МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

до виконання лабораторних робіт
з дисципліни «Якість інформаційних систем та тестування»
для здобувачів освітнього ступеня «бакалавр»
зі спеціальності 126 Інформаційні системи та технології
(освітньої програми «Web-технології, Web-дизайн»)
усіх форм навчання

Черкаси
2020

УДК 004.05(07)
М 54

Затверджено вченою радою ФІТІС, протокол № 5 від 17.02.2020 р., згідно з рішенням кафедри інформаційних технологій проектування, протокол № 8 від 10.01.2020 р.

Упорядник: Тарасенко Я. В., к.т.н.

Рецензент: Миронець І. В., к.т.н., доцент

М 54 **Методичні** рекомендації до виконання лабораторних робіт з дисципліни «Якість інформаційних систем та тестування» для здобувачів освітнього ступеня «бакалавр» зі спеціальності № 126 Інформаційні системи та технології (освітня програма «Web-технології, Web-дизайн») усіх форм навчання [Електронний ресурс] / [Упоряд.: Я.В. Тарасенко]; М-во освіти і науки України, Черкас. держ. технол. ун-т. – Черкаси : ЧДТУ, 2020. – 30 с.– Назва з титульного екрана.

Методичні рекомендації містять основні теоретичні відомості, практичні завдання до проведення лабораторних робіт та методичні вказівки для виконання цих завдань з дисципліни «Якість інформаційних систем та тестування» з використанням сучасних технологій та програмного забезпечення. Особлива увага приділяється формуванню навичок дослідження і випробування програмного забезпечення для перевірки відповідності реальних програмних систем очікуваній поведінці з урахуванням інженерії вимог та специфікаціям якості програмного забезпечення. В тому числі розглядаються питання тестування web-орієнтованих інформаційних систем, та автоматизації процесу тестування.

Для здобувачів освітнього ступеня «бакалавр» зі спеціальності № 126 «Інформаційні системи та технології» (освітньої програми «Web-технології, Web-дизайн») усіх форм навчання.

УДК 004.05(07)

Виробничо-практичне
електронне видання
комбінованого використовування

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

до виконання лабораторних робіт
з дисципліни «Якість інформаційних систем та тестування»
для здобувачів освітнього ступеня «бакалавр»
зі спеціальності 126 Інформаційні системи та технології
(освітньої програми «Web-технології, Web-дизайн»)
усіх форм навчання

Упорядник **Тарасенко** Ярослав Володимирович
В авторській редакції.

ЗМІСТ

ВСТУП

Навчальна дисципліна «Якість інформаційних систем та тестування» належить до циклу дисциплін професійної підготовки і має міждисциплінарні зв'язки з такими дисциплінами, як «Web-програмування», «Об'єктно-орієнтоване програмування», «Алгоритмізація та програмування», «Професійний практикум», «Безпека інформаційних і комунікаційних систем».

Програма навчальної дисципліни складається з двох частин:

1. Стандарти визначення якості інформаційних систем та тестування.
2. Автоматизоване тестування web-додатків.

Методичні рекомендації налічують опис процесу виконання 8 лабораторних робіт з уточненням мети роботи, коротких теоретичних відомостей та методичних вказівок з виконання роботи, ходу роботи, індивідуальних варіантів завдання та контрольних запитань.

Метою виконання лабораторних робіт є закріплення теоретичних основ визначення якості інформаційних систем, основних понять та визначень, технологій та методів проведення тестування інформаційних систем, загальноприйнятих стандартів тестування програмного забезпечення, отримання навичок застосування основних методів планування та виконання функціонального і навантажувального тестування програмного забезпечення, розробки типових концепцій визначення якості програмного забезпечення та опрацювання нетипових випадків тестування, застосовування стандартів якості інформаційних систем.

Лабораторні роботи спроектовані таким чином, що по завершенню курсу, здобувачі вищої освіти набувають компетенцій для проведення усіх етапів мануального та автоматизованого тестування як інформаційних систем взагалі, так і веб-додатків зокрема. Крім того, здобувачі набувають здатності застосовувати стандарти в області інформаційних систем та технологій при розробці функціональних профілів, побудові та інтеграції систем, продуктів, сервісів і елементів інфраструктури організації, здатності використовувати сучасні інформаційні системи та технології (виробничі, підтримки прийняття рішень, інтелектуального аналізу даних та інші), методики й техніки кібербезпеки під час виконання функціональних завдань та обов'язків, здатності застосовувати інформаційні технології у ході створення, впровадження та експлуатації системи менеджменту якості та оцінювати витрати на її розроблення та забезпечення.

Крім того, матеріали методичних вказівок можуть бути використані при дипломному проектуванні.

При проектуванні лабораторних робіт використані такі новітні методи навчання, як модульне, контекстне (на ранніх етапах репродуктивний метод, на пізніх – евристичний), проблемне навчання. При цьому, проблемне інтегроване з модульним.

Лабораторна робота № 1
Оцінка якості інформаційної системи на основі інженерії вимог

Мета роботи – ознайомитися з базовими принципами якості інформаційних систем, моделями та стандартами якості. Набути навичок чисельної оцінки якості на основі розмірно- та функціонально-орієнтованих метрик. Освоїти особливості інженерії вимог в рамках визначання та забезпечення якості інформаційних систем.

Короткі теоретичні відомості

Якість інформаційної системи визначається шляхом дослідження (в тому числі і чисельному виміру якості, тобто кваліметрії) відповідності системи реальним вимогам та можливості задовольнити потреби користувачів. Виділяють 2 основні напрямки кваліметрії: розмірно-орієнтовані (оцінюють розмір та вагу інформаційної системи) та функціонально-орієнтовані (оцінюють функціонал системи) метрики. До першого напрямку належать: метрики продуктивності та якості (продуктивність=[кількість рядків коду / витрати]; якість=[кількість помилок / кількість рядків коду]) і метрики вартості та документованості (вартість=[вартість в тис. грн. / кількість рядків коду]; документованість = [кількість сторінок документації / кількість рядків коду]). Функціональні метрики враховують, перш за все функціональні покажчики (Functional Points, FP) та обраховуються за формулою (1):

$$FP = UI \cdot (0{,}65 + 0{,}01 \cdot \sum_{i=1}^{n} F_i]) \quad , (1)$$

де UI – оцінка складності користувацького інтерфейсу, $\sum_{i=1}^{n} F_i$ – сума всіх коефіцієнтів регулювання складності, кожен з яких оцінює системні параметри та набуває значення від 0 до 5. Коефіцієнти регулювання складності враховують наступні параметри: обсяг застосованих засобів передачі даних; ступінь розподілу обробки; ступінь поширеності даної апаратної платформи; ступінь жорсткості вимог до продуктивності; частоту виконання транзакцій; складність обробки даних; складність інсталяції; ступінь переносимості та модифікованості.

Для стандартизації перевірки якості використовують моделі якості (CMM, МакКола, SPICE, Боема, FURPS/FURPS+, Гецці, Дромі тощо). Найбільш простою та поширеною є модель МакКола. Її суть у трьох основних напрямках визначення якості: використання (коректність, надійність, ефективність, цілісність, практичність), модифікація (тестованість, гнучкість, супроводжуваність); переносимість (мобільність, можливість багаторазового

використання, функціональна сумісність). Оцінювати усі перелічені чинники слід суб'єктивно за шкалою від 0 до 10.

