Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Магістр
(назва освітнього ступеня)

на тему: Дослідження ефективності використання ручного та автоматизованого тестування з використанням мови програмування Python та Selenium

Виконав: студент VI курсу, групи СНнм-61
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

(підпис) Іващенко Є. Д. (прізвище та ініціали)

Керівник (підпис) Гром'як Р. С. (прізвище та ініціали)

Нормоконтроль (підпис) Готович В. А. (прізвище та ініціали)

Завідувач кафедри (підпис) Боднарчук І. О. (прізвище та ініціали)

Рецензент (підпис) Луцик Н. С. (прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Іващенку Євгенію Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження ефективності використання ручного та автоматизованого тестування з використанням мови програмування Python та Selenium

Керівник роботи Гром'як Роман Сильвестрович, к.ф.-м.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 24 » листопада 2023 року № 4/7-1100

2. Термін подання студентом завершеної роботи 30 травня 2024р.

3. Вихідні дані до роботи Наукові публікації щодо використання ручного та автоматизованого тестування

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Сучасні підходи до тестування програмного забезпечення. 2 Аналіз функціонального та нефункціонального тестування. 3 Оцінювання ефективності ручного та автоматизованого підходів у функціональному тестуванні. 4 Охорона праці та безпека в надзвичайних ситуаціях. Висновки. Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Сенчишин В.С., доцент		
Безпека в надзвичайних ситуаціях	Клепчик В.М., ст. викладач		

7. Дата видачі завдання 24 листопада 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	29.11.2023	*Виконано*
2.	Підбір наукових джерел щодо використання ручного	30.11.2023-8.12.2023	*Виконано*
	та автоматизованого тестування		
3.	Переклад і опрацювання наукових публікацій, збір	13.12.2023-25.01.2024	*Виконано*
	даних щодо процесів та використання ручного і		
	автоматизованого тестування		
4.	Аналіз ефективності ручного та автоматизованого	29.01.2024-16.03.2024	*Виконано*
	підходів до функціонального і нефункціонального		
5.	Оформлення розділу «Сучасні підходи до тестування	19.03.2024-3.04.2024	*Виконано*
	програмного забезпечення»		
6.	Оформлення розділу «Аналіз функціонального та	4.04.2024-15.04.2024	*Виконано*
	нефункціонального тестування»		
7.	Оформлення розділу «Оцінювання ефективності	18.04.2024-30.04.2024	*Виконано*
	ручного та автоматизованого підходів у		
	функціональному тестуванні»		
8.	Виконання завдання до підрозділу «Охорона праці»	2.05.2024-6.05.2024	*Виконано*
9.	Виконання завдання до підрозділу «Безпека в	7.05.2024-9.05.2024	*Виконано*
	надзвичайних ситуаціях»		
10.	Оформлення кваліфікаційної роботи	10.05.2024-15.05.2024	*Виконано*
11.	Нормоконтроль	16.05.2024-17.05.2024	*Виконано*
12.	Перевірка на плагіат	20.05.2024	*Виконано*
13.	Попередній захист кваліфікаційної роботи	23.05.2024	*Виконано*
14.	Захист кваліфікаційної роботи	30.05.2024	

Студент _____ Іващенко Є. Д.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Гром'як Р. С.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дослідження ефективності використання ручного та автоматизованого тестування з використанням мови програмування Python та Selenium // Кваліфікаційна робота освітнього рівня «Магістр» // Іващенко Євгеній Дмитрович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНнм-61 // Тернопіль, 2024 // С. 69, рис. – 15, табл. – 1, кресл. – 15, додат. – 2, бібліогр. – 51.

Ключові слова: тестування програмного забезпечення, тестові сценарії, інтерфейс користувача, забезпечення якості, функціонал застосунку, імітація взаємодії, виявлення дефектів.

Кваліфікаційна робота присвячена дослідженню ефективності використання ручного та автоматизованого підходів тестування ПЗ. Об'єктом дослідження є процеси ручного та автоматизованого підходів тестування ПЗ. Предметом дослідження є аналіз ефективності використання ручного та автоматизованого тестування.

В першому розділі кваліфікаційної роботи описано сфери застосування та цілі тестування. Розглянуто принципи використання різних підходів тестування. Проаналізовано сучасні засоби автоматизації тестування. В другому розділі кваліфікаційної роботи описано принципи тестування ПЗ. Досліджено класифікацію видів тестування. Проаналізовано використання різних підходів до нефункціонального тестування. Подано основні метрики для визначення ефективності тестування. В третьому розділі кваліфікаційної роботи описано розробку тест-плану. Проведено налаштування середовища автоматизованого тестування. Розроблено автоматизовані тестові сценарії та проаналізовано ефективність використання ручного та автоматизованого підходів.

ANNOTATION

Research of the efficiency of manual and automated testing with Python programming language and Selenium // The educational level "Master" qualification work // Yevhenii Dmytrovych Ivashchenko // Ternopil Ivan Pulyuy National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, SNnm-61 group // Ternopil, 2024 // P. 69, fig. - 15, tables - 1, posters - 15, annexes - 2, ref. - 51.

Key words: software testing, test scenarios, user interface, quality assurance, application functionality, interaction simulation, defect detection.

Thesis is devoted to the development of the effectiveness of using manual and automated software testing approaches. The object of research is the processes of manual and automated software testing approaches. The subject of the research is the analysis of the effectiveness of manual and automated testing.

The first chapter of the qualification work describes the scope and purpose of testing. The principles of using different testing approaches are considered. Modern test automation tools are analyzed. The second section of the qualification work describes the principles of software testing. The classification of types of testing is investigated. The use of different approaches to non-functional testing is analyzed. The main metrics for determining the effectiveness of testing are presented. The third chapter of the qualification work describes the development of a test plan. The automated testing environment is configured. Automated test scenarios are developed and the effectiveness of using manual and automated approaches is analyzed.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – Програмне забезпечення.

API (англ. Application Programming Interface) – програмний інтерфейс додатку.

CI/CD (англ. Continuous Integration/Continuous Delivery) – неперервна інтеграція/неперервна доставка.

IDE (англ. Integrated Development Environment) – інтегроване середовище розробки.

GUI (англ. Graphical User Interface) – графічний інтерфейс користувача.

QA (англ. Quality Assurance) – забезпечення якості.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасна розробка програмного забезпечення прагне до покращення швидкості та ефективності створення якісного продукту. Ручне тестування залишається важливим етапом в процесі розробки, але воно може бути обмеженим у здатності швидко виявляти та виправляти помилки. Автоматизоване тестування набуває більшого значення, оскільки воно дозволяє автоматизувати процеси, забезпечити більш швидку зміну та покращення коду, а також підвищити загальну якість програмного забезпечення. Однак, автоматизоване тестування також має свої виклики, включаючи складність підтримки скриптів та необхідність постійної адаптації до змін у програмному забезпеченні. Таким чином, проведення дослідження щодо ефективності обох методів тестування є критичним для забезпечення якості продуктів, збільшення швидкості їх впровадження та зниження загальних витрат на розробку і підтримку програмного забезпечення.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Магістр» є оцінка та порівняння ефективності обох методів тестування з метою з'ясування їх переваг та недоліків у контексті сучасного програмного забезпечення. Для досягнення поставленої мети потрібно виконати ряд завдань, зокрема:

– Проаналізувати стан досліджень щодо ручного та автоматизованого тестування, включаючи їхні призначення, переваги, обмеження та виклики.

– Дослідити засоби автоматизації програмного забезпечення, їхнє призначення, переваги та недоліки.

– Визначити ефективні підходи для виконання нефункціонального тестування.

– Розглянути метрики дослідження ефективності процесу тестування.

– Провести емпіричне дослідження шляхом створення тестових наборів для обох методів та порівняти показники ефективності.

Об’єкт дослідження. Процеси ручного та автоматизованого підходів тестування програмного забезпечення.

Предмет дослідження. Порівняльний аналіз ефективності використання ручного та автоматизованого підходів до тестування програмного забезпечення.

Наукова новизна одержаних результатів. Наукова новизна кваліфікаційної роботи полягає у оцінці передових методів тестування з метою виявлення оптимальних підходів для підвищення якості програмного забезпечення та ефективності тестування.

Практичне значення одержаних результатів. Виконано порівняльний аналіз ефективності використання підходів тестування програмного забезпечення, що сприятиме розробці оптимальної стратегії тестування для покращення якості продукту.

Апробація результатів магістерської роботи. Основні результати проведених досліджень обговорювались на VI Міжнародній студентській науково-технічній конференції «Природничі та гуманітарні науки. Актуальні питання» Тернопільського національного технічного університету імені Івана Пулюя (м. Тернопіль, 2023 р.) та VII Міжнародній студентській науково-технічній конференції «Природничі та гуманітарні науки. Актуальні питання» Тернопільського національного технічного університету імені Івана Пулюя (м. Тернопіль, 2024 р.).

Публікації. Основні результати кваліфікаційної роботи опубліковано у чотирьох працях конференції (Див. додатки А).

Структура й обсяг кваліфікаційної роботи. Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку літератури з 51 найменування та 2 додатків. Загальний обсяг кваліфікаційної роботи складає 69 сторінок, з них 46 сторінок основного тексту, який містить 15 рисунків та 1 таблицю.

1 СУЧАСНІ ПІДХОДИ ДО ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Огляд методів тестування програмного забезпечення

У сучасному світі ПЗ набуває все більшого значення, оскільки воно задіяне у будь-якому компоненті нашого життя: техніка, наука, соціальне життя тощо. Прояв цього явища відслідковується всюди, наприклад, у банківській сфері виникло поняття онлайн та мобільний банкінг, медична техніка є запрограмованою, велику частку дозвілля людей зайняли комп'ютерні ігри та соціальні мережі, без використання комп'ютерних засобів неможливо отримати більшість документів. [1]

Таким чином, забезпечення якості зайняло критично важливу ланку в процесі розробки ПЗ, оскільки користувачі очікують отримати повністю функціональний продукт в якому кожний крок заздалегідь передбачено, відсутні будь-які надокучливі дефекти та забезпечено інтуїтивне користування. Крім того, забезпечення якості важливе для бізнесу, адже необхідно впевнитись у відповідності кінцевого продукту до вимог, унеможливити зловживання системою та відповідати очікуванням користувачів. [2]

Забезпечення якості включає велику кількість різноманітних процесів, але критично важливою частиною є тестування продукту. Тестування є необхідним процесом, який виконується для наступних цілей [3]:

– Пошук дефектів: своєчасне виявлення дефектів дозволяє знизити вартість майбутнього виправлення цього дефекту, зменшити можливі погіршення метрик використання, покращити юзабіліті та створити user-friendly застосунок.

