалідні), у разі невалідних – повинні бути наведені приклади таких даних (наприклад, для поля введення електронної пошти невалідними будуть неприпустимі спеціальні символи, без @, без домена, та ін.);
- тест-кейс повинен бути завершеним і цілісним (наприклад, функція відновлення пароля повинна закінчуватися на успішній зміні пароля, а не на відправці листа з посиланням для зміни пароля).

Завдання до роботи

1. створити 4 тест-кейси для сайту http://prestashop.qatestlab.com.ua/en/ для наступного функціоналу:
- форма реєстрації нового користувача;
- форма авторизації.

Для кожної форми має бути по 2 тест-кейси – один з позитивним сценарієм (з валідними даними) і один з негативним сценарієм (з невалідними даними).

Шаблон тест-кейса «*Тест_кейси.xlsx*» завантажити з навчальної системи Moodle.

2. Скриншоти створених тест-кейсів додати до звіту.

3. Оформити звіт з лабораторної роботи та надати його на перевірку.

Зміст звіту

1 Тема та мета роботи.

2 Завдання до роботи.

3 Короткі теоретичні відомості.

4 Хід роботи з виконання завдання до лабораторної роботи.

5 Висновки, що відображують результати виконання роботи та їх критичний аналіз.

6 Перелік використаних джерел.

Контрольні запитання

1 Дайте визначення поняттю тест-кейс та вкажіть його основні атрибути.

2 Вказати причини того, чому спочатку проводиться

3 позитивне тестування, а потім негативне.

4 В чому полягає важливість використання тестових випадків у тестуванні ПЗ?

5 Вказати переваги та недоліки використання тестових випадків у тестуванні ПЗ на прикладі.

6 В чому полягає різниця між тест-кейсом та звітом про дефект?

Додаткова інформація

Альтернативні платформи для проведення тестування:

1. https://demo-opencart.com/

- **Тип**: Демонстраційний магазин на базі CMS OpenCart.
- **Можливості для тестування**:
  - Пошук товарів
  - Додавання до кошика
  - Оформлення замовлення
  - Реєстрація користувача
  - Робота з формами
- **Типові баги**: Через часте тестування – часто з'являються конфлікти, некоректні відповіді або зламані UI-елементи.

2. https://testpages.eviltester.com/

- **Мова**: Англійська, але чудовий ресурс для практики.
- **Можливості**:
  - Створений спеціально для тестувальників.
  - Є сторінки з формами, HTTP-помилками, таблицями, що генеруються динамічно, pop-up вікнами тощо.
  - Добре підходить для написання тест-кейсів, баг-репортів, перевірки edge-case сценаріїв.

Практична робота 11
Інструментальний засіб jira software.
Тестування програмного забезпечення.

Короткі теоретичні відомості

Питання про роль та місце психології у програмуванні стали виникати багато років тому, при становленні програмування як професійної діяльності. Наскільки справедливо розглядати розробку програмних продуктів лише з погляду обчислювальної математики?

В даний час переважає тверде переконання, що програмування - це рід людської діяльності, де люди створюють програми виключно для людей. Комп'ютер використовується лише як інструмент. Звідси випливає, що розробка програмного забезпечення, зокрема мультимедійних продуктів, має розглядатися з позицій людино-машинної системи.

Стандарт ISO 9241-210 ставить у центр процесу розробки користувальницькі вимоги. Однак концентрація виключно на користувачах може призвести до того, що продукт виявиться невигідним для бізнесу або важким з технічної точки зору. Тому для створення успішного продукту треба враховувати також бізнес-вимоги та технічну складову. Користувачів, бізнес та технології поєднує UX-технології, а точніше UX- тестування.

UX-тестування – комплекс заходів, спрямованих на виявлення будь-яких проблемних місць на вашому ресурсі: чи достатньо зрозумілий, логічний, зручний, чи правильно працюють всі його технічні елементи.

Результатом грамотного UX-тестування є перелік рекомендацій, що і яким чином потрібно змінити, щоб підвищити кількість конверсій та перетворити відвідувачів сайту на його постійних та відданих користувачів.