При цьому, не слід забувати про інженерію вимог, адже в даному контексті контроль якості прямо пов'язаний з контролем за реалізацією вимог до системи. Це означає, що при аналізі вимог, слід керуватися не лише вимогами клієнта, а шукати компроміс між ними та об'єктивними вимогами до системи, основаними на моделях якості. При цьому, важливо розрізняти внутрішні та зовнішні характеристики якості програмного забезпечення. До зовнішніх відносять: коректність, практичність, надійність, ефективність, цілісність, адаптованість, правильність, живучість. До зовнішніх належать: зручність супроводу, портованість, можливість повторного використання, зручність читання, тестованість, зрозумілість. При аналізі вимог клієнтів для оцінці відповідності інформаційної системи саме поставленій задачі слід враховувати відповідність вимог усім внутрішнім та зовнішнім характеристикам якості програмного забезпечення.

Завдання:

Провести різнопланову оцінку якості інформаційної системи, основуючись на висунутих вимогах потенційних клієнтів та існуючих загальноприйнятих стандартах оцінки якості.

Хід роботи:

1. Обрати інформаційну систему згідно індивідуального завдання.
2. Провести оцінку на надати результати обчислення, базуючись на розмірно-орієнтованих метриках.
3. Провести оцінку на надати результати обчислення, базуючись на функціонально-орієнтованих метриках.
4. Перевірити на відповідність вимогам потенційних замовників, результати подати у вигляді графіку та таблиці.
5. Перевірити на відповідність стандарту моделі МакКола, результати подати у вигляді графіку та таблиці.
6. Формалізувати вимоги потенційних замовників на базі моделі МакКола, усунути можливі протиріччя чи дублювання у вимогах клієнтів та надати остаточну оцінку відповідності інформаційної системи поставленій задачі та моделі якості.
7. Формування звіту та висновків.

Примітка 1: *у звіті необхідно надати розрахунки та результати розмірно- та функціонально-орієнтованих метрик, а також графіки та таблиці.*
Примітка 2: *У висновку описати обґрунтовану загальну оцінку системи.*

Типи інформаційних систем відповідно до індивідуального варіанту (номер варіанту визначається за порядковим номером у списку групи):

1. Інтернет-магазин (3 000 рядків коду).

2. Система оцінки знань (тестова система) (4 500 рядків коду).
3. Система бронювання квитків (5 100 рядків коду).
4. Соціальна мережа (18 000 рядків коду).
5. Веб-сайт візитка (800 рядків коду).
6. Інформаційний портал (3 700 рядків коду).
7. Електронний каталог в бібліотеці (2 300 рядків коду).
8. Система електронних платежів (8 500 рядків коду).
9. Форум (7 000 рядків коду).
10. Service Desk система технічної онлайн-підтримки (12 000 рядків коду).

Вимоги до програмного забезпечення, висунуті клієнтами:

1. Пришвидшити час завантаження сторінки.
2. Розмістити додаткові функціональні кнопки з унікальним дизайном, додати більше унікальних шрифтів та забезпечити привабливий інтерфейс за допомогою Java Script.
3. Додати можливість цілодобового супроводу.
4. Оптимізувати роботу форм відправки.
5. Забезпечити доступ до системи лише з ОС Windows 10.

Контрольні запитання

1. Які варіанти визначення «якість інформаційної системи» існують? Який з них найбільш чітко відображає сутність поняття? Дайте визначення поняття «кваліметрія».
2. Які властивості відносять до зовнішніх характеристик якості ПЗ? Дайте їм визначення.
3. Які властивості відносять до внутрішніх характеристик якості ПЗ? Дайте їм визначення.
4. Наведіть найбільш поширену та інші існуючі моделі якості програмного забезпечення (щонайменше 6 найменувань). Коротко охарактеризуйте на яких характеристиках якості вони ґрунтуються.
5. Опишіть розмірно-орієнтовані метрики оцінки якості ПЗ.
6. Опишіть функціонально-орієнтовані метрики оцінки якості ПЗ.
7. Що являє собою інженерія вимог в рамках предмету?

Лабораторна робота № 2
Проведення тестування предмету

Мета роботи – ознайомитися з базовими типами видами, моделями та стратегіями тестування. Набути навичок тестування простого предмету. Освоїти етапи побудови стратегії тестування та класифікацію критеріїв тестування.

Короткі теоретичні відомості

Дослідження інформаційних систем відбувається шляхом їх тестування. Тестування має на меті пошук відмінностей між реальним та необхідним станом системи на основі вимог до неї. Налагодження відрізняється від тестування тим, що саме налагодження направлене на локалізацію та виправлення знайдених відповідностей. Будь-яке тестування складається з трьох основних етапів: створення тестового набору, власне тестування системи та оцінка результатів її роботи. Воно забезпечує: виявлення помилок, перевірку відповідності функціоналу інформаційної системи її основним задачам, визначення ступеня реалізації вимог до системи та виявлення якості системи шляхом аналізу її надійності.

Стратегія тестування складається з наступних складових: 1 –типи застосованих тестів (тестування установки версії; продуктивності; регресійне; інтеграційне; тестування локалізації; кросбраузерності та кросплатформеності), 2 –пріоритети тестування, 3 –оточення для роботи, 4 – робота тестувальника, 5 – тестова документація, 6 – генерація тестових сценаріїв, 7 – порядок роботи з трекером, 8 – критерії початку та завершення тестування, 9 – інструменти для роботи, 10 – оцінка якості проекту та інструменти моніторингу. При обранні стратегії тестування простого предмету слід керуватися тими ж складовими, моделями та видами тестування, що і для інформаційної системи. Необхідно обрати предмет за індивідуальним варіантом та визначити усі складові його випробування. Існують такі моделі тестування: Agile Model (увага акцентується на отриманні швидких та видимих результатів), Rapid Application Development (модель базується на стрімкому еволюційному підході), спіральна модель (в основі лежить підхід використання ряду циклів з послідовних кроків каскадної моделі), Rational Unified Process (процес тестування також розділяється на декілька циклів).

Усі тести поділяються на певні типи, що проводяться у такій послідовності: модульне, інтеграційне, системне тестування та приймальні випробування. Бувають такі види тестування: основні тести (тестування методом чорного, білого та сірого ящика), не функціональні тести (тестування безпеки, сумісності, ефективності, юзабіліті).

При цьому, не слід забувати, що виділяють такі критерії тестування: структурні критерії, або клас 1 (використовують тестування білого ящика на етапах модульного та інтеграційного тестування), функціональні критерії або

клас 2 (відбувається перевірка ступеня дотримання вимог замовника за допомогою методу чорного ящика), стохастичні критерії, клас 3 (застосовуються в тестування об'ємних програмних комплексів), мутаційні критерії, клас 4 (оцінюється загальна кількість помилок на основі дослідження статистики дрібних помилок). До критеріїв тестування висуваються такі вимоги, як достатність, повнота, надійність, можливість легко перевірити. У роботі необхідно створити блок-схему, де від предмету відгалужуються типи та види тестування з коротким описом результату.