– Створення впевненості щодо якості кінцевого продукту: завдяки виконанню великої кількості тестових сценаріїв команда розуміє, що вірогідність появи критичної помилки є низькою.

– Валідація вимог: основною ціллю тестування є забезпечення перевірки відповідності поставлених вимог до реальної поведінки застосунку в різних ситуаціях.

Для тестування ПЗ використовуються методи, що дозволяють ефективно забезпечити поставлені цілі. [4] Методи тестування – це способи, за допомогою яких продукт перевіряється на відповідність заданим вимогам, специфікаціям, документації, очікуванням користувачів тощо.

Розрізняють ручний та автоматизований методи тестування.

Ручний метод тестування полягає у перевірці ПЗ, що виконується вручну людиною без використання автоматизованих засобів [5]. Таким чином, інженер з забезпечення якості власноруч виконує тестові сценарії, що дозволяє використати більш творчий підхід. Основними принципами ручного методу тестування є:

– Глибокий аналіз вимог та кінцевого продукту: завдяки розумінню взаємодії елементів ПЗ тестувальник може знайти дефекти, які виникають через зв’язки між елементами. [6]

– Індивідуальний творчий підхід: завдяки перевіркам креативних та нестандартних способів використання забезпечується більш широке покриття перевірками. Також, використовуючи техніки тестування можливе знаходження дефектів завдяки особистому досвіду тестувальника.

– Написання чіткої документації: необхідно створювати зрозумілу документацію, у якій кожний з пунктів достатньо описаний, що дозволить забезпечити відтворюваність та розуміння іншими членами команди розробки та тестування.

– Реалістичні тестові сценарії: під час процесу тестування інженер з забезпечення якості повинен розглядати продукт з точок зору професіонала та звичайного користувача, що дозволить відтворити типову поведінку та виявити дефекти, які виникають при детальному та звичайному використанню об’єкта тестування. [6]

Ручний метод передбачає виконання наступних етапів:

1. Аналіз вимог.

2. Планування.

3. Створення тест-кейсів або чеклистів.

4. Виконання тестових сценаріїв.

5. Реєстрація дефектів.

6. Звітність.

На противагу ручному методу існує автоматизоване тестування. Метод автоматизованого тестування полягає у використанні спеціалізованих програм та написанні програмних скриптів. [7]

Завдяки автоматизованому методу тестові сценарії можуть виконуватись автоматично без втручання людини та на основі цього генерувати звіт стосовно провалених перевірок. Основними принципами автоматизованого методу тестування є [8]:

– Ретельне планування тесту: планування має вирішальну роль, адже під час цього етапу визначаються засоби автоматизації, об’єм, підхід, архітектуру тощо, що значною мірою впливає на результативність автоматизованого методу тестування.

– Масштабованість: автоматизований метод повинен забезпечувати гнучкість тестових сценаріїв, оскільки необхідно відтворювати тести у різноманітних сценаріях та конфігураціях.

– Оновлення тестових сценаріїв: тестові сценарії повинні мати можливість зручного редагування, оскільки їх необхідно постійно підтримувати при зміні функціоналу, умов тощо.

Автоматизований метод передбачає виконання наступних етапів:

1. Вибір інструментів.

2. Написання тестових скриптів.

3. Налаштування тестового середовища.

4. Автоматизоване виконання тестових сценаріїв.

5. Реєстрація дефектів.

6. Звітність.

Крім того, автоматизований метод може використовуватись на різних рівнях: код, GUI, API. [9]

Автоматизація на рівні коду є автоматичними юніт-тестами, тобто тестування програмного коду, класів, змінних, функцій на предмет коректного відпрацювання.

Автоматизація на рівні GUI дозволяє тестувальнику записувати послідовність дій на інтерфейсі користувача, а саме запис прокручувань, натискань на кнопки, переходи тощо.

Автоматизація на рівні API дозволяє виконувати перевірки взаємодії через інтерфейси.

1.2 Інструменти автоматизації тестування

Існує велика кількість критеріїв до вибору інструменту автоматизації. Перш за все, від вибору інструментів залежить склад команди, витрачений час на автоматизацію, можливість відпрацювання більш комплексних сценаріїв, вартість використання, можливість задоволення певних бізнес-вимог тощо. Саме тому вибору засобів приділяється значна увага при плануванні тестової ітерації, адже цей етап значною мірою впливає на успіх продукту в цілому.

Важливим, але доволі очевидним є критерій щодо типу системи, тобто платформа та призначення цільового ПЗ (веб-додаток, мобільний застосунок, комп’ютерна гра) [10]. Більше того, від типу системи залежить не тільки вибір інструментів автоматизації, а й весь план та стратегія тестування, оскільки різні платформи мають унікальні вимоги, потреби та доступні інструменти.

Також важливим є підбір інструмента автоматизації відповідно до виду тестів: тестування API, юзабіліті, навантажувальне, UI тощо. Існує величезна кількість видів тестування, та кожний з них вимагає унікальних інструментів, оскільки універсальних практично не існує [11]. Крім того, важливим фактором є підтримка технологій, які використовуються безпосередньо у розробці ПЗ.

Вартість засобів також грає значну роль, особливо для невеликих студій та компаній, адже ціна на ліцензію деяких засобів може сягати декількох тисяч на один пристрій.

Крім того, необхідно зважати на рівень входу засобу автоматизації, оскільки розповсюдженим явищем є спеціальне навчання існуючих працівників замість підбору нового чи додаткового персоналу.

Для виконання дослідження необхідно обрати інструмент для автоматизації UI тестових сценаріїв на веб-платформі.

Найбільш популярними інструментами автоматизації UI є Selenium, Katalon, Appium, TestComplete та Cypress. [12]

На рисунку 1.1 наведено порівняння найбільш популярних засобів автоматизації UI.

Product	Katalon	Selenium	appium	TestComplete	cypress
Application Under Test	Web/API/ Mobile/Desktop	Web	Mobile (Android/iOS)	Web/Mobile/ Desktop	Web
Supported platform(s)	Windows/ macOS/ Linux	Windows/ macOS/ Linux/Solaris	Windows/ macOS	Windows	Windows/ macOS/ Linux
Setup & configuration	Easy	Coding Required	Coding Required	Easy	Coding Required
Low-code & Scripting mode	Both	Scripting Only	Scripting Only	Both	Scripting Only
Supported language(s)	Java & Groovy	Java, C#, Python, JavaScript, Ruby, PHP, Perl	Java, C#, Python, JavaScript, Ruby, PHP, Perl	JavaScript, Python, VBScript, JScript, Delphi, C++, C#	JavaScript
Advanced test reporting	✓	✗	✗	✗	✓

Рисунок 1.1 – Порівняння засобів автоматизації UI тестів

Оберемо засіб для подальшого дослідження. Порівняємо засоби Selenium, Katalon та Cypress, оскільки Appium використовується лише для мобільних застосунків, а TestComplete є доволі застарілою платформою з обмеженим функціоналом для веб-платформи.

Selenium є найбільш популярним і широко використовуваним інструментом автоматизації тестування з open-source кодом для автоматизованого кросбраузерного тестування веб-додатків. Стандартизація W3C для API WebDriver є основним архітектурним оновленням в Selenium 4. Це робить кросбраузерне тестування більш надійним та ефективним. [13]

Тестувальники можуть автоматизувати тестові сценарії використовуючи будь-яку мову програмування, з якою вони знайомі, завдяки сумісності Selenium з Java, JavaScript, Python, C# та інші.

Крім того, Selenium може допомогти з надійним, безпомилковим робочим процесом розгортання за допомогою підключення до CI/CD пайплайну.

Найбільшими перевагами Selenium є кросбраузерність, швидкість виконання, відкритий вихідний код та використання локаторів для швидкого пошуку елементів.

Cypress – це інструмент автоматизації фронтенд-тестування, яке зосереджене на виконанні наскрізного тестування. Тестувальники можуть швидко знаходити debug логування використовуючи трасування стеку програмного забезпечення то отримувати читабельні помилки за допомогою Chrome DevTools. [14]

Під час виконання тестів Cypress збирає знімки, дозволяючи користувачеві використовувати журнал команд, щоб відстежувати і вивчати, що відбувалося на кожному етапі. Щоразу, коли тест змінюється, програмне забезпечення перезавантажується. Крім того, це програмне забезпечення дозволяє користувачам переглядати команди в режимі реального часу.

Перевагами Cypress є збір знімків, автоматичне очікування, контроль інтернет-трафіку, швидкість створення тестових сценаріїв.

Katalon – це сучасна комплексна платформа управління якістю, яка допомагає командам надавати послуги за допомогою платформи зі штучним інтелектом, що дозволяє планувати, створювати та виконувати автоматизовані тести. [15]

Katalon Studio – це ефективний інструмент автоматизації для тестування десктопних, мобільних та онлайн API. Інтеграція всіх необхідних тестових

компонентів з шаблонами проектів і вбудованими ключовими словами забезпечує низьких рівень входу. Крім того, Katalon представляє тестові дані та результати виконання за допомогою звітів, діаграм та графіків. Також він включає в себе широкий спектр функціональних можливостей, в тому числі ретельний підхід до автоматизації, який враховує різні платформи і типи тестування.

Перевагами Katalon є легке використання, підтримка різних платформ, використання штучного інтелекту та вбудовані засоби CI/CD.