Тестування виявляє великі та дрібні проблеми інтерфейсу, кожна з яких відсіває ваших потенційних покупців.

Також UX-тестування показує, наскільки зрозумілий покупцям ваш інтерфейс, чи використовують вони його так, як ви задумали, чи зовсім іншим чином. Отже, показують, яким чином потрібно змінити user flow на сайті, щоб користувачам було зручно.

UX-тестування потрібно, якщо ви хочете перевірити існуючий інтерфейс на зручність користувальницьких сценаріїв, відзначити всі "проблемні" місця та покращити їх.

Тестувальнику доводиться працювати з величезною кількістю інформації, вибирати з безлічі варіантів вирішення завдань та винаходити нові. У процесі цієї діяльності об'єктивно неможливо пам'ятати усі думки, а тому продумування та розробку тест-кейсів рекомендується виконувати з використанням «test case».

Test Case – це документ, що описує сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації функції або її частини.

Навіщо потрібний test case?

1. Щоб описати детально, як будемо тестувати функцію.

2. Для будь-якого спеціаліста, який не знайомий із вимогами.

3. Для звітності.

Під час проведення тестування заповнюють так званий Bug report (звіт про знаходження помилки).

Bug report – докладна покрокова інструкція для відтворення помилки.

Атрибути Bug report:

Id – номер звіту в системі.

Summary – короткий опис проблеми, що явно вказує на причину помилки. Описується за принципом «Що? Де? Коли?».

Preconditions – умови, які мають бути дотримані для відтворення помилки.

Steps to reproduce – кроки відтворення помилки.

Actual result – фактичній результат, отриманий після кроків відтворення.

Expected result – очікуваний результат, прописаний у специфікації (вимогах).

Attachments – вкладення, що допомагають відтворити bag та прояснити ситуацію.

Additional info – додаткова інформація.

Author – упорядник bag-report.

Assigned to / Asignee – розробник, який має виправити bag.

Status – поточний стан bag.

Severity – критичність, показник, що відображає вплив дефекту на працездатність програми (зазвичай виставляє тестувальник).

Priority – пріоритет, показник, що визначає порядок робіт (більш інструмент для менеджера).

Environment – оточення, на якому було знайдено bag (ОС, ім’я та версія браузера).

Version – версія продукту, в якій було знайдено bag.

Можливі значення Priority.

High – найвищий пріоритет. Ця помилка має бути виправлена у максимально короткий термін. Наприклад: не працює кнопка «Login».

Medium – помилка, яку необхідно виправити в короткий термін, але вона не критична для даного продукту. Наприклад: не працює кнопка переходу на сторінку магазину в Instagram.

Low – помилка не вимагає швидкого вирішення, але обов’язково повинна бути виправлена. Наприклад: друкарська помилка в тексті на сторінці з описом одного з товарів.

Можливі значення Severity.

Blocker – помилка, яка призвела до повного блокування програм. Цей атрибут означає, що подальша робота неможлива, потрібно вжити термінових заходів для усунення отриманого результату. Наприклад: користувач не може оформити замовлення в інтернет-магазині, оскільки не відкривається кошик.

Critical – критична помилка, яка також потребує термінового вирішення. Це можуть бути збої в системі безпеки або помилка, через яку впав сервер. Система буде працювати з цією помилкою, але ключові функції можуть бути недоступні. Наприклад: користувач не може ввійти у свій аккаунт.

Major – значна помилка. Цей атрибут означає, що можливість працювати з цією функцією є, але при цьому певна частина логіки працює некоректно. Наприклад: не приходить сповіщення про нове повідомлення в месенджері.

Minor – бізнес-логіка працює коректна, але знайдена помилка інтерфейсу користувача. Наприклад: колір кнопки не відповідає основним кольорам сайту.

Trivial – помилка, що має найменшу серйозність, вона на зачіпає бізнес-логіку, користувач може навіть не потрапити на цю помилку. Наприклад: помилка в тексті; не робоче посилання в тексті статті.