Завдання:

Провести всебічне тестування предмету згідно індивідуального завдання та описати послідовність дій при цьому.

Хід роботи:

1. Обрати систему згідно індивідуального завдання.
2. Побудувати стратегію тестування предмету згідно правил побудови стратегії тестування програмного забезпечення.
3. Обрати модель тестування, що найбільше підходить для даного предмету та описати послідовність майбутнього тестування.
4. Застосувати до предмету типи тестування, що використовуються в проекті та занести дані до блок-схеми (примітка 1).
5. Доповнити блок-схему іншими видами тестування згідно класифікації методів тестування.
6. Перевірити класи критеріїв тестування, що були застосовані до предмету та описати на яких етапах побудованої стратегії тестування вони застосовувались.
7. Формування звіту та висновків.

Примітка 1: *у звіті необхідно надати результати тестування у вигляді блок-схеми побудованої навколо предмету, де кожен елемент описує відповідний метод тестування та задовільність властивостей предмету.*

Примітка 2: *У висновку описати чи предмет пройшов тестування, а також чи виконує свої функції і чи виправдовує очікування користувача.*

Типи тестованих предметів відповідно до індивідуального варіанту (номер варіанту визначається за порядковим номером у списку групи):

1. Кулькова ручка.
2. Шкіряний ремінь.
3. Офісний стілець.
4. Підсвічник.
5. Столова виделка.

6. Канцелярська скріпка.
7. Паперова закладка у книжку.

Контрольні запитання

1. Що таке тестування та чим воно відрізняється від налагодження?
2. Що забезпечує процес тестування?
3. З яких етапів складається стратегія тестування?
4. Наведіть типи тестування, що застосовується до проекту.
5. Класифікуйте методи тестування.
6. Назвіть основні моделі тестування.
7. Які існують класи критеріїв тестування?

Лабораторна робота № 3
Розробка та проведення структурного тесту програмного забезпечення

Мета роботи – ознайомитися з перевагами та недоліками структурного підходу тестування. Набути навичок тестування білої скриньки та побудови графу потоку управління з потоком даних. Освоїти особливості використання критеріїв тестування потоків керування та потоків даних програми.

Короткі теоретичні відомості

Структурне тестування передбачає тестування елементів інформаційної системи на основі її відомої структури. Іноді цей вид тестування називають тестуванням білої скриньки, оскільки код програми при цьому доступний. В структурному тестуванні проводиться тестування за маршрутами, визначеними алгоритмом. Отже, звідси слідує, що можна зобразити граф потоку управління програми. Для цього слід пам’ятати основні особливості потокового графу: він є відображенням керуючої структури програми; вершини графа – це лінійні ділянки програми; дуги відображають потік управління (передача керування між операторами); існують операторні (виходить одна дуга) та предикатні (виходять дві дуги) вузли; предикатні вузли символізують прості умови.

Щоб коректно реалізувати структурне тестування, необхідно дотримуватись критеріїв тестування потоків керування (дослідження умов і циклів) та потоків даних програми (дослідження змінних). Під потоками керування розуміють такі критерії: критерій покриття (кожен оператор повинен бути виконаний хоча б 1 раз); критерій покриття рішень (кожне галудження алгоритму повинно бути пройдене принаймні один раз); критерій покриття шляхів (кожна послідовність вершин графа повинна бути досліджена не менше одного разу); граничне тестування циклу (повинно бути здійснено вхід до кожного циклу без виконання ітерацій); внутрішнє тестування циклу (повинно бути здійснено вхід до кожного циклу з виконанням мінімум однієї ітерації). До критеріїв покриття умов і рішень відносять: простий критерій покриття умов (набуття правильних і помилкових значень одиничної умови має бути досліджено не менше одного разу); критерій покриття умов і рішень (набуття правильних і помилкових значень одиничної умови та кожна гілка алгоритму має бути досліджено не менше одного разу); модифікований критерій покриття умов і рішень (кожна одинична умова, яка впливає на загальну умову має бути досліджена за допомогою незалежних тестів). До критеріїв тестування потоків даних програми відносять: all-defs (перевірка правильності ініціації змінних); all p-uses (перевірка ініційованості змінних в предикатних виразах та обчисленнях), all c-uses (перевірка ініційованості змінних в предикатних виразах та обчисленнях), all c-uses / some p-uses (перевірка використання ініційованих замінних або в обчисленнях або в предикатних виразах), all p-uses / some c-uses (перевірка використання ініційованих замінних або в обчисленнях

або в предикатних виразах), all uses (є узагальнюючим для двох попередніх критеріїв), all du-paths (забезпечення найбільшої повноти покриття).

Однак у структурного підходу тестування існують відповідні недоліки, які слід враховувати при виконанні тестування. Це такі недоліки, як: відсутність можливості виявити пропущений маршрут чи помилки, які залежні від обчислень в програмі, а також неможливість підтвердження правильності програми. Однак перевагами є: направленість тестування повне охоплення коду, керування потоком, можливість відстеження цілісності даних, використання внутрішніх граничних точок, залежність тестування від обраного алгоритму. На ці переваги слід опиратися при складанні плану проведення структурного тестування.

Завдання:

Розробити, провести та проаналізувати результати структурного тестування програми за варіантом індивідуального завдання згідно з критеріями тестування програмного забезпечення на основі потоку управління та потоків даних програми.

Хід роботи:

1. Проаналізувати код програми та зобразити граф потоку управління з потоком даних.
2. Скласти план структурного тестування.
3. Виконати тестування, використовуючи критерії структурного тестування на основі потоку управління.
4. Описати тести, проведені для виконання кожної умови.
5. Виконати тестування, використовуючи критерії на основі потоків даних програми.
6. Описати тести, проведені для виконання кожної умови.
7. Сформувати графіки успішності виконання тестів.
8. Формування звіту та висновків.

Примітка 1: *План структурного тестування подати у вигляді таблиці, в якій містяться наступні колонки: назва функціонального модуля, зображення на графі, частота використання, природній розвиток, альтернативне рішення.*

Примітка 2: *у звіті необхідно надати результати тестування у вигляді чотирьох графіків (по 2 для кожного класу критеріїв тестування). По осі X позначити критерій, а по осі Y – кількість успішних тестів (2 графіки). По осі X позначити критерій, а по осі Y – загальну кількість тестів (2 графіки).*

Варіанти програмного забезпечення відповідно до індивідуального варіанту (номер варіанту визначається за порядковим номером у списку групи, зазначений файл додається в електронному вигляді до лабораторної роботи).:

1. var1.txt
2. var2.txt.
3. var3.txt.

4. var4.txt.
5. var5.txt.

Контрольні запитання:

1. Наведіть переваги «White box» тестування.
2. Опишіть особливості структурного тестування.
3. Назвіть недоліки структурного підходу тестування.
4. Яким чином будується потоковий граф програми?
5. Які існують критерії тестування потоків керування програми?
6. Які існують критерії тестування потоків даних програми?