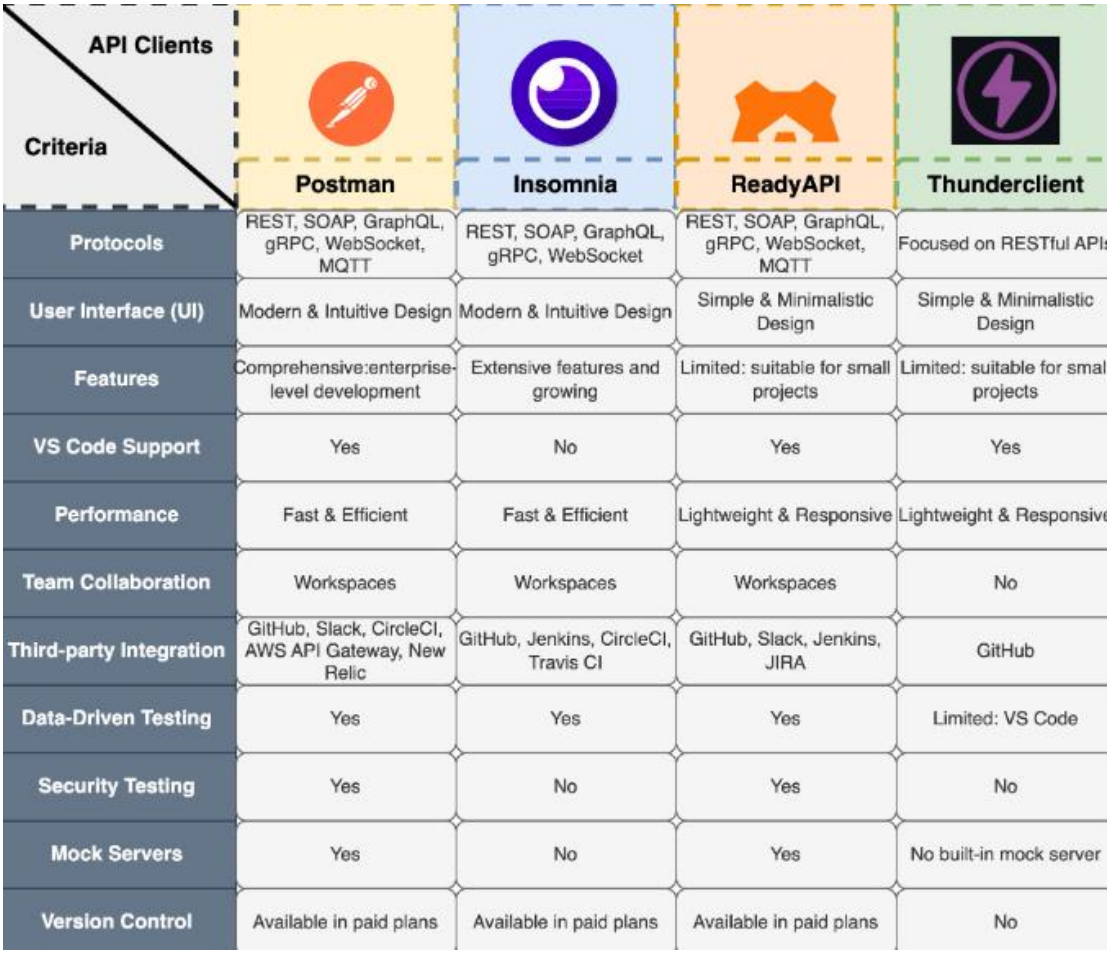
Після ретельного аналізу було вирішено, що для подальшого дослідження буде використано засіб Selenium, оскільки саме цей засіб дозволяє створювати тестові сценарії будь-якої складності.

Також одним з найбільш затребуваних видів є тестування API. Тестування API аналізує інтерфейс прикладної програми, щоб перевірити, чи відповідає він очікуваній функціональності, безпеці, продуктивності та надійності [16]. Тести виконуються або безпосередньо на API, або в рамках інтеграційного тестування.

На відміну від тестування інтерфейсу користувача UI, яке фокусується на перевірці зовнішнього вигляду програми, тестування API фокусується на аналізі бізнес-логіки програми, а також реакції безпеки і даних. API-тест зазвичай виконується шляхом надсилання запитів до однієї або декількох кінцевих точок API та порівняння відповідей з очікуваними результатами.

Порівняємо найбільш популярні засоби автоматизації тестування API стосовно використовуваних протоколів, функціоналу, тестування безпеки та інших. [17]

Порівняння інструментів автоматизації тестування API Postman, Insomnia, ReadyAPI та Thunderclient зображено на рисунку 1.2.



API Clients / Criteria	Postman	Insomnia	ReadyAPI	Thunderclient
Protocols	REST, SOAP, GraphQL, gRPC, WebSocket, MQTT	REST, SOAP, GraphQL, gRPC, WebSocket	REST, SOAP, GraphQL, gRPC, WebSocket, MQTT	Focused on RESTful APIs
User Interface (UI)	Modern & Intuitive Design	Modern & Intuitive Design	Simple & Minimalistic Design	Simple & Minimalistic Design
Features	Comprehensive:enterprise-level development	Extensive features and growing	Limited: suitable for small projects	Limited: suitable for small projects
VS Code Support	Yes	No	Yes	Yes
Performance	Fast & Efficient	Fast & Efficient	Lightweight & Responsive	Lightweight & Responsive
Team Collaboration	Workspaces	Workspaces	Workspaces	No
Third-party Integration	GitHub, Slack, CircleCI, AWS API Gateway, New Relic	GitHub, Jenkins, CircleCI, Travis CI	GitHub, Slack, Jenkins, JIRA	GitHub
Data-Driven Testing	Yes	Yes	Yes	Limited: VS Code
Security Testing	Yes	No	Yes	No
Mock Servers	Yes	No	Yes	No built-in mock server
Version Control	Available in paid plans	Available in paid plans	Available in paid plans	No

Рисунок 1.2 – Порівняння інструментів автоматизації API

Існує значна кількість інструментів для автоматизованого тестування, які використовуються для виконання як одного, так і різних видів тестування. Таким чином, підбір вірного засобу автоматизації грає значну роль, оскільки кожний з них володіє власними перевагами та недоліками, функціоналом та переліком підтримуваних технологій.

1.3 Особливості автоматизації тестування шляхом написання програмного коду

Існують 2 основних шляхи автоматизації тестових сценаріїв UI: живий «запис» дій користувача, який можна відтворити шляхом автоматизації та написання програмного коду, який буде імітувати взаємодію з браузером через драйвер. [18]

Живий запис дій користувача є доволі зручним способом автоматизації, адже він не вимагає значних знань у програмуванні та має низький рівень входу. Недоліком цього способу є те, що більш складні сценарії таким чином відтворити не вийде. Таким чином, автоматизація за допомогою програмування є більш пріоритетним способом, адже забезпечує більш широке покриття.

Як і будь яке програмування, такий спосіб автоматизації має власні особливості.

Метою автоматизованих тестів є спроба перевірити, чи працює програмне забезпечення відповідно очікуваному результату зараз і в майбутньому [19]. Очікуваними результатами можуть бути [20]:

– Виклик функції з відомими вхідними даними і перевірка очікуваного результату.

– Створення тестового оточення та перевірка того, що елементи разом з усіма системами, які стоять за ними, можуть правильно виконувати прості операції.

– Запуск системи з великою кількістю випадкових вхідних даних.

Важливим поняттям є інтеграційні та юніт тести, оскільки вони є основою автоматизованого тестування за допомогою коду. [21]

Інтеграційне тестування – це етап тестування ПЗ, на якому тестується весь програмний модуль або, якщо він складається з декількох програмних модулів, вони об'єднуються, а потім тестуються як група.

Юніт тестування – це тестування відносно невеликої кількості коду, який можна ізолювати від решти кодової бази, яка може бути великою і складною системою.

Таким чином, постає питання, які тести необхідно писати: інтеграційні чи юніт? Наприклад, юніт тести є більш швидкими, оскільки охоплюють меншу частину коду, є більш надійними та вимагають менше налаштування. З іншого боку, юніт тести охоплюють лише частину залежностей та лише невелику частку системи.

Отже, принципи вигляду «писати тільки модульні тести, а не інтеграційні» або «писати тільки інтеграційні тести, а не модульні» не сприяють повноцінному тестуванню застосунку.

Якщо кожна окрема функція ретельно протестована, але модуль в цілому неправильно комбінує всі функції, або якщо кожен окремий модуль ретельно протестований, але загальний процес ПЗ використовує модулі неправильно, то це є значною проблемою, оскільки такі юніт тести не можуть виявити помилки, що виникають на верхніх рівнях ієрархії.

З іншого боку, не варто використовувати лише інтеграційні тести. В теорії такий підхід може працювати, але внаслідок цього буде витрачатись надмірна кількість обчислювальних ресурсів та часу.

Таким чином, структура автоматизованих тестів повинна «відзеркалювати» структуру ПЗ. Необхідно використовувати тести на всіх рівнях, пропорційно до кількості коду та ймовірності/серйозності виникнення помилки. [22]

Також необхідно вірно пріоритезувати кількість необхідних автоматизованих тестів для певного функціоналу системи. Необхідно керуватись наступними правилами:

– Важливі речі потребують додаткового тестування. Наприклад, у соціальних мережах необхідно приділити значну увагу системі авторизації, фінансові застосунки повинні бути ретельно протестованими на предмет безпеки тощо.

– Менш важливі речі, можуть потребувати незначного тестування або взагалі не потребувати автоматизації.

– Функціонал в стадії активної розробки потребує більше тестів, в свою чергу на застарілому функціоналі можна зосередити менше уваги.

– Важливо зосередити більше зусиль на тестуванні стабільних елементів системи, а не на тестуванні нестабільного коду, який може повністю зникнути через тиждень.

Крім того, автоматизовані тестові сценарії все ще залишаються кодом, тому необхідно ставитись до цього як до будь-якого іншого програмного коду.