При тестуванні програмного забезпечення використовуються інструментальні засоби. Інструментальний засіб управління тестуванням бере на себе всі рутинні технічні операції, які необхідно виконувати в процесі реалізації життєвого циклу тестування. Величезною перевагою також є здатність таких інструментальних засобів відстежувати взаємозв'язки між різними документами та іншими артефактами, взаємозв'язки між артефактами та процесами тощо, підпорядковуючи ці дії системі розмежування прав доступу та гарантуючи збереження та коректність інформації.

В даний час використовується значна кількість інструментальних засобів тестування. У цьому курсі ми будемо використовувати систему Jira.

Jira Software – інструмент управління робочим процесом для команд розробників ПЗ, які хочуть систематизувати і відстежувати свою роботу. Неймовірна гнучкість дозволяє налаштувати Jira відповідно до унікального робочого процесу команди.

Jira – це потужна система для управління проєктами (рис. 11.1), розроблена компанією *Atlassian*, яка широко використовується в галузі розробки і тестування програмного забезпечення. Основне призначення Jira полягає в підтримці процесів планування, моніторингу та контролю виконання завдань у командах, що працюють за гнучкими методологіями, зокрема Scrum та Kanban.

Система забезпечує інструментарій для створення та адміністрування задач (issues), ведення беклогу проєкту, організації спринтів, відстеження дефектів (багів), а також формування звітів щодо ходу виконання проєкту. Кожна задача в Jira має набір атрибутів, таких як тип, пріоритет, статус, виконавець, оцінка складності тощо, що дозволяє забезпечити повноцінне управління життєвим циклом розробки.

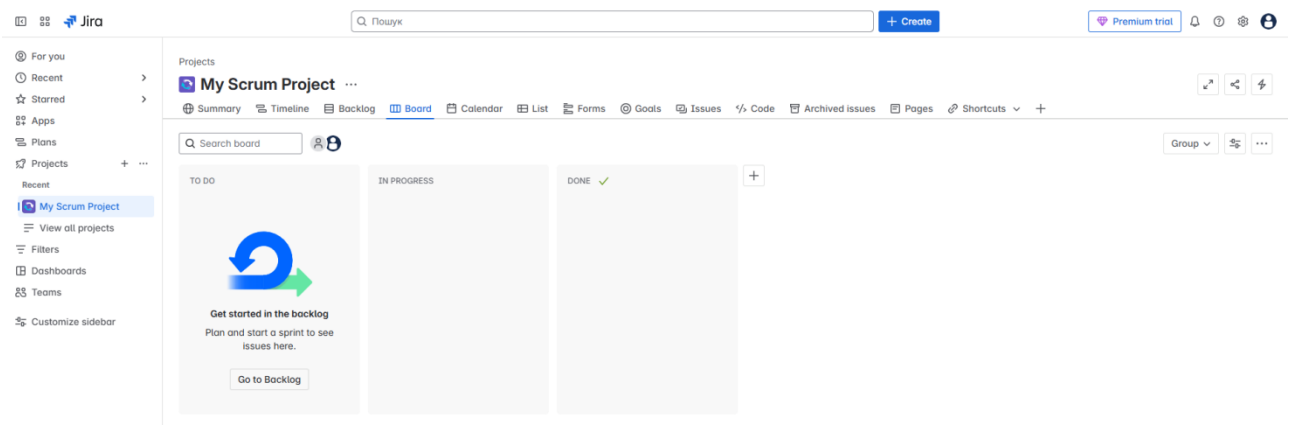

Рис. 11.1. Робочий проєкт Jira

Jira інтегрується з іншими продуктами *Atlassian* (зокрема, Confluence, Bitbucket, Bamboo), а також з великою кількістю сторонніх інструментів, що дозволяє використовувати її як єдину точку контролю в процесі DevOps.

Завдяки гнучкій системі налаштування та розширень, Jira може адаптуватися до різних типів проєктів і команд. Інтерфейс користувача підтримує візуальне представлення задач на дошках, що сприяє прозорості процесів та полегшує координацію роботи між членами команди.