Лабораторна робота № 4
Розробка та проведення функціонального тесту програмного забезпечення

Мета роботи – ознайомитися із функціональним підходом тестування. Набути навичок тестування чорної скриньки, побудови класів еквівалентності та формування граничних умов. Освоїти стохастичне тестування, техніку тестування за методом таблиці прийняття рішень, попарного тестування та варіантів використання.

Короткі теоретичні відомості

В лабораторній роботі необхідно використати методику функціонального тестування. Її завдання полягає у перевірці програми та її функції специфікаціям. При цьому програмний код недоступний, а тому цей метод називають методом чорного ящика. При цьому, відбувається пошук таких помилок: некоректні чи відсутні функції, помилки інтерфейсу, доступу до зовнішньої бази даних, помилки характеристик, ініціації та завершення програми. А тести показують, яким чином виконуються функції програми, як приймаються вхідні дані, як виробляються результати та як зберігається цілісність зовнішніх даних.

Стохастичне тестування характеризується тим, що тести є незалежними, а вхідні дані генеруються випадково. Складність такого підходу в необхідності виконання великої кількості тестів та дослідженні статистичних даних. Основуючись на законі великих чисел, вплив супутніх факторів на досліджуваний показник згладиться та буде можливо визначити взаємозв’язок між явищами.

Одним з підходів функціонального тестування є тестування за класами еквівалентності. Це означає необхідність проведення тестів, які є еквівалентними іншим тестам свого класу. Для визначення класів еквівалентності проводиться розділення множини значень вхідних даних на кінцеве число підмножин (класів). Спочатку будуються класи еквівалентності, а потім розробляються тест-кейси. Побудова класів еквівалентності відбувається за правилами: 1 – якщо умова введення задає діапазон n…m, то визначається один допустимий і два неприпустимих класи еквівалентності; 2 – якщо умова введення задає конкретне значення a, то визначається один допустимий і два неприпустимих класи; 3 – якщо умова задає безліч значень {a, b, c}, то визначається один допустимий і один неприпустимий клас; 4 – якщо умова введення задає логічне значення, наприклад true, то визначається один допустимий і один неприпустимий клас еквівалентності

Наступним підходом є аналіз граничних умов (певні межі вхідної та вихідної умов). Правила тестування з цим підходом полягають в наступному: будувати тести слід для меж множин допустимих значень та для значень, які незначним чином виходять за ці рамки; якщо множина допустимих значень

вхідних даних дискретна, то слід побудувати тест-кейси для мінімального і максимального значення вхідних умов і тести для значень, більших чи менших за ці величини; якщо вхідні й вихідні дані це впорядкована множина, то тестується перший і останній елементи множини.

Крім зазначених підходів тестування чорного ящика, існують і інші не менш дієві. Таким чином, техніка застосування таблиці прийняття рішень наступна: визначаються всі умови, потім складаються всі можливі комбінації умов, видаляються зайві комбінації, визначаються дії, створюються тест-кейси для всіх комбінацій. Техніка попарного тестування: визначення параметрів, кількості значень для кожного параметра, побудова масиву з колонками для кожного параметра, зіставити отриманого прямокутного масиву з метою тестування, формування тест-кейсів. Техніка варіантів використання: розпочати з валідних даних, перевірити граничні значення, провести тести на кожну гілку-альтернативу, виконати операцію в незвичному порядку, виконати транзакцію в зворотному порядку.

Завдання:

Розробити, провести та проаналізувати результати функціонального тестування програми за варіантом індивідуального завдання використовуючи методи функціонального тестування: стохастичний, тестування за класами еквівалентності, аналізу граничних умов та інших підходів функціонального тестування.

Хід роботи:

1. Скласти план виконання тестів із зазначенням кількості за кожною групою методів тестування.
2. Сформувати множину випадкових тестів, провести стохастичне тестування та проаналізувати статистичні залежності у множині отриманих результатів.
3. Побудувати класи еквівалентості, обрати тестові варіанти та провести тестування на основі методу тестування за класами еквівалентності.
4. Сформувати граничні умови, необхідні для виконання всіх етапів поставленої перед програмою задачі та провести ряд тестів, використовуючи особливості методу аналізу граничних умов.
5. Застосувати техніку тестування за методами: таблиця прийняття рішень, попарне тестування та варіанти використання.
6. Сформувати таблицю успішності виконання тестів (примітка 1).
7. Формування звіту та висновків.

Примітка 1: *у звіті необхідно надати результати тестування у вигляді таблиці, в якій містяться наступні колонки: номер тесту, група використаних методів, очікуваний результат, реакція програми.*

Примітка 2: *у висновках зіставити результати проведення функціонального тестування із структурним тестуванням, зробити підсумки щодо ефективності кожного підходу та зазначити область використання кожного з них.*

Варіанти програмного забезпечення відповідно до індивідуального варіанту (номер варіанту визначається за порядковим номером у списку групи, зазначений файл додається в електронному вигляді до лабораторної роботи):

1. var1.txt
2. var2.txt.
3. var3.txt.
4. var4.txt.
5. var5.txt.

Контрольні запитання

1. Що показують тести при «Black box» тестуванні?
2. Які помилки допомагає виявити «Black box» тестування?
3. В чому полягає особливість стохастичного тестування?
4. Яким чином проводиться тестування за класами еквівалентності
5. Наведіть правила формування класів еквівалентності.
6. Що розуміють під граничними умовами?
7. В чому різниця методу граничних умов та методу класів еквівалентності?
8. Які існують методи функціонального тестування та в чому їх суть?

Лабораторна робота № 5
Основи автоматизованого тестування

Мета – ознайомитися із можливостями та особливостями автоматизованого тестування. Набути навичок власноручного написання автоматизованих тест-кейсів для тестування коду та API. Освоїти правила вибору автоматизованого інструментального засобу тестування та основ роботи з такими засобами.

Короткі теоретичні відомості

Мануальне тестування, що застосовувалось при виконанні лабораторних робіт 1-4 мало на меті проведення тест-кейсів без використання програмних засобів. Автоматизоване ж тестування передбачає запуск тест-кейсів за допомогою програмних скриптів. Мануальне рекомендується використовувати при дослідницькому, стохастичному та юзабіліті тестуванні. Автоматизують зазвичай тестування навантаження, локалізації та регресійне. Однак автоматизація можлива в будь-якому виді тесту, різниця лише в ефективності.

До початку процесу тестування, необхідно застосувати відповідні прийоми підготовки цільового програмного забезпечення, а саме використати стаби та моки. Стаби – це певні заглушки, які повертають визначений результат замість виконання дій в тих місцях, як не впливають на процес тестування. Моки – це інший різновид заглушки, що використовується для перевірки чи функція біла викликана. При цьому тест передає функції відповідні вхідні дані та відбувається перевірка повернення очікуваного результату. Ці критерії необхідно описати в роботі до початку тесту (наприклад якщо тестується функція рішення квадратного рівняння, то заздалегідь робиться список відповідей до рівнянь).