Тобто, необхідно дотримуватись стандартів якості коду, які відносяться й до звичайного коду: правильна назва змінних, форматування, коментарі, вбудована документація та зрозуміла організація коду. Також програмний код необхідно рефакторити для забезпечення легко читання та розуміння функціоналу.

Важливим принципом створення коду автоматизації тестових сценаріїв є забезпечення гнучкості та адаптивності [23]. Наприклад, деякі API можуть швидко змінюватись, тому зміна відповідного автоматизованого тесту повинна бути швидкою.

Однією з найважливіших технік є використання одинарних та масових тестів [24]. Одинарні тести – це тести, які проганяють певне поле вводу на одному (або невеликій кількості) прикладів значень з ретельними перевірками. З іншого боку, об'ємні тести – це тести, які проганяють через велику кількість прикладів з менш ретельною перевіркою поведінки кожного з них. Нечітке тестування або тестування на основі властивостей є двома поширеними підходами в цій категорії.

Одинарні тести є зручною документацією. Будь-яка людина, яка прочитає декілька прикладів одинарних тестів відразу зможе зрозуміти що робить модуль і як він буде використовуватися. Такий вид тестових даних ідеально використовується для покриття «очікуваних» варіантів результатів функціоналу. Однак їх недостатньо для покриття «неочікуваних» випадків. Саме для таких випадків використовуються масові тести.

Масові тести перевіряють набагато більше випадків, ніж тестувальник може охопити за допомогою ручного тестування. Замість послідовного запуску системи та перевірки декількох значень, масові тести запускають код з сотнями і тисячами різних вхідних даних. Це дозволяє охопити несподівані випадки, які тестувальник ніколи б не відтворив вручну або за допомогою одинарних тестів.

Також необхідно завжди пам’ятати про вартість автоматизованих тестів. Будь-який тест не є безкоштовним, адже його повинен хтось написати, а кожен тест впливає на загальну вартість кожного прогону наборів автоматизованих тестів [25]. Кожний новий тест зробить прогін тестового набору повільнішим, та

кожен з них потенційно необхідно підтримувати у випадках зміни відповідного функціоналу.

Деякі автоматизовані тести можуть бути марними, якщо вони займають багато часу, не надійні, складні в обслуговуванні або охоплюють речі, які не є пріоритетними для тестування.

1.4 Порівняння ручного та автоматизованого тестування

Ретельний вибір між автоматизованим та ручним тестуванням є ключовим аспектом ефективного процесу тестування програмного забезпечення. Обидва підходи мають свої переваги і обмеження, і правильний вибір залежить від конкретних потреб проекту [26]. При неправильному виборі способів, методів та співвідношень кількості тестувальників будь-який проєкт ризикує втратити значну частину ресурсів.

Ручне тестування має велику кількість переваг, але на противагу існує багато обмежень та недоліків цього способу [27]. Перевагами ручного тестування є:

– Низький вхідний рівень. Зазвичай, ручним тестувальникам не потрібно освоювати складні технології та поглиблюватись у певну мову програмування.

– Гнучкість. Команди можуть легко підлаштовуватись під різноманітні ситуації, склад команди тощо.

– Низькі витрати часу. Ручним тестувальникам необхідно невелику кількість часу для ознайомлення з проєктом та поверхневого тестування.

– Можливість креативного підходу. Тестувальники можуть застосовувати своє уявлення щодо системи та перевіряти складні нестандарті ситуації.

– Легка робота над проблемами. Відсутні потенційні проблеми з мовами програмування та бібліотеками, тому для вирішення проблемних місць достатньо лише обговорення.

– Робота з нестандартними випадками. Ручним тестувальникам не потрібно вивчати складну технологію при появі нестандартних випадків або ситуацій, коли автоматизація не ефективна або занадто дорога.

Недоліками ручного тестування є [27]:

– Витрачений час. Для виконання регресійного чи смокового тестування необхідно витратити значну кількість часу, оскільки рутинні тестові сценарії не автоматизовуються.

– Людський фактор. Завжди необхідно враховувати можливість пропуску дефектів через неуважність, «парадокс пестициду», втомленість.

– Труднощі зі складними системами. Великі системи та значні обсяги даних важно опрацювати ручним способом, оскільки існує величезна кількість можливих ситуацій.

– Керування великими командами. Управління командами багатьох тестувальників є складним завданням, оскільки необхідно контролювати результат кожного члена команди.

– Невиправдані очікування. При тестуванні однотипних ситуацій рідко знаходяться дефекти, тому виникає відчуття слабкого впливу на загальний результат.

Таким чином, ручне тестування може бути доволі ефективним, але для цього необхідно правильно опрацювати мету тестування, скласти детальний план тестування, підібрати інструменти тощо.

Розглянемо переваги автоматизованого тестування [28]:

– Ефективність. Зазвичай, автоматизовані тести є значно ефективніші у ситуаціях, коли необхідно повторно виконувати тестові сценарії. При цьому, регресійне тестування займає в рази менше часу.

– Відсутність людського фактору. Автоматизований тестовий сценарій не має чіткі критерії, тому помилки практично неможливі. Але, помилки можуть виникнути на етапі програмування критеріїв.

– Масштабованість. До існуючих тестових наборів доволі легко додаються нові тестові сценарії.

– Метрики. Використовуючи спеціальні інструменти є можливість легко використати різноманітні метрики, наприклад відсоток покриття програмного коду автоматизованими тестами, результативність прогонів тестів та динаміка знайдених дефектів.

– Інтеграція CI/CD. Автоматизовані тести можна інтегрувати у CI/CD систему, що дозволить зручно налаштовувати критерії прогону різних тестових наборів. [29]

Наведемо недоліки автоматизації тестування:

– Витрати. Автоматизація тестування вимагає значних вкладень у інструменти та персонал, а також автоматизація зазвичай займає значно більше часу у порівнянні з ручним тестуванням.

– Проблеми з гнучкістю. Доволі складно обслуговувати існуючі автоматизовані тести при зміні вимог чи функціоналу.

– Обмеженість можливостей автоматизації. Існує велика кількість видів тестів, для яких необхідне вузькоспеціалізоване ПЗ, погляд реальної людини або випадки, які складно автоматизувати.

– Пропуск дефектів. Автоматизовані тести можуть спіймати лише дефект, який вони очікують, при цьому нестандартні випадки не відслідковуються.

Отже, автоматизоване тестування може бути корисним інструментом, але існують ситуації, коли автоматизація недоцільна, наприклад, короткострокові проєкти. Крім того, деякі види ПЗ доволі складно автоматизувати, оскільки вони є доволі специфічні або мають нестандартну взаємодію з користувачем. Наприклад, комп’ютерні ігри (крім ігор, які працюють на веб-сторінці) практично неможливо автоматизувати, оскільки кожна гра може мати особливу форму взаємодії з користувачем, а розробка автоматизованих інструментів під конкретну гру є надзвичайно дорогою. Зазвичай, у комп’ютерних іграх автоматизуються зовнішні системи (наприклад, API), а основний функціонал тестується ручним способом.

На рисунку 1.3 зображено порівняльну таблицю основних властивостей ручного та автоматизованого тестування.

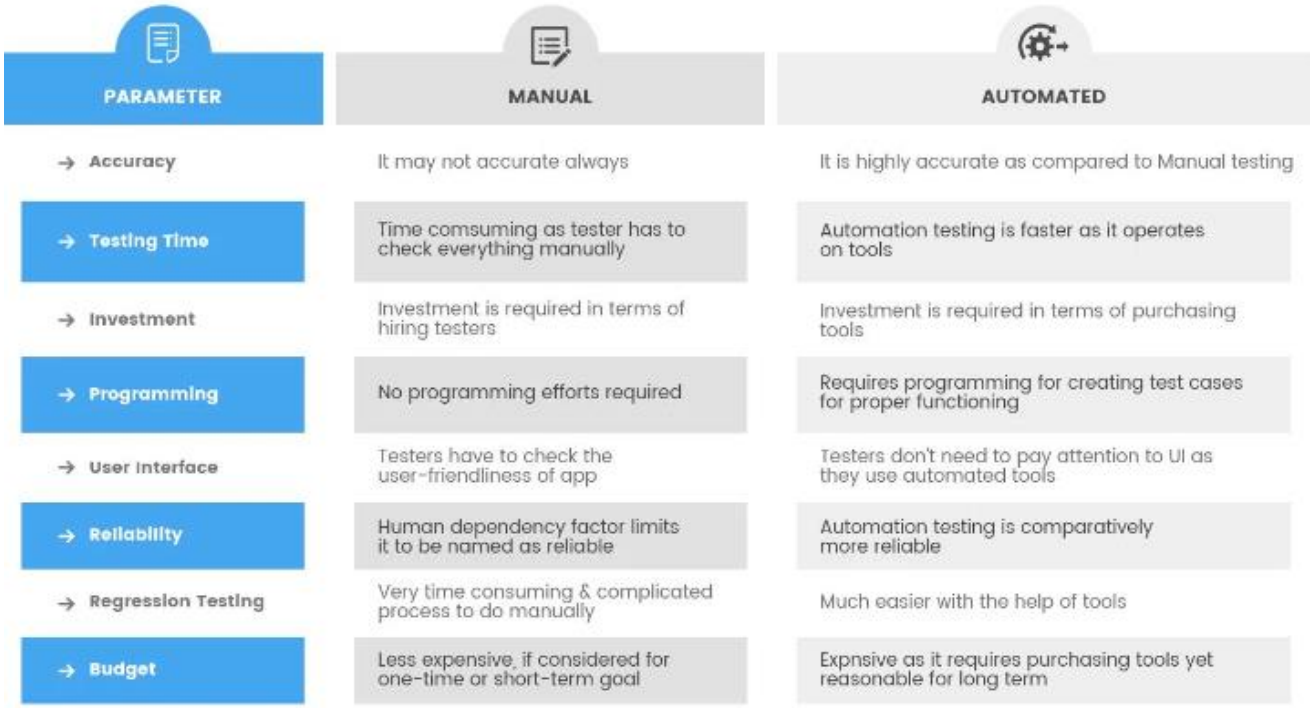


Рисунок 1.3 – Порівняння ручного та автоматизованого тестування

Таким чином, автоматизація тестових сценаріїв є найбільш ефективною у рутинних ситуаціях, важкодоступних місцях, функціональності з високими ризиками, наскрізних перевірках та тестах зі складними математичними операціями.