У контексті тестування програмного забезпечення Jira є ефективним інструментом для реєстрації, класифікації та відстеження дефектів, що виникають у процесі тестування. Це дозволяє підвищити якість продукту, зменшити кількість нерозв'язаних проблем та забезпечити зворотний зв'язок між тестувальниками і розробниками.

Життєвий цикл задач у Jira

Життєвий цикл задачі (issue lifecycle) у Jira визначається робочим процесом (workflow), який може бути стандартним або налаштованим під потреби команди. Типовий життєвий цикл задачі включає наступні етапи:

1. **To Do (Заплановано)** – задача створена, але ще не розпочата.
2. **In Progress (У процесі виконання)** – задача перебуває в роботі.
3. **In Review (На перевірці)** – задача виконана і потребує рев'ю (перевірки коду, результатів тестування тощо).
4. **Testing (Тестування)** – задача перевіряється тестувальником.
5. **Done (Завершено)** – задача виконана повністю та пройшла всі перевірки.
6. **Reopened (Пере відкрито)** – задача повернута до роботи у разі виявлення проблем або помилок.
7. **Closed (Закрито)** – остаточний стан, після якого задача не потребує жодних дій.

Порядок виконання практичної роботи

1. Встановити он-лайн версію Jira Software за посиланням: https://www.atlassian.com/ru/software/jira/free
2. Створити новий проєкт.
3. Підготувати шаблон з тестування за зразком:

Preconditions:

Посилання https://d.goit.global/ua/qa/ відкрите в браузері

Steps to reproduce:

Натиснути на іконку "Фейсбук" (в футере)

Проінспектувати посилання

Actual result:

При натисканні на іконку «Фейсбук» (в футере) відкривається посилання на «Телеграм»

Expected result:

При натисканні на іконку «Фейсбук» (в футере) відкривається посилання на сторінку у «Фейсбук»

Environment:

Windows, Firefox 121.0

4. Протестувати сайт https://d.goit.global/ua/qa/ на помилки.
5. Знайти не менше 5 помилок.

Індивідуальне завдання:

9. Зареєструватися в Jira (якщо ще не маєте акаунта).
10. Створити Scrum-дошку для тестування.
11. Протестувати сайт https://d.goit.global/ua/qa/ на помилки.
12. Знайти не менше 5 помилок.
13. Прикріпити знімки екрана готового проєкту Jira у звіт.

Практична робота 12
Використання uml-моделей для створення тест-кейсів

Короткі теоретичні відомості

UML (Unified Modeling Language) – це стандартна мова моделювання, яка використовується для візуалізації, специфікації, розробки та документування компонентів програмного забезпечення. Для тестувальників UML є інструментом, що дозволяє зрозуміти функціональність системи до написання коду, і, відповідно, створити тест-кейси ще на етапі проєктування.

Використання UML у тестуванні сприяє:

- ранньому виявленню дефектів – помилки в логіці можна побачити ще до початку розробки;
- покращенню комунікації між тестувальниками, аналітиками та розробниками;
- чіткому розумінню функціоналу і сценаріїв використання;
- автоматизації створення тестів у деяких випадках (Model-Based Testing);
- контролю повноти тестів – можна оцінити покриття шляхів, станів, сценаріїв.

Основні типи UML-діаграми в тестуванні

Тип діаграми	Призначення в тестуванні	Типи тестів, що витягуються
Use Case Diagram	Ідентифікація функціоналу з точки зору користувача	Smoke tests, acceptance tests
Activity Diagram	Аналіз послідовності дій і логіки	Functional flows, path coverage
Sequence Diagram	Моделювання взаємодії об’єктів	Integration tests, interaction testing
State Machine Diagram	Аналіз поведінки об’єктів у різних станах	State transition testing
Class Diagram	Розуміння структури об’єктів	Тестування об’єктних залежностей

Тест-кейси можна створювати на основі UML-діаграм

1. Use Case Diagram

- Ключова мета: визначити основні сценарії (main flows) та альтернативи (alternate flows).
- Як формулювати тест-кейси:
  - Для кожного use case – принаймні один позитивний і один негативний сценарій.
  - Враховувати умови входу/виходу та передумови.