Існують різні типи автоматизованого тестування: автоматизоване тестування коду, автоматизоване тестування графічного інтерфейсу (GUI) та автоматизоване тестування API. Тестування коду має на увазі тестування програмних модулів, класів чи бібліотек. Його види: юніт-тести (ізольоване тестування одного модуля, функції чи класу); інтеграційні тести (тестування компоненту інформаційної системи, що складається за багатьох модулів). GUI тестування передбачає використання спеціального програмного забезпечення для генерації дій користувача (натискання клавіш, кліки мишкою). Тестування API передбачає тестування інтерфейсу для взаємодії з користувачем чи іншими програмами. І хоча повністю стохастичне тестування не піддається автоматизації, однак можливо автоматизувати процес аналізу даних для виявлення певних закономірностей, використовуючи математичну статистику, що і слід зробити в роботі.

Вибір інструментів тестування досить широкий, однак існують певні критерії, яких слід дотримуватись при виборі автоматизованого засобу

тестування. Це такі критерії, як: вартість, підтримка, термін життя та інтеграція. Існують як безкоштовні, так і платні засоби. При цьому, характеристики кожної групи впливають на вибір. Безкоштовні інструменти (Selenium, Katalon Studio, Watir, Robot framework) відрізняються дешевизною, можливістю розширення, підтримкою нових технологій, відсутністю технічної підтримки, вузько направленістю інструменту та високим рівнем знань тестувальника. Платні інструменти (Ranorex, Tricentis Tosca, TestPlant eggplant, TestComplete, IBM Functional Tester, UFT) характеризуються офіційною підтримкою, широким профілем охоплення тестованих технологій, інтеграцією з іншими продуктами, високою як вартістю так і якістю, неможливістю самостійного розширення.

При тому описані характеристики в одних ситуаціях можуть виступати недоліками, а в інших перевагами, тому в роботі слід зважати на конкретну задачу, що ставиться перед вибором інструменту. Враховувати слід також розподіл інструментів на універсальні та спеціалізовані. Недоліком спеціалізованих є можливість тестування лише одного типу додатків, в свою чергу ж універсальні можуть бути менш стабільними.

Завдання:

Розробити власні автоматизовані тест-кейси для спрощення процесів тестування. Обрати готові автоматизовані інструментальні засоби, керуючись базовими критеріями вибору інструментарію та провести тестування за його допомогою.

Хід роботи:

1. Визначити критерії автоматизованого тестування коду та API тестування відповідно до програми в індивідуальному варіанті завдання.
2. Реалізувати автоматизований тест-кейс для проведення тестування коду (юніт-тест будь-якого модуля коду, застосовуючи стаби (stub) та моки (mock)).
3. Реалізувати автоматизований тест-кейс для проведення тестування API для перевірки взаємодії програми з людиною (введення/виведення даних).
4. Реалізувати програмний засіб аналізу даних, отриманих в ході стохастичного тестування (лаб. 4).
5. Обрати та обґрунтувати вибір автоматизованого інструментального засобу тестування у вигляді таблиці (примітка 1).
6. Провести будь-який вид автоматизованого тестування обраним інструментом та надати результати.
7. Формування звіту та висновків.

Примітка 1: *у звіті необхідно надати обґрунтування вибору інструменту тестування у вигляді таблиці, що містить наступні пункти: вартість, підтримка, термін існування, інтеграція, універсальність.*

Примітка 2: *у висновках порівняйте ефективність застосування мануального та автоматизованого тестування.*

Варіанти програмного забезпечення відповідно до індивідуального варіанту (номер варіанту визначається за порядковим номером у списку групи, зазначений файл додається в електронному вигляді до лабораторної роботи).

1. var1.txt
2. var2.txt.
3. var3.txt.
4. var4.txt.
5. var5.txt.

Контрольні запитання

1. Чим відрізняється ручне тестування від автоматизованого?
2. В яких видах тестування рекомендується застосовувати мануальний, а в яких автоматизований підхід?
3. Наведіть види автоматизованого тестування та опишіть основні особливості кожного з них.
4. Які існують критерії вибору програмного засобу тестування?
5. Наведіть основні переваги та недоліки безкоштовних автоматизованих інструментів тестування.
6. Наведіть основні переваги та недоліки платних автоматизованих інструментів тестування.
7. Якими недоліками володіють універсальні та спеціалізовані інструменти для автоматизації тестування?
8. Зазначте найбільш популярні інструменти та фреймворки для автоматизації тестування програмного забезпечення.

Лабораторна робота № 6
Розробка та проведення навантажувального тесту

Мета – ознайомитися із особливостями та видами навантажувального тестування. Набути навичок проведення автоматизованого стресового тестування та тестування продуктивності. Освоїти правила побудови моделі навантаження.

Короткі теоретичні відомості

Для забезпечення перевірки продуктивності та працездатності інформаційної системи застосовується тестування навантаження. Його цілі полягають в: оцінці продуктивності і працездатності додатка на етапі розробки, передачі в експлуатацію та оновлення, оптимізації продуктивності системи, підборі платформи і конфігурації сервера. Навантажувальне тестування буває наступних видів: тестування продуктивності (визначення масштабованості додатка при навантаженні), стресове тестування (перевіряється працездатність в умовах пікових навантажень), об'ємне тестування (оцінка поведінки програми при збільшенні об'єму бази даних), тестування стабільності (перевірка працездатності при тривалому використанні).

Перш ніж переходити до навантажувального тестування, розробляють модель навантаження. Для цього визначають список тестованих операцій (критичні з точки зору бізнесу та технічних характеристик), інтенсивність виконання операцій та залежність зміни інтенсивності виконання операцій від часу. Далі слід виділити специфіку моделі навантаження: вивчення додатку (його компонентів, критичних операцій); визначення профілю навантаження (профіль навантаження описується у вигляді «Операція_n – інтенсивність виконання n раз / одиницю часу»), розрахунок навантажувальних точок (кожна операція перевіряється відповідною кількістю віртуальних користувачів). При цьому не слід забувати, що інтенсивність виконання операцій є частотою виконання операцій за одиницю часу. Віртуальний користувач – змодельований програмою циклічний процес. Продуктивність – кількість виконуваних операцій на одиницю часу. Навантажувальна точка – певна кількість віртуальних користувачів в групах, які виконують операції з відповідними інтенсивностями.

Приклад розробки моделі навантаження для кількості операцій = 300 за 6 години при 30 користувачах. Визначається кількість операцій для кожного користувача 300/30 = 10. Отже, за першу годину кожен користувач виконує 10/6 = 1,6 операцій. Після цього визначається інтенсивність за кожним користувачем 60 / 1.6 = 36,5 хвилин. Для зменшення часу виконання тесту до рамок лабораторної роботи інтенсивність збільшується в 5 разів, при цьому зменшивши кількість користувачів також в 5 разів. 36,5/5 = 7,3 хв. = 438 секунд

Таким чином, 6 віртуальних користувачів, виконуючи операцію з періодом 7,3 хвилини, забезпечать за 6 годин продуктивність в 300 операцій.

Автоматизувати процес навантажувального тестування можна використовуючи програмні засоби: Load Impact (задаються параметри навантаження та автоматично моделюється група віртуальних користувачів) чи Apache JMeter (підходить для сайтів, а не веб-додатків з використанням методу POST. Для роботи необхідні: JMeter, Java, Access log сайту).