1.5 Висновок до першого розділу

В першому розділі кваліфікаційної роботи освітнього рівня «Магістр» наведено сфери застосування та визначено головні цілі тестування. Крім того, визначено основні принципи використання ручного та автоматизованого методів тестування, особливості застосування автоматизованого тестування з використанням програмного коду та порівняно переваги та недоліки обох методів тестування.

Крім того, в першому розділі було оглянуто найбільш сучасні інструменти та засоби автоматизації тестування, а саме Katalon, Selenium, Cypress і Appium для тестування UI та Postman, Insomnia, ReadyAPI, Thunderclient для тестування API. На основі порівняння засобів для тестування UI було обрано Selenium для

подальшого дослідження ефективності ручного та автоматизованого тестування, оскільки саме цей засіб дозволяє створювати тестові сценарії будь-якої складності, підтримує значну кількість різних мов програмування та забезпечує інтеграцію CI/CD процесу для забезпечення швидкого застосування тестових наборів.

2 АНАЛІЗ ФУНКЦІОНАЛЬНОГО ТА НЕФУНКЦІОНАЛЬНОГО ТЕСТУВАННЯ

2.1 Принципи тестування програмного забезпечення

Тестування ПЗ спрямоване на виявлення дефектів і помилок, що допомагає підвищити якість продукту і підтримує його надійність після випуску. Для ефективного тестування були визначені принципи, за допомогою яких підтримується висока продуктивність, формуються стратегії та в деякій мірі визначається поле відповідальності.

Принципи є особливими вказівками, які дозволяють покращити процес тестування. Детально розглянемо дані 7 принципів: [30]

1. Тестування вказує на наявність дефектів, а не їх відсутність. Надзвичайно складною задачею є знайти щось, про що взагалі невідомо, ні розташування, ні приблизного опису, ні навіть відомостей про існування дефекту. Оскільки фізично неможливо перевірити поведінку складного програмного продукту у всіх можливих ситуаціях і умовах, тестування не може гарантувати, що в певній ситуації, за певних обставин не виникне дефект. Для запобігання цього тестування використовує величезний набір методів, підходів, інструментів та рішень для пошуку найбільш ймовірних, найбільш критичних ситуацій і виявлення дефектів. Саме такі дефекти будуть усунуті, що значно підвищить якість продукту, але все одно не гарантує відсутність проблем у решті неперевірених ситуацій і умовах.

2. Вичерпне тестування неможливе. Вичерпне тестування означає тестування ПЗ використовуючи всі можливі комбінації стосовно всіх вхідних вимог, даних та конфігурацій, що фактично неможливо виконати . Навіть для одного простого поля введення імені користувача може існувати близько 32 позитивних перевірок (на кількість символів, наявність цифр, літер, спеціальних значень тощо) і нескінченна кількість негативних перевірок (наприклад, перевірка використання 99999 символів при максимальному обмеженні у 40). Саме тому немає можливості протестувати програмний продукт повністю. Однак

це не означає, що не є ефективним. Ретельний аналіз вимог, оцінка ризиків, розстановка пріоритетів, аналіз предметної області, моделювання, робота з кінцевими користувачами, використання спеціальних методик тестування дозволяють тестувальникам виявити тестові сценарії, які потребують особливої уваги.

3. Раннє тестування є більш ефективним. Раннє тестування дозволяє знайти критичні дефекти та проблемі зони на початкових стадіях, що дозволить зменшити витрати на їхнє виправлення, оскільки вони ще не перетворились на фундаментальні та системні частини ПЗ. Крім того, таке тестування дозволяє зробити зміни в проекті ще на ранніх стадіях розробки, коли це менш витратно і затрати часу на це найменші. [31]

4. Кластеризація дефектів. Дефекти виникають внаслідок певних дій команди розробки, бізнес-аналізу, дизайну тощо. Наприклад, використовується нова або складна технологія, ПЗ доводиться працювати в несприятливих умовах або взаємодіяти із зовнішніми ненадійними компонентами. Крім того, може виникнути ситуація, що частина технічної документації не було досліджено належним чином, або певну частину програми реалізовували недостатньо компетентні люди. Таким чином, дефекти мають схильність до групування у певних частинах функціоналу. При виникненні подібної ситуації найбільш вірним є рішення продовжити дослідження цієї ділянки програмного продукту.

5. Парадокс пестициду. Назва цього принципу походить від відомого явища в сільському господарстві: якщо тривалий час обробляти сільськогосподарські культури одним пестицидом, то у комах незабаром виробляється імунітет, що робить пестицид неефективним. Те саме стосується і тестування ПЗ, де парадокс пестицидів проявляється при багаторазовому повторенні однакових тестових сценаріїв. З часом таке тестові сценарії знаходити нові дефекти. Щоб подолати парадокс пестицидів, необхідно регулярно переглядати та оновлювати тест-кейси, урізноманітнювати підходи до тестування та застосовувати різні техніки тестування.

6. Тестування залежить від контексту. Програмні продукти можуть належати до різних предметних галузей, бути побудовані за різними

технологіями предметних областей, можуть бути побудовані з використанням різних технологій, можуть використовуватися у сферах з підвищеним ризиком (медицина, військова справа) тощо. Дані особливості середовища впливають на те, як повинен бути організований процес тестування. Набір характеристик програмного продукту впливає на ретельність тестування, набір використовуваних методик та інструментів, принципи організації роботи тестувальників, що забезпечить найбільш ефективну та ретельну перевірку цільового ПЗ.

7. Неможливо гарантувати відсутність дефектів. Продукт повинен бути не лише без дефектів, але й задовольняти вимоги замовника і кінцевих користувачів, інакше він стане непридатним для використання. Часто порушення цього принципу полягає в недостатній розробці та реалізації нефункціональних вимог до продукту, що спричиняє незадоволеність з боку кінцевих користувачів і загальне зниження метрик застосунку.

Отже, розуміння контексту продукту, потреб користувачів, запобігання парадоксу пестициду та дотримання інших принципів дозволяє тестувальникам обрати найкращу стратегію, оптимізувати процес тестування ПЗ та в загальному покращувати ефективність.

2.2 Класифікація видів тестування

Впровадження програмного забезпечення вимагає систематичного й комплексного тестування для забезпечення високої якості продукту та відповідності його функціональних характеристик вимогам користувачів. Однак, розмаїття методів та підходів до тестування вимагає чіткої класифікації для кращого розуміння й використання цих методів у практичних задачах розробки програмного забезпечення. [32]

Необхідно систематизувати види тестування на основі різних критеріїв, таких як рівень, об’єкт, за доступом до інформації тощо. Класифікація є ключовим елементом для усвідомлення їх сутності, особливостей і областей застосування. [33]

Перш за все, найбільш об’ємною та детальною є класифікація тестування за об’єктом. Вид тестування за об’єктом поділяється на функціональне та нефункціональне тестування.

Функціональне тестування поділяється на:

– Власне функціональне тестування. Перевірка прямих функціональних вимог.

– Тестування безпеки. Відноситься до функціонального тестування, у випадку якщо безпека є невід’ємною частиною продукту, наприклад, у банківській сфері.

– Тестування інтерфейсу користувача. Перевірка візуальної функціональної складової системи.

Нефункціональне тестування поділяється на велику кількість підвидів: [34]

– Тестування документації. Вичитка технічної документації на предмет логічності, відсутності несумісних вимог, потреби у реалізації тощо.

– Тестування юзабіліті. Перевірка зручності використання з точки зору користувача.

– Тестування навантаження. Оцінка роботи системи при очікуваному середньому навантаженні.

– Стресове тестування. Навантаження системи понад очікувані пікові значення.

– Тестування стабільності. Перевірка звичайного навантаження системи на тривалому часовому проміжку.

– Тестування масштабованості. Перевірка можливості залучення додаткових обчислювальних ресурсів у випадку регулярної перенавантаженості системи.

– Тестування безпеки. Перевірка захищеності систем авторизації, доступу до адміністративної панелі тощо.

– Тестування інсталяції. Встановлення, оновлення та видалення застосунку на цільових платформах різної конфігурації.

– Тестування інтерфейсу. Перевірка непрямих вимог до інтерфейсу користувача.

– Тестування локалізації. Перевірка адаптації продукту до різних регіональних особливостей.

– Тестування інтернаціоналізації. Необхідно впевнитись, що продукт підтримує різні міжнародні налаштування з точки зору програмної реалізації (можливість вводу тексту справа наліво, адаптація під ієрогліфи та інші спеціальні символи тощо).

– Тестування сумісності. Сумісність програмного продукту до середовищ з різними конфігураціями.

– Тестування відмов та відновлень. Перевірка можливості використання резервної копії при непередбачуваних ситуаціях.

На рисунку 2.1 зображено візуалізацію розширеної класифікації виду тестування за об’єктом у вигляді дерева.