2. Activity Diagram
- Ключова мета: охопити всі можливі шляхи (paths) і гілки (branches).
- Як формулювати тест-кейси:
  - Один тест на основний потік.
  - Додаткові тести на альтернативні або помилкові потоки.

3. Sequence Diagram
- Ключова мета: перевірити коректність взаємодії між компонентами.
- Як формулювати тест-кейси:
  - Чи відбуваються всі виклики?
  - Чи в правильному порядку?
  - Чи повертають очікувані значення?

4. State Machine Diagram
- Ключова мета: перевірити перехід між станами.
- Як формулювати тест-кейси:
  - Один тест на кожен можливий перехід.
  - Перевірка недопустимих переходів.

5. Class Diagram (опціонально)
- Використовується при об'єктно-орієнтованому тестуванні.
- Тести на залежності між класами, інтерфейсами, атрибутами.

Для виконання роботи можна скористатися онлайн-редакторами PlantUML:
- https://www.plantuml.com/plantuml/uml/
- https://plantuml-editor.kkeisuke.com/

або іншими сервісами на власний розсуд.

Приклад виконання практичної роботи

Опис завдання:

Ви працюєте як тестувальник у команді, що розробляє веб-застосунок для онлайн-банкінгу. Ваша задача – на основі опису нижче створити Use Case Diagram, Activity Diagram і, за бажанням, Sequence або State Diagram.

Опис функціоналу:

1. Користувач може зареєструватися у системі, вказавши ім'я, email, пароль.
2. Після цього він може увійти, використовуючи email і пароль.
3. Якщо пароль забуто, є функція скидання пароля через email.
4. Після успішного входу користувач може переглядати баланс, переглядати історію транзакцій, та ініціювати переказ коштів.
5. Під час переказу користувач вводить суму, IBAN отримувача і підтверджує операцію кодом із SMS.
6. Система перевіряє баланс, і якщо його недостатньо – показує повідомлення про помилку.

7. Усі дії повинні бути логовані.
8. Сесія автоматично завершується після 10 хвилин неактивності.

Необхідно побудувати:
1. **Use Case Diagram**:
- Визначити акторів і основні функції.
- Показати залежності <<include>> / <<extend>>, якщо доречно.

2. **Activity Diagram**:
- Побудувати один з ключових сценаріїв (наприклад: “переказ коштів” або “вхід в систему”).

3. **Sequence Diagram (опціонально)**:
- Для сценарію “переказ коштів” – показати взаємодію між UI, сервером, SMS-сервісом і базою даних.

4. **State Diagram (опціонально)**:
- Побудувати для об’єкта “Сесія користувача”.

Приклад реалізації на основі Use Case Diagram

Актори:
- **Користувач** – взаємодіє з системою.
- **SMS-сервіс** – надсилає SMS-код підтвердження.
- **Система (сервіс логування, БД)** – внутрішній обробник.

Use Cases:
- Зареєструватися
- Увійти
- Скинути пароль ← <<extend>> зі сценарію "Увійти"
- Переглянути баланс ← <<include>> з "Успішний вхід"
- Переглянути історію транзакцій ← <<include>> з "Успішний вхід"
- Переказ коштів
  - Ввести суму, IBAN
  - Отримати SMS-код ← <<include>>
  - Підтвердити переказ ← <<include>>
- Отримати повідомлення про помилку ← <<extend>> з "Переказ коштів" (якщо баланс недостатній)
- Автоматичний вихід із системи ← <<extend>> з "Сесія користувача"
- Логування дій ← <<include>> до всіх основних сценаріїв