Тест продуктивності, в свою чергу перевіряє стабільність, масштабованість і можливість використання системи та інфраструктури. Тестування продуктивності сайту відбувається в 2 етапи: емалювання користувацьких запитів, порівняння очікуваних показників продуктивності з реальними. При цьому перевіряється: час відгуку, максимально дозволене навантаження, максимальне навантаження, при якому система зберігає працездатність, середній час роботи до відмови, налаштування продуктивності. Слід пам’ятати, що критерії продуктивності повинні бути: вимірними, кількісними, прогнозованими та зрозумілими.

Завдання:

Розробити модель навантаження згідно показникам за варіантом індивідуального завдання, провести автоматизований навантажувальний тест, використовуючи відповідні інструментальні засоби та проаналізувати результати.

Хід роботи:

1. Створити простий web-сайт (допускається використання конструкторів сайтів) з мінімальним функціоналом та розмістити його на будь-якому безкоштовному хостингу.
2. Визначити список тестованих операцій, інтенсивність виконання операцій та залежність зміни інтенсивності виконання операцій від часу.
3. Розробити модель навантаження з огляду на її специфіку.
4. Провести стресове тестування (Stress Testing) за допомогою програмних засобів Load Impact та JMeter, або обрати (примітка 1) будь-які 2 автоматизовані засоби стресового тестування та провести перевірку створеного web-сайту.
5. Провести тестування продуктивності (Performance Testing) створеного web-сайту.
6. Надати отримані дані щодо рекомендованої оптимальної кількості одночасних віртуальних користувачів, інтенсивності виконання операцій та навантаження, а також виявленої продуктивності сайту.
7. Формування звіту та висновків.

Примітка 1: *якщо було обрано будь-які інші засоби тестування, потрібно надати детальне обґрунтування доцільності використання саме їх, основуючись на правилах вибору інструментально-програмних засобів автоматизованого тестування (лаб. 5).*

Примітка 2: *у висновках необхідно надати детальний аналіз отриманих результатів з рекомендаціями щодо можливого граничного навантаження сайту та щодо покращення його продуктивності.*

Показники навантаження відповідно до індивідуального варіанту (номер варіанту визначається за порядковим номером у списку групи):

1. Кількість операцій 300 за 5 годин. Користувачів – 25.
2. Кількість операцій 400 за 3 годин. Користувачів – 20.
3. Кількість операцій 500 за 4 годин. Користувачів – 35.
4. Кількість операцій 200 за 3 годин. Користувачів – 25.
5. Кількість операцій 100 за 1 годину. Користувачів – 20.

Контрольні запитання

1. Дайте визначення поняттю «навантажувальне тестування» та опишіть його цілі.
2. Які існують види навантажувального тестування?
3. Дайте визначення поняттям: віртуальний користувач, продуктивність, профіль навантаження, навантажувальна точка.
4. В чому полягає специфіка моделі навантаження?
5. Наведіть особливості тестування за допомогою програмних засобів Load Impact та Apache JMeter.
6. Яким чином відбувається перевірка продуктивності сайту?
7. Якими повинні бути критерії продуктивності?
8. Що необхідно перевіряти при тестуванні продуктивності?

Лабораторна робота № 7
Розробка та проведення тесту користувацького інтерфейсу

Мета – ознайомитися з особливостями складання звіту про дефекти. Набути навичок проведення автоматизованого GUI тестування та UI тестування інтерактивного прототипу веб-додатку. Освоїти використання схеми зв'язку проблем в розробці програмного забезпечення.

Короткі теоретичні відомості

Автоматизація тестування веб-додатків вимагає використання певного інструментального комплексу: SpecFlow (DSL), NUnit (тестовий фреймворк), PageObject + PageElements (UI абстракція), Selenium.WebDriver. Основна суть в інкапсуляції логіки поведінки сторінки в класі сторінки.

При проведенні тестування користувацького інтерфейсу моделюються дії користувача з метою випробування логіки додатка та виявлення очікувань користувачів. Для цього слід будувати інтерактивний прототип, тобто копію інтерфейсу досліджуваного веб-ресурсу з клікабельними елементами. Основні принципи тестування інтерактивного прототипу полягають в наступному: створюючи інтерфейс необхідно враховувати юзабіліті; після реалізації UI слід створити інтерактивний прототип основних (чи сумнівних) сценаріїв; залучати осіб, не пов'язаних з розробкою системи; при тестуванні не слід коментувати дії користувача; при дослідженні дизайну чи сценарію слід просити користувача вчинити необхідну дію.

При тестування графічного інтерфейсу (GUI) зазвичай перевіряють такі елементи: Radio, Checkbox, TextBox, ListBox. Для тестування слід дотримуватись списку необхідних перевірок: розмір, положення, ширина, довжина і акцепт символів або цифр для всіх елементів; відповідність графічного інтерфейсу своєму функціоналу; правильність відображення повідомлень про помилки; наявність чіткого розмежування різних ділянок на екрані; читабельність використаних шрифтів; вирівнювання тексту; естетичність кольору шрифту і попереджувальних повідомлень; прозорість та розмір зображень; вирівнювання зображень та заголовків; розташування елементів графічного інтерфейсу для різної розподільчої здатності екрану; правильність орфографії; колір гіперпосилань.

Процес GUI-тестування також можливо автоматизувати. Для цього використовуються наступні засоби: Java – мова написання скриптів для тестування; Maven – засіб для збірки проекту; Selenide – фреймворк для написання GUI тестів; TestNG –фреймворк для керування запуском тестів; Selenoid – засіб управління сесіями браузера; Allure – засіб створення звіту; Jenkins – для безперервної інтеграції тестів в процес розробки.

Важливо пам'ятати, що тестування проводиться для виявлення дефектів (невідповідність виконуваних функцій очікуваним). Відповідно до силабусу

ISTQB, спрощена схема проблем системи виглядає так: помилка -> дефект -> збій чи відмова. Або ж у більш широкому розумінні (рис. 1):

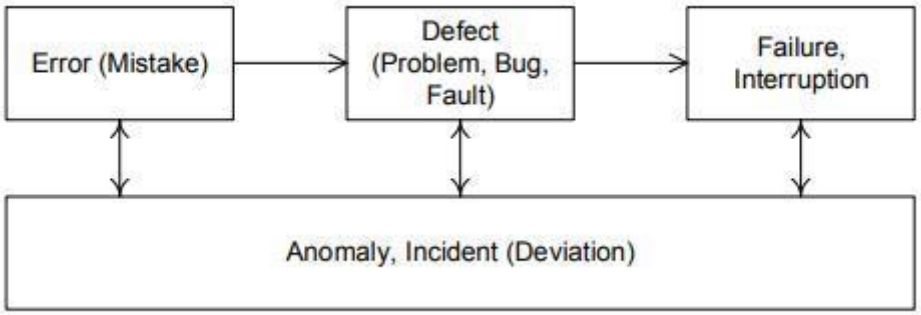

Рисунок 7.1 – Зв'язок проблем в розробці програмного забезпечення

Виявлення дефектів на усіх етапах життєвого циклу інформаційної системи та її тестування супроводжується звітом, який описує дефект та сприяє його усуненню. Звіт складається з метою надати інформацію про проблему, пріоритезувати її, сприяти усуненню. У звіті вказуються наступні пункти: виявлено (початковий стан), призначено (призначено відповідального за усунення), виправлений (усунено дефект), перевірений (підтвердження усунення), закрито (з дефектом не планується подальших дій), відкрито заново (дефект відтворюється), рекомендований до відхилення, відхилений (замість стану закрито), відкладений (виправлення дефекту не є раціональним на часі). В стан «Закрито» звіт може переводитись з резолюціями: не є дефектом, дублікат, не вдалося відтворити, не буде виправлено, неможливо виправити.