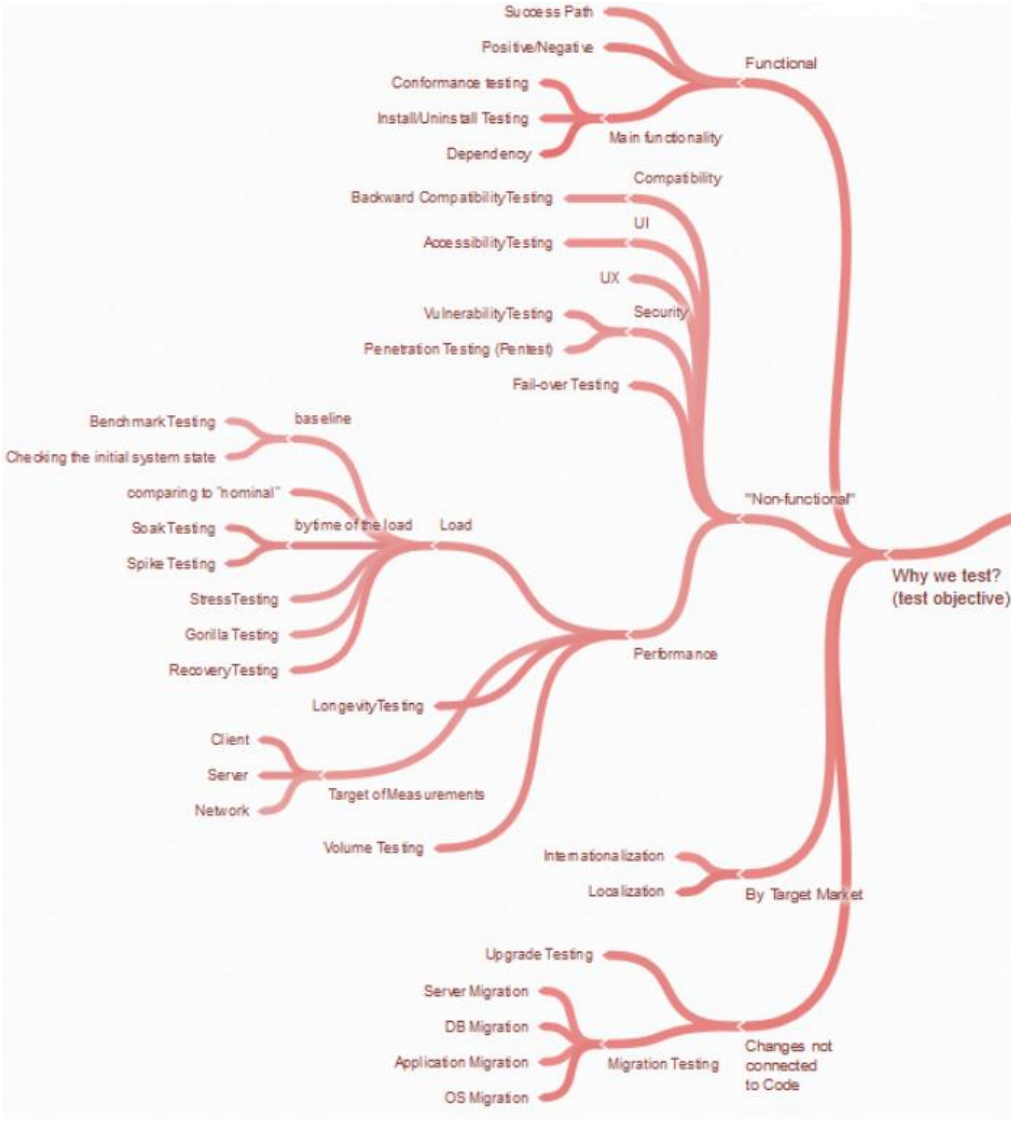


Рисунок 2.1 – Класифікація виду тестування за об’єктом

Наведемо класифікацію виду тестування за змінами: [35]

– Ретест. Повторна перевірка виправленого дефекту для підтвердження факту його відсутності у новій версії.

– Смокове тестування. Перевірка критичного функціоналу.

– Регресійне тестування. Повторна перевірка старого функціоналу.

– Санітарне тестування. Перевірка частини системи або функціоналу в залежності від контексту.

Класифікація виду тестування за рівнем поділяється на:

– Модульне тестування. Перевірка малого модуля системи, невеликої частини програмного коду.

– Тестування інтеграції. Перевірка зв’язків між різними модулями.

– Системне тестування. Повний набір тестування всіх компонентів системи в цілому.

– Приймальне тестування. Виділення та тестування деяких тестових сценаріїв, наприклад, для презентації роботи функціоналу команді.

Крім того, тестування можна класифікувати за ступенем доступу до інформації: [36]

– Чорна скринька. Ситуація, коли тестувальник не має додаткових відомостей про систему, лише робочий продукт.

– Сіра скринька. Тестувальник має частковий доступ до технічної частини або документації (наприклад, доступ до бази даних в режимі «лише для перегляду»).

– Біла скринька. Повний доступ до внутрішньої інформації.

Для подальшого дослідження буде використано класифікацію виду тестування за об’єктом, тобто функціональне та нефункціональне тестування, оскільки даний вид найбільш детально описує ціль та спосіб перевірки тестових сценаріїв.

2.3 Використання ручного та автоматизованого підходів для нефункціонального тестування

Нефункціональне тестування програмного забезпечення є критичним аспектом процесу розробки, спрямованим на оцінку характеристик системи, що не стосуються її функціональності. Воно визначається як тестування аспектів системи, таких як продуктивність, надійність, безпека, сумісність та інші. Для ефективного виконання нефункціонального тестування можна використовувати як ручні, так і автоматизовані підходи.

Загалом, перевірка нефункціональних критеріїв має співставну важливість, як і функціональне тестування. Вони є більш спрямованими на перевірку загальних характеристик системи, а не функціональних вимог.

Метою нефункціонального тестування є перевірка відповідності продукту очікуванням користувачів та оптимізація продукту перед випуском. [37]

Визначимо найбільш ефективний спосіб тестування кожного нефункціонального підвиду, а саме за допомогою порівняння доцільності використання ручного, автоматизованого чи комбінованого підходу.

Перш за все, необхідно визначити, якими є перевагами ручного та автоматизованого тестування з точки зору нефункціонального тестування.

Ручне тестування має значну перевагу в контексті креативності перевірок, а також у сприйнятті нестандартних ситуацій та дефектів, оцінці візуального сприйняття інтерфейсу користувача. Крім того, ручне тестування є значно більш гнучким та дозволяє більш глибоко розуміти контекст об’єкту тестування. У деяких випадках, ручне тестування є значно дешевшим у порівнянні з автоматизованим.

Завдяки автоматизованому тестуванню можливо забезпечити значно швидше проходження тестових сценаріїв, але значним недоліком є тривалість творення такого тесту, тому доцільно автоматизувати тести, які будуть виконуватись значну кількість разів. Якщо не враховувати підтримку тестів, то деякі з них можуть виконуватись десятки або до сотні разів без додаткових витрат (крім залучення дорогих обчислювальних ресурсів). Таким чином,

автоматизація тестів нестабільних систем, або систем, до яких швидко змінюються вимоги, є не ефективним.

Розглянемо підвиди нефункціонального тестування для визначення найбільш ефективного способу тестування.

Тестування документації зазвичай виконується шляхом вичитки всіх необхідних видів документів: специфікації до продукту, FAQ, інструкції користувача. Таке тестування доцільно проводити лише ручним способом, оскільки значну роль грає пошук помилок, невідповідностей тощо.

Тестування юзабіліті фактично неможливо виконати автоматизованим способом, оскільки важливо розглянути систему з точки зору користувача, інтуїтивної навігації, приємного візуального сприйняття та наявності необхідних UI елементів.

Тестування навантаженості є тестом, який імітує поведінку кількох користувачів, які одночасно отримують доступ до системи. Таким чином, автоматизований спосіб є більш ефективним, оскільки дозволяє генерувати дані більш точно та зі значно вищими об’ємами даних. Найбільш популярним засобом автоматизації тестування навантаженості є JMeter, ПЗ з відкритим програмним кодом. Цей засіб зарекомендував себе завдяки широкому спектру можливих сценаріїв використання: тестування продуктивності, навантаженості, стрес-тестування, тестування відновлюваності. Крім того, цей засіб має зручний та зрозумілий графічний інтерфейс, підтримку великої кількості протоколів, кросплатформеність, багатопотокову роботу та наявність якісних навчальних посібників.

На рисунку 2.2 зображено інтерфейс застосунку JMeter.

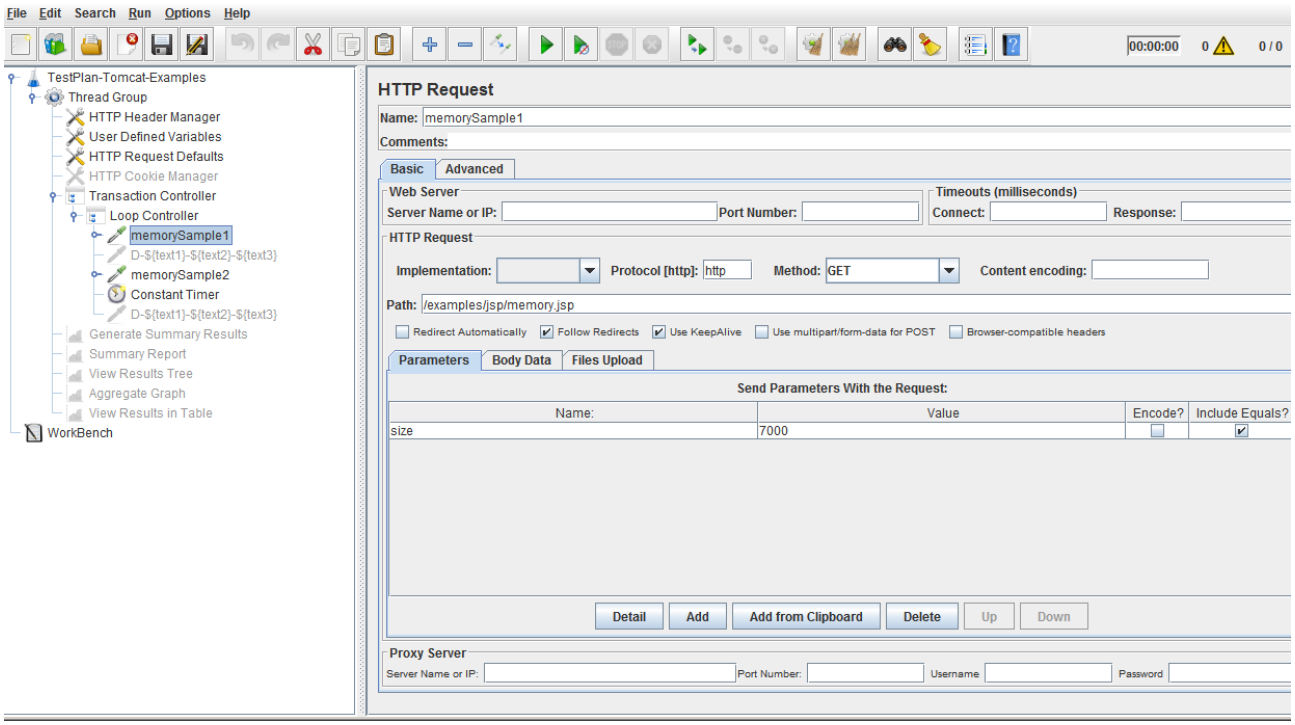


Рисунок 2.2 – Графічний інтерфейс засобу JMeter

Тестування масштабованості виконується для перевірки можливості залучення додаткових обчислювальних ресурсів при високій завантаженості. Таким чином, для виконання тесту необхідно навантажити продукт за допомогою автоматизованого засобу. На рисунку 2.3 зображено графічний інтерфейс засобу BlazeMeter, який використовується для тестів масштабованості та стабільності.