Для реалізації в PlantUML **Use Case Diagram** можна описати наступним чином:

```
@startuml
left to right direction
actor User
actor "SMS Service" as SMS

rectangle "Online Banking System" {
```

```
  User --> (Register)
  User --> (Login)
  User --> (Reset Password) : <<extend>>

  (Login) --> (View Balance) : <<include>>
  (Login) --> (View Transaction History) : <<include>>
  (Login) --> (Transfer Funds)

  (Transfer Funds) --> (Enter Amount & IBAN)
  (Transfer Funds) --> (Receive SMS Code) : <<include>>
  (Transfer Funds) --> (Confirm Transfer) : <<include>>
  (Transfer Funds) --> (Show Insufficient Balance Error) : <<extend>>

  (Transfer Funds) --> (Log Action) : <<include>>
  (Login) --> (Log Action) : <<include>>

  User --> (Auto Logout) : <<extend>>
  SMS --> (Receive SMS Code)
}
@enduml
```

Варіанти реалізації представлені на рис. 12.1 та 12.2

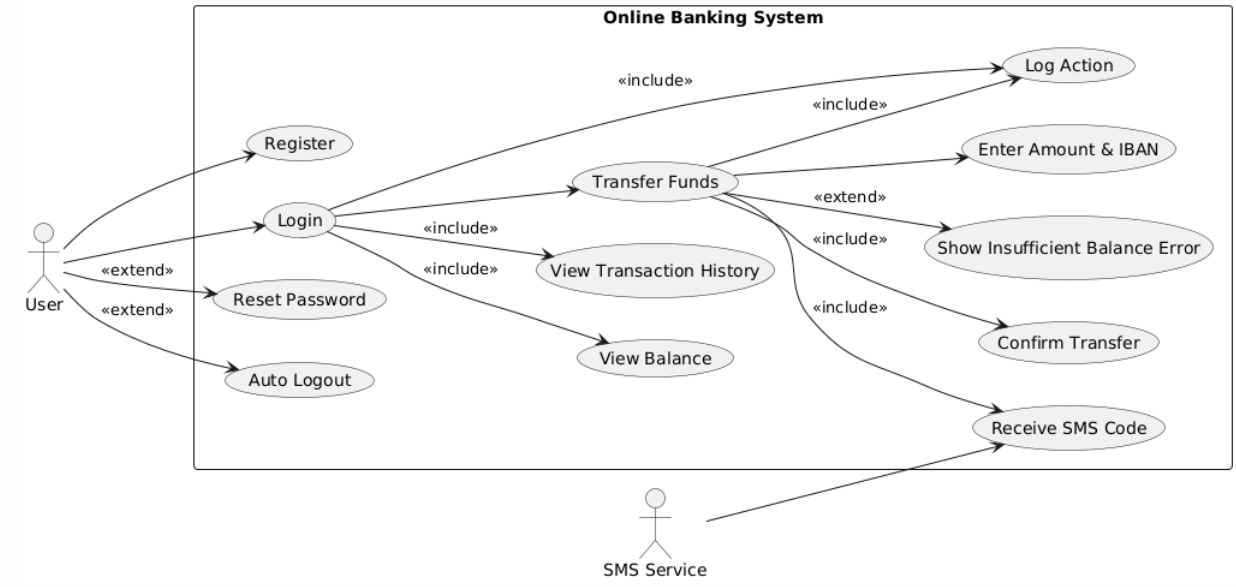

Рис. 12.1. Реалізація за допомогою plantuml.com

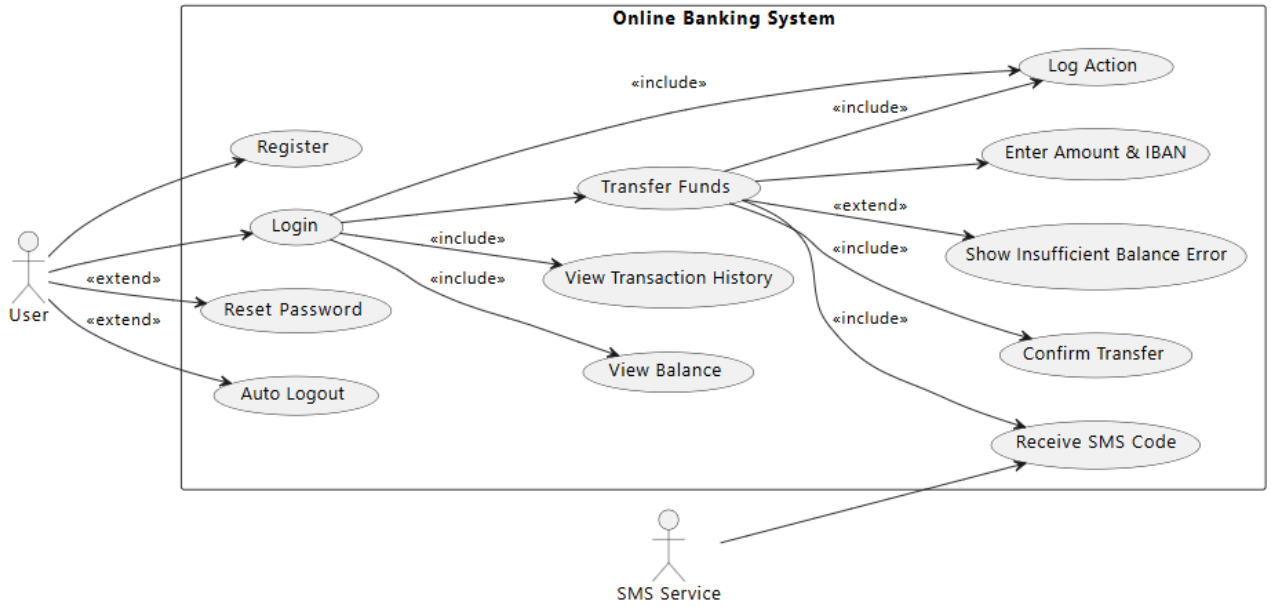

Рис. 12.2. Реалізація за допомогою plantuml-editor.kkeisuke.com

Індивідуальне завдання:

1. Для предметної області (див. Практична робота 1) створити опис і побудувати Use Case Diagram, Activity Diagram, Sequence Diagram (якщо має сенс), State Diagram (якщо має сенс)

ПІСЛЯМОВА

Шановний читачу! Ось ми й дійшли до завершення нашого спільного практичного шляху дванадцятьма роботами, присвяченими тестуванню програмного забезпечення. Сподіваюся, що цей практикум став для вас не просто збіркою завдань, а справжньою майстернею, де ви змогли відточити свої навички, навчитися працювати з інструментами та відчути себе частиною професійної QA-спільноти.