Завдання:

Реалізувати автоматизоване тестування веб-додатку. Провести тестування інтерактивного прототипу користувацького інтерфейсу та автоматизувати процес тестування графічного користувацького інтерфейсу. Сформувати звіт щодо виявлених дефектів в процесі тестування, враховуючи силабус ISTQB.

Хід роботи:

1. Використовуючи інструментальний засіб Selenium IDE, як розширення до браузера Firefox провести автоматизоване тестуваня web-сайту, створеного в лаб. 6. (Примітка 1)
2. Створити прототип користувацького інтерфейсу web-ресурсу згідно варіанту індивідуального завдання та провести UI тестування інтерактивного прототипу (Примітка 2).
3. Провести GUI тестування згідно списку необхідних перевірок графічного інтерфейсу.
4. Реалізувати автоматизований тест GUI.
5. Створити звіт про кожен виявлений дефект, керуючись схемою зв’язку проблем в розробці ПЗ згідно ISTQB.

6. Провести всі етапи життєвого циклу звіту про дефект та надати рекомендації щодо можливого їх усунення.
7. Формування звіту та висновків.

Примітка 1: *у висновках надати результати аналізу проведеного тестування з описом виявлених недоліків (якщо такі присутні).*

Примітка 2: *достатньо приблизної копії головної сторінки з основним функціоналом переходів та форм заповнення.*

Показники навантаження відповідно до індивідуального варіанту (номер варіанту визначається за порядковим номером у списку групи):

1. Ресурс для вирішення систем лінійних рівнянь

http://ua.onlinemschool.com/math/assistance/equation/gaus/

2. Тренувальне онлайн-тестування для підготовки до ЗНО:

http://lv.testportal.gov.ua:8080

3. Оналайн-редактор для створення тестів:

http://master-test.net/uk

4. Ресурс для перевірки орфографії:

https://ukrainskamova.com/index/wiki/0-30

5. Сервіс транслітерації:
6. https://pasport.org.ua/vazhlivo/transliteratsiya

Контрольні запитання

1. Опишіть технологічний стек автоматизації веб-додатків.
2. Наведіть зв'язок проблем в розробці ПЗ згідно ISTQB.
3. Назвіть цілі звіту про дефекти.
4. Наведіть стадії життєвого циклу звіту про дефекти.
5. В чому суть UI тестування?
6. Опишіть головні принципи тестування інтерактивного прототипу.
7. Наведіть список необхідних перевірок тестування GUI.
8. Перелічіть інструментальні засоби автоматизації GUI тестування.

Лабораторна робота № 8
Розробка та проведення тесту захищеності web-додатку

Мета – ознайомитися з тестування інфраструктури коду та протоколу передачі даних. Набути навичок проведення тестування захищеності веб-додатку із застосуванням автоматизованих сканерів уразливостей. Освоїти особливості сумісної розробки з використанням сервісу GitHub.

Короткі теоретичні відомості

Інфраструктурний код вирішує технічні деталі реалізації проекту, не зачіпаючи при цьому бізнес цілі. При цьому перевіряються синтаксис і стиль написання коду, функціонал та результат роботи системи управління конфігурацією (програмних комплексів для централізованого керування конфігурацією програмного забезпечення). Існують такі системи управління конфігурацією для тестування інфраструктурного коду: Chef, Puppet, Ansible, SaltStack. Перевірку мови програмування проводять із застосуванням rubocop для Ruby і pep8 для Python. Стиль коду перевіряють за допомогою Chef Foodcritic, Puppet-lint, Ansible-lint, Salt-lint. Функціонал перевіряють наборами тестових даних (кукбуки, модулі, ролі чи формули). Щоб перевірити результат роботи системи управління конфігурацією застосовують Serverspec. Автоматизується процес за допомогою систем безперервної інтеграції (Kitchen CI, Travis CI).

Тестування захищеності веб-ресурсу – це один з видів тестування, що має на меті перевірку безпеки ресурсу, аналіз його вразливостей та ризиків. Загальні принципи безпеки базуються на принципах конфіденційності, цілісності та доступності. Перевіряють такі аспекти: контроль доступу, аутентифікацію, валідацію вхідних значень, криптографічні алгоритми, механізми обробки помилок, конфігурацію сервера, інтеграцію із сторонніми сервісами, стійкість до Dos / DDos атак. Ступінь безпеки веб-додатку в роботі слід оцінити відповідно до міжнародного стандарту OWASP (Open Web Application Security Project). Інформацію щодо стандарту можна отримати за посиланням https://owasp.org/www-pdf-archive/ASVS_3_0_Ukrainian_Beta.pdf.

Тестування безпеки також передбачає пошук вразливостей веб-ресурсу, які бувають наступних видів: XSS (Cross-Site Scripting – виконання шкідливих скриптів); XSRF / CSRF (Request Forgery – використання недоліків HTTP протоколу); Code injections (SQL, PHP, ASP і т.д. – запуск зловмисником запуск виконуваного коду); Server-Side Includes (SSI) Injection (вставка серверних команд в HTML код або запуск їх безпосередньо з сервера); Authorization Bypass (можливість отримання несанкціонованого доступу до облікового запису користувача).

Пошук згаданих вразливостей можливо проводити також за допомогою автоматизованих сканерів захищеності. Вони допомагають виявити вразливості:

етапу кодування; етапу впровадження та конфігурації; етапу експлуатації веб-сайту. Усі сканери поділяються на наступні категорії: мережеві сканери (Nmap, IP Tools); сканери пошуку вразливостей у веб-скриптах (Nikto, Skipfish, Wapiti); засоби пошуку експлойтів (Metasploit Framework, Nessus); засоби автоматизації ін'єкцій (SQLMap); дебагери (Burp Suite, Fiddler); універсальні утиліти (Web Application Attack and Audit Framework, N-Stalker Web Application Security Scanner X Free Edition, N-Stealth).

На захищеність веб-додатку також впливає надійність протоколу передачі даних. Тестування проводиться за наступною схемою: тестування продуктивності, спільного функціоналу, взаємодії та функціональне тестування. Для тестування протоколу HTTP використовується метод HEAD, який працює схожим чином із методом GET, однак сервер у відповідь не посилає тіло HTTP повідомлення. Підхід використовується для тестування HTTP з'єднань і досяжності вузлів і ресурсів

Завдання:

Розробити тест інфраструктури веб-ресурсу та провести різнобічне тестування його вразливостей шляхом використання автоматизованих сканерів з метою тестування безпеки веб-ресурсу та перевірки протоколу передачі даних, застосовуючи засоби автоматизації тестування протоколу.