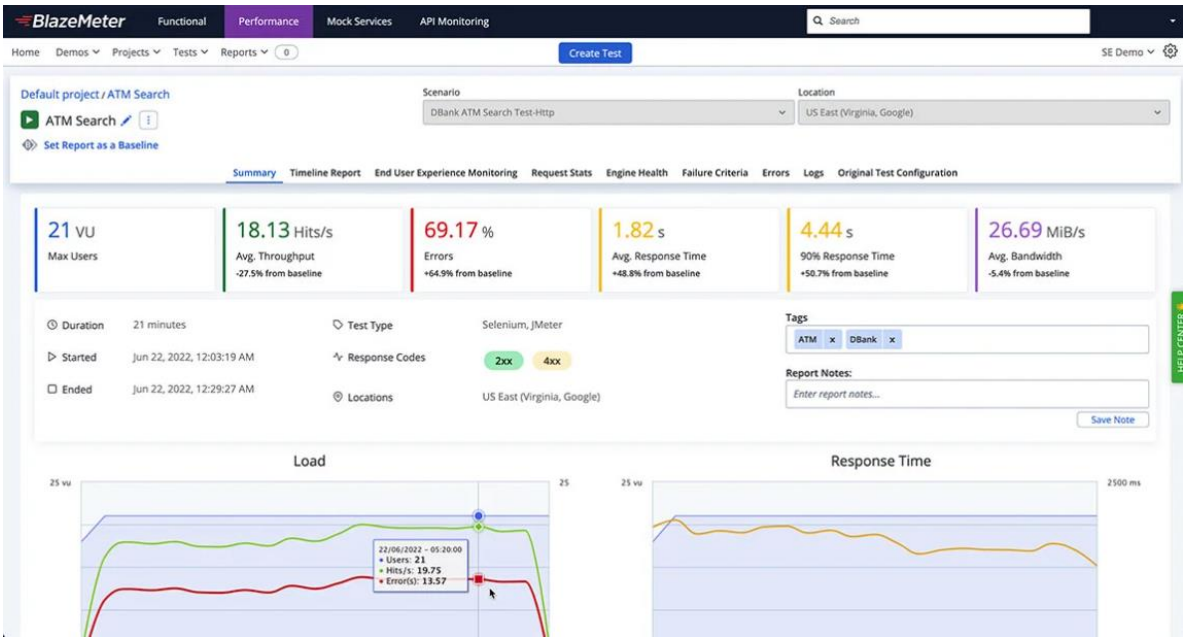


Рисунок 2.3 – Графічний інтерфейс засобу BlazeMeter

Тестування безпеки є доволі специфічним підвидом тестування, оскільки його застосування значною мірою залежить від платформи та особливості продукту. Зазвичай, для тестування безпеки використовують комбінований підхід: ручний та автоматизований. За допомогою ручного тестування перевіряють бізнес-логіку на предмет прогалин у безпеці, використання специфічних XSS та SQL ін'єкцій, аутентифікацію тощо [38]. Автоматизовані засоби дозволяють знайти вразливості шляхом перебору, аналізу протоколів, HTTP запитів та інших. Зручним та доволі популярним засобом автоматизованого тестування безпеки веб-застосунку є Nessus, який може виявляти вразливості в мережах, системах і веб-додатках.

На рисунку 2.4 зображено графічний інтерфейс застосунку Nessus.

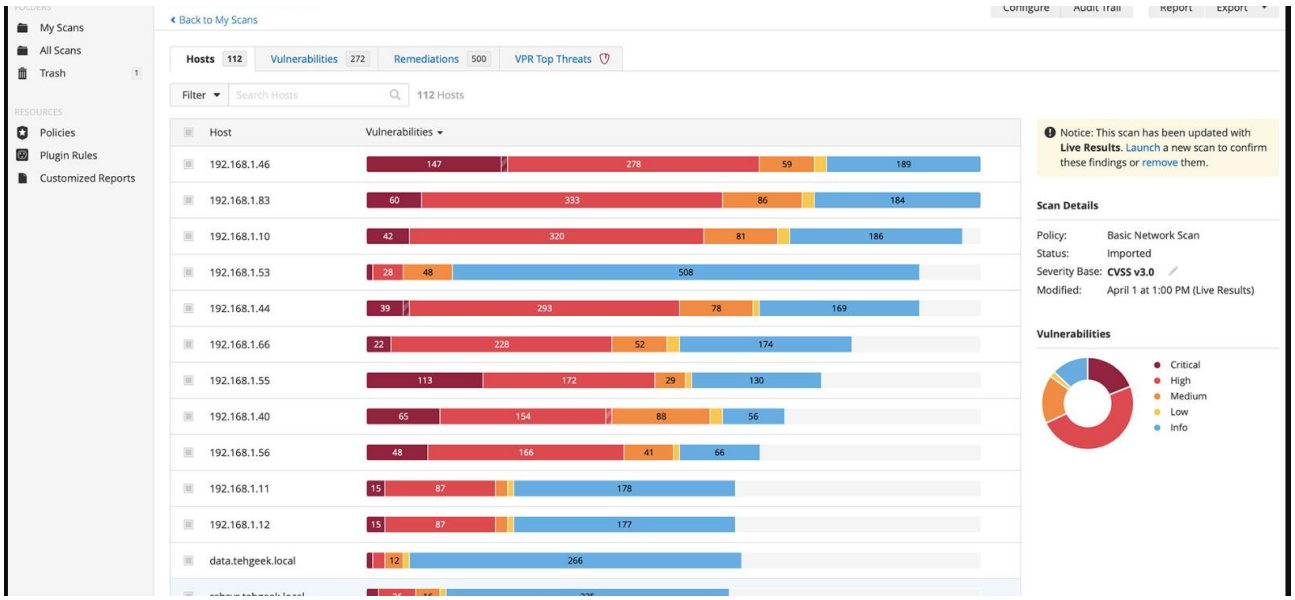


Рисунок 2.4 – Графічний інтерфейс засобу Nessus

Тестування інтерфейсу, локалізації та інтернаціоналізації зазвичай виконується ручним підходом, оскільки такі тести вимагають сприйняття об'єктів людиною, що дозволить ідентифікувати дефекти та потенційні вузькі місця.

2.4 Методологія оцінки ефективності тестування

Необхідно визначити, який з підходів тестування є більш ефективним для функціонального тестування. Для цього необхідно проаналізувати, які метрики та дані використовуються для оцінки ефективності ручного та автоматизованого тестування.

Існує декілька типів метрик для визначення ефективності тестування продукту: [39]

– Проєктні метрики. Визначають загальну якість продукту для оцінки витрачених ресурсів у співвідношенні до результатів.

– Продуктові метрики. Визначають загальну складність продукту, дизайн, продуктивність тощо.

– Метрики тестового покриття. Оцінка міри покриття ПЗ тестовими сценаріями.

– Метрики дефектів. Відстеження кількості та серйозності знайдених дефектів.

– Метрики виконаних тестів. Визначення загальної ефективності та результативності тестування.

– Метрики тест-дизайну. Оцінка якості дизайну тестової документації відповідно до кількості тестових сценаріїв та покриття.

Для визначення ефективності автоматизованого та ручного тестування зазвичай використовуються різні метрики, оскільки ці підходи мають різне технічне підґрунтя та практичне застосування.

Метрики ручного тестування дозволяють визначити якість та ефективність процесу забезпечення якості. В якості основних метрик ручного тестування використовується:

– Час на виконання тестового сценарію. Визначається співвідношенням загального витраченого часу до кількості тестових сценаріїв, за допомогою чого визначається середній час на виконання одного тесту.

– Pass/Fail Ratio. Співвідношення тестових сценаріїв зі статусом Pass відносно Fail.

– Щільність дефектів. Визначення співвідношення кількості знайдених дефектів відносно до одиниці витраченого зусилля.

– Якість тест-дизайну. Оцінка покриття тестовими сценаріями та ефективності використовуваних тестових даних.

На рисунку 2.5 зображено графік прогресу виконання тестування, за допомогою якого є можливість визначити корисні дані для оцінки загальної ефективності.