Протягом цих занять ми пройшли довгий шлях – від знайомства з Agile-методологіями до створення складних UML-діаграм для генерації тест-кейсів. Ви навчилися формувати Scrum-команди, працювати з беклогом, створювати Kanban-дошки в Trello, писати історії користувача, визначати їхній пріоритет та оцінювати трудовитрати. Це ті навички, які ви будете використовувати щодня, працюючи в сучасних IT-командах.

Особливу увагу ми приділили практичному тестуванню реальних веб-сайтів. Ви виконували функціональне тестування, перевіряли верстку в різних браузерах, оцінювали зручність використання, аналізували технічні характеристики за допомогою інструментів розробника та спеціалізованих програм. Ви навчилися створювати чек-листи, тест-кейси, баг-репорти – тобто всю ту документацію, яка є основою професійної роботи тестувальника.

Окремим блоком стало знайомство з професійними інструментами. Jira Software відкрила перед вами світ управління проектами та відстеження дефектів, а робота з UML-діаграмами дозволила подивитися на систему з різних точок зору і створювати більш повні тестові набори. Це ті інструменти, які ви зустрінете в більшості комерційних проектів, і володіння ними є вагомою конкурентною перевагою на ринку праці.

Але найголовніше, що я хотів би, аби ви винесли з цього практикуму – це не просто набір технічних навичок, а особливий спосіб мислення. Мислення тестувальника – це постійне прагнення ставити під сумнів, шукати приховане, передбачати непередбачуване. Це вміння дивитися на продукт очима користувача, аналізувати, де саме може критися проблема, і не заспокоюватися, доки не знайдеш відповідь. Кожна виконана вами практична робота, кожен знайдений баг, кожен створений тест-кейс – це цеглинка у формуванні цього мислення.