Хід роботи:

1. Провести тестування інфраструктури веб-ресурсу, розробленого під час виконання лаб. 6.
2. Розмістити файли проекту на веб-сервісі GitHub та підготувати його для сумісної розробки.
3. Провести ряд перевірок, відповідно до тестування захищеності веб-ресурсу та, на основі отриманих даних оцінити його ступінь безпеки (Примітка 1).
4. Застосувати ряд автоматизованих сканерів уразливостей згідно індивідуального варіанту та перевірити реакцію захисних механізмів продукту.
5. Запропонувати комплекс заходів щодо підвищення рівня захисту веб-додатку (Примітка 2).
6. Провести тестування протоколу HTTP методом HEAD.
7. Автоматизувати тестування протоколу з використанням інструменту Postman.
8. Формування звіту та висновків.

Примітка 1: *для оцінки ступеня безпеки використати міжнародний стандарт підтвердження безпеки додатків OWASP (Open Web Application Security Project).*

Примітка 2: *надати перелік виявлених зауважень та дефектів з градацію по критичності вразливостей.*

Використання категорії сканерів захищеності відповідно до індивідуального варіанту (номер варіанту визначається за порядковим номером у списку групи):

1. Мережеві сканери.
2. Сканери пошуку вразливостей в веб-скриптах.
3. Засоби пошуку експлойтів.
4. Засоби автоматизації ін’єкцій.
5. Універсальні утиліти.

Контрольні запитання

1. Які перевірки включає в себе тестування інфраструктури?
2. Що представляють собою системи управління конфігурацією? Наведіть приклади подібних систем.
3. Дайте визначення поняттю «тестування безпеки» та наведіть принципи безпеки програмного забезпечення.
4. Які параметри підлягають перевірці під час тестування безпеки?
5. Які існують види вразливостей та яким чином проводять їх пошук?
6. Наведіть приклади автоматизованих сканерів уразливостей за кожною категорією.
7. Опишіть схему тестування протоколу передачі даних.
8. Яким чином проводиться тестування протоколу HTTP?

ЛІТЕРАТУРА

1. Руда О.А., Моденов Ю.Б. Методи та моделі тестування програмного забезпечення. Проблеми інформатизації та управління. 2013. Том 2, № 42. С. 93-98.
2. Новикова К.В., Люта М.В., Розломій І.О. Аналіз методів тестування програмного забезпечення. *Сучасні перспективи розвитку науки: матеріали Міжнародної науково-практичної конференції (м. Київ, 15-16 вересня 2017 року)*. С.61-63.
3. Desikan S., Ramesh G. Software Testing: Principles and Practice. Bangalore : Dorling Kindersley, 2006. 480 p.
4. Порошин С.М., Можаєв О.О., Можаєв М.О. Методологія проведення pen-тестування веб-додатків. *Системи обробки інформації*. 2016. Випуск 3 (140). С. 33-35.
5. Quadri S. M. K., Sheikh U.F. Testing Techniques Selection: A Systematic Approach. *INDIACom-2011: proceedings of the 5th National Conference (New Delhi, March 10 – 11, 2011)*. P. 279-281.
6. Blokdyk G. Black-Box Testing a Complete Guide – 2019 Edition. Brisbane : Emereo Pty Ltd, 2018. 312 p.
7. Липаев В.В. Тестирование компонентов и комплексов программ: учебник. М.-Берлин : Директ-Медиа, 2015. 528 с.
8. Бейзер Б. Тестирование «черного ящика». Технология функционального тестирования программного обеспечения. СПб. : Питер, 2004. 318 с.
9. Василенко Н.В., Макаров В.А. Модели оценки надежности програмного обеспечения. *Вестник НГУ*. 2004. №28. С. 126–132.
10. Горбаченко И.М. Оценка качества программного обеспечения для создания систем тестирования. *Фундаментальные исследования*. 2013. № 6 (часть 4). С. 823-827.
11. Вишневская Т.И. Тестирование программного обеспечения – как учебная дисциплина. *Образовательные ресурсы и технологии*. 2014. № 1(4). С. 83-89.
12. Бураков В.В. Методика оценки качества программных средств. *Известия высших учебных заведений. Приборостроение*. 2008. Том 51, № 1. С. 35-42.
13. Уиттакер Дж., Арбон Дж., Каролло Дж. Как тестируют в Google. СПб. : Питер, 2014. 320 с.
14. Yumoto T., Matsuodani T., Tsuda K. A Test Analysis Method for Black Box Testing Using AUT and Fault Knowledge. *Procedia Computer Science*. 2013. Volume 22. P. 551-560.
15. Burns D. Selenium 1.0 Testing Tools. London : Packt Publishing, 2011. 248 p.

16. Винниченко И.В. Автоматизация процессов тестирования. СПб : Питер, 2005. 203 с.

17. Sudirman I., Govindaraju1 R., Pratiwi A. A. Information System Quality and Its Impact on Individual Users' Benefit: Analyzing the Role of Knowledge Enablers. *Jurnal Teknik Industri*. 2014. Vol. 16, No. 2. P. 65-74.

18. Ткаченко В.П., Огірко І.В., Огірко О.І. Математична модель оцінювання захисту web-сайтів. *Полиграфические, мультимедийные и WEB-технологии (PMW-2016): тези доповідей 1-ї Міжнародної науково-технічної конференції (м. Харків, 16–20 травня, 2016 р.)*. Т. 1. С. 98–101.

19. Куликов С.С. Тестирование программного обеспечения. Базовый курс. Минск : Четыре четверти, 2017. 312 с.

20. Андон Ф. И. Основы инженерии качества программных систем. Второе издание. Киев: Издательский дом «Академпериодика», 2007. 334 с.

21. Шматко О.В., Мироненко М.І. Інформаційна технологія відслідковування помилок програмного забезпечення. *Збірник наукових праць Харківського національного університету Повітряних Сил*. 2018. № 2(56). С. 120-125.

22. Степанченко И.В. Методы тестирования программного обеспечения: Учеб. Пособие. Волгоград: ВолгГТУ, 2006. 74 с.

23. Липаев В.В. Основные понятия, факторы и стандарты, определяющие качество крупномасштабных программных средств. М.-Берлин : Директ-Медиа, 2015. 237 с.

24. Орлов С.А., Цилькер Б.Я. Технологии разработки программного обеспечения. СПб : Питер, 2012. 608 с.

25. Коцюба И.Ю., Чунаев А.В., Шиков А.Н. Методы оценки и измерения характеристик информационных систем. Учебное пособие. СПб: Университет ИТМО, 2015. 264 с.

26. Myers G. J., Badget T., Sandler C. The Art of Software Testing, Third Edition. Hoboken : John Wiley & Sons, Inc, 2012. 240 p.

27. Силаков Д.В. Автоматизация тестирования Web-приложений, основанных на скриптовых языках. *Труды Института системного программирования РАН*. 2008. Том 14, № 2. С 159-178.

28. Кубашева Е.С., Гаврилов А.Г. Методика оценки качества веб-приложений. Программные системы и вычислительные методы. 2013. №1(2). С. 28-34.