Пам’ятайте, що шлях професійного розвитку ніколи не завершується. Технології змінюються, з’являються нові інструменти, нові методології, нові види тестування. Те, що сьогодні є передовим, завтра може стати застарілим. Тому найважливіша навичка, яку ви маєте розвивати – це вміння самостійно опановувати нове, адаптуватися до змін і постійно вдосконалюватися. Не зупиняйтеся на досягнутому, експериментуйте, пробуйте нові підходи, вивчайте досвід колег, діліться своїми знаннями.

Щиро дякуємо вам за те, що ви пройшли цей шлях разом з нами. Сподіваємось, що знання та навички, отримані під час виконання практичних робіт, стануть міцним фундаментом для вашої успішної кар’єри в IT. Бажаємо вам цікавих проектів, розумних колег, вдячних користувачів і, звісно, якомога

менше критичних багів у ваших продуктах!

Пам’ятайте: якість починається з вас. Саме ви творите світ, у якому технології служать людям, а не завдають їм клопоту. Пишайтеся своєю професією і розвивайте її!

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Авраменко А. С., Авраменко В. С., Косенюк Г. В. Тестування програмного забезпечення : навчальний посібник. Черкаси : ЧНУ імені Богдана Хмельницького, 2017. 284 с.URL : https://eprints.cdu.edu.ua/1482/1/testyvan.pdf
2. Золотухіна О. А., Негоденко О. В., Резник С. Ю., Разіна С. Я. Якість та тестування інформаційних систем : навчальний посібник. Київ : ННІІТ ДУТ, 2020. 128 с.
3. Повне керівництво з тестування графічного інтерфейсу: Підручник з тестування інтерфейсу користувача. URL: https://uk.myservername.com/gui-testing-tutorial
4. Смагіна О. О., Переяславська С. О. Якість програмного забезпечення та тестування : навч. посіб. до вивчення дисц. для студ. спец. 121 – «Інженерія програмного забезпечення» / Держ. закл. «Луган. нац. ун-т імені Тараса Шевченка». Старобільськ : ДЗ «ЛНУ імені Тараса Шевченка», 2021. 286 с. URL: https://dspace.luguniv.edu.ua/xmlui/bitstream/handle/123456789/7508/2021.pdf?sequence=5&isAllowed=y
5. Трофименко О. Г., Дика А. І. Тестування та забезпечення якості програмних систем : навч. посіб. для підгот. здобув. вищої освіти галузі знань 12 «Інформаційні технології». Одеса : Фенікс, 2024. 195 с. URL: https://doi.org/10.32837/11300.27717
6. Цибульник С. О., Барандич К. С. Технології розроблення програмного забезпечення. Частина 1. Життєвий цикл програмного забезпечення : підручник. Київ : КПІ ім. Ігоря Сікорського 2022. 270 с. URL: https://ela.kpi.ua/bitstream/123456789/50623/1/TRPZ_Ch1_ZhTsPZ.pdf
7. Якість програмного забезпечення та тестування: базовий курс : навчальний посібник / за ред. С. Я. Крепич, І. Я. Співак. Тернопіль : ФОП Паляниця В.А., 2020. 478с. URL: https://surl.lu/gdmifk

Навчальне видання

ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Методичні рекомендації

Укладачі: **Пархоменко** Олександр Юрійович
Тищенко Світлана Іванівна
Ємельянов Святослав Ігорович
Жебко Олександр Олегович
Богатєнкова Олександра Євгенівна

Формат 60х84 1/16. Ум. друк. арк. 4,0.
Тираж 20 прим. Зам. № ____

Надруковано у видавничому відділі
Миколаївського національного аграрного університету
54008, м. Миколаїв, вул. Георгія Гонгадзе, 9

Свідоцтво суб'єкта видавничої справи ДК № 4490 від 20.02.2013 р.