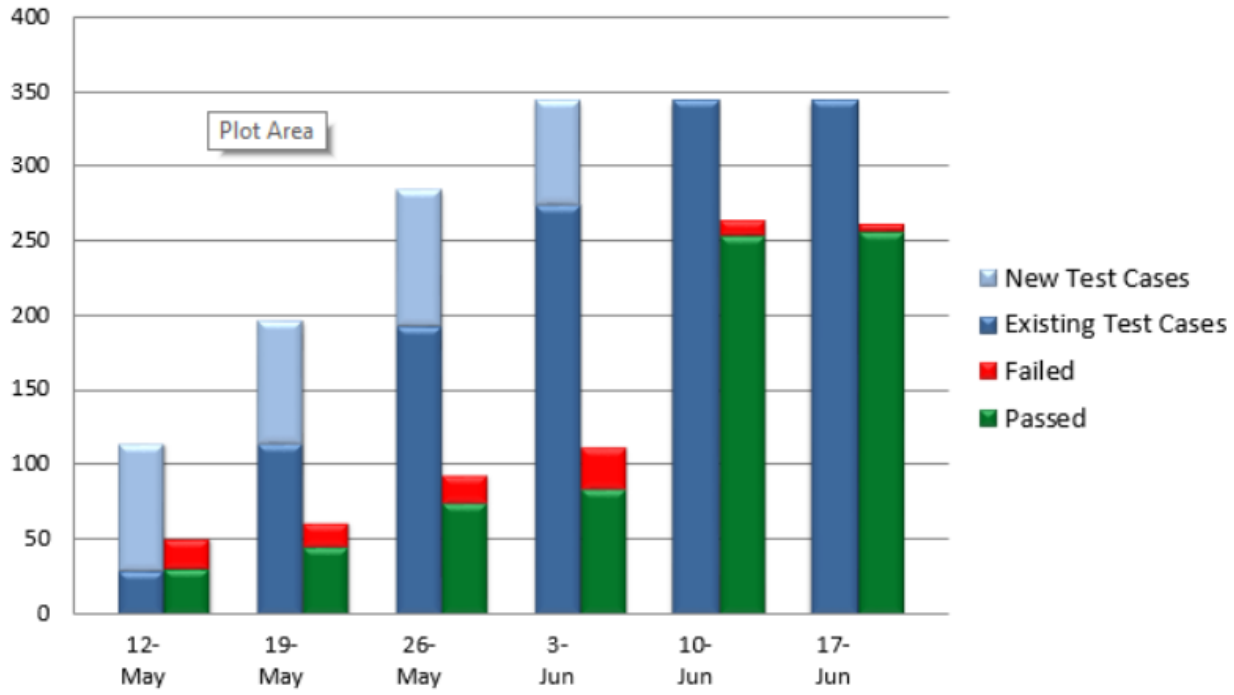


Рисунок 2.5 – Прогрес виконання тестування

Для визначення ефективності автоматизованого тестування використовують наступні метрики: [40]

– Покриття програмного коду. Вимірюється за допомогою визначення відсотка коду програмного продукту, що покритий автоматизованими тестовими сценаріями.

– Час виконання тестового прогону. Вимірювання загального витраченого часу на підготовку, виконання та отримання результатів виконання автоматизованих тестових сценаріїв.

– Кількість помилок у тестових сценаріях. Оцінка кількості автоматизованих тестових сценаріїв, які мають помилки у власному програмному коді.

– Стабільність тестів. Визначення впливу використання автоматизованих тестових сценаріїв на загальний стан системи.

– Вартість розробки та підтримки. Оцінка витрат на розробку, підтримку та впровадження автоматизованих тестових сценаріїв.

На рисунку 2.6 зображено приклад метрики вартості розробки та підтримки тестових сценаріїв.

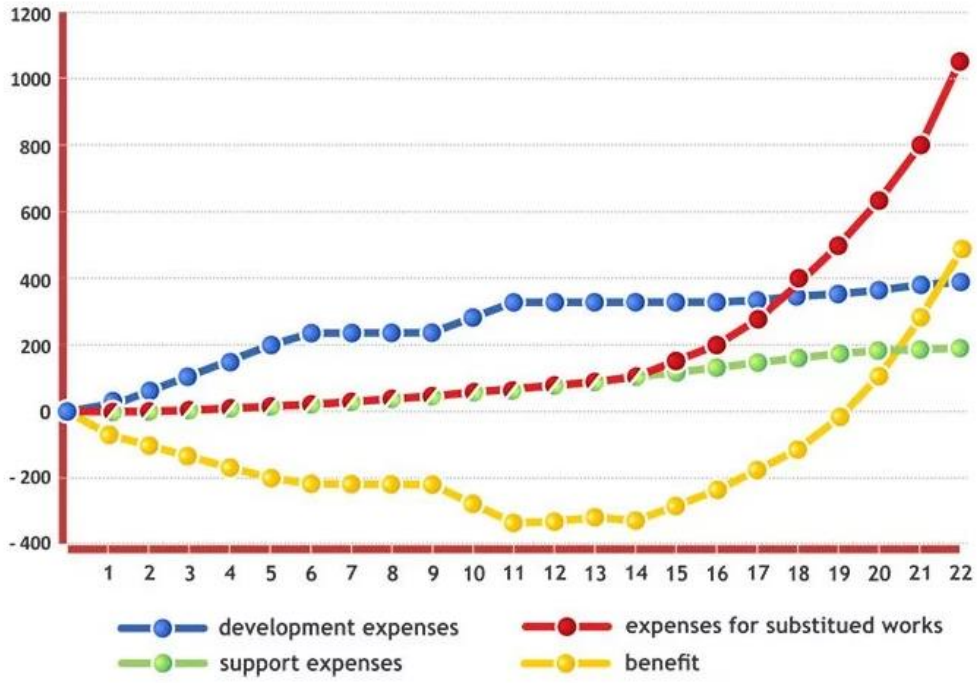


Рисунок 2.6 – Приклад оцінки вартості розробки та підтримки автоматизованих тестових сценаріїв

Таким чином, для подальшого порівняння ефективності ручного та автоматизованого тестування необхідно визначити час на створення та впровадження тестових сценаріїв, витрати часу на повторне використання створених тестових сценаріїв, оцінити загальні витрати на початку та в кінці тестових ітерацій, а також оцінити гнучкість тестових сценаріїв для подальшої підтримки.

2.5 Висновок до другого розділу

В другому розділі кваліфікаційної роботи освітнього рівня «Магістр» оглянуто принципи, висвітлено класифікацію видів тестування та досліджено доцільність використання автоматизованого та ручного підходу для виконання підвидів нефункціонального тестування згідно до класифікації видів тестування за об’єктом.

Крім того, в другому розділі оглянуто найбільш корисні метрики для оцінки ефективності тестування, а також визначено, що для подальшої оцінки ефективності використання ручного та автоматизованого підходів для виконання функціонального тестування необхідно визначити витрачені ресурси на створення і впровадження тестових сценаріїв, гнучкість та витрати коштів обох підходів.

3 ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ РУЧНОГО ТА АВТОМАТИЗОВАНОГО ПІДХОДІВ У ФУНКЦІОНАЛЬНОМУ ТЕСТУВАННІ

3.1 Розробка плану ручного та автоматизованого тестування

Для початку тестування будь-якого програмного продукту необхідно створити важливий артефакт – тест-план.

План тестування є артефактом тестування, у якому детально описуються цілі тестування, стратегії, процесів, часові обмеження тощо [41]. Він розробляється командою QA і використовується всіма командами для підтримки прозорості, контролю та послідовності всієї діяльності з тестування.

Загальна структура тест-плану може варіюватись в залежності від потреб. Основними пунктами є:

1. Обсяг. Окреслення цілей конкретного проекту та надає інформацію про сценарії користувача, призначені для тестування. За необхідності, у технічному завданні можуть бути вказані сценарії або питання, які не будуть розглянуті в рамках проекту.

2. Терміни. Визначення дати початку і терміни, в які тестувальники повинні надати результати.

3. Розподілення ресурсів. Розподіл конкретних модулів, функціоналу або сфер відповідальності між окремими тестувальниками, а також включає ролі в команді.

4. Середовище. Опис характеристик, налаштування та доступність тестового середовища.

5. Інструменти. Перелік інструментів, які будуть використовуватися для тестування, повідомлення про помилки та інших відповідних дій.

6. Поведінка з дефектами. Опис процедури повідомлення про дефекти, включаючи перелік оповіщення осіб і необхідні супровідні елементи для кожного репорту щодо багів. Це може включати визначення спеціальних полів

баг-репорту, наприклад, додавання скріншотів, логування або відеозапис екрану, що демонструють появу дефекта

7. Управління ризиками. Визначення потенційних ризиків, які можуть виникнути під час тестування програмного забезпечення, а також ризики, з якими може зіткнутися ПЗ, якщо воно буде випущене без належного тестування.

8. Критерії входу та виходу. Визначення точки початку та завершення тестування. У цьому пункті описується очікувані результати операцій з контролю якості.

Тест-план створюється для багатьох цілей. Детальний тест-план сприяє потенційному зменшенню дефектів завдяки ранньому початку тестування. Проекти, які мають детальні тест-плани мають менші витрати на розробку та підтримку, завдяки чому покращується загальна якість ПЗ, адже Виявлення та вирішення проблем на ранніх стадіях є більш економічно вигідним порівняно з їх вирішенням після випуску програмного забезпечення [42]. Крім того, наявність плану тестування сприяє кращому плануванню та розподілу ресурсів, запобігаючи виникненню вузьких місць і забезпечуючи ефективне виконання тестової діяльності. Важливою перевагою наявності тест-плану є наявність оцінки ризиків, що дозволяє виявити ризики на більш ранніх стадіях процесу та заздалегідь визначити шляхи усунення передбачуваних обставин. Це можуть бути як ризики, пов'язані з перехресною співпрацею, так і внутрішні ризики команди. [43]

Тест-плани для ручного та автоматизованого тестування відрізняються. Особливостями тест-плану для ручного тестування є:

– Орієнтація на людський фактор. Передбачення цілей тестування та ризиків, які враховують специфічність ручного тестування.

– Детальний опис кроків виконання. Для ручного тестування необхідно детально описати процеси, що дозволить уникнути появі потенційних технічних помилок.

– Опис поведінки з дефектами. Ручне тестування передбачає використання специфічних засобів для створення звітів.

Особливостями тест-плану для автоматизованого тестування є:

– Опис необхідних інструментів. Як правило, автоматизоване тестування передбачає використання значно більшої кількості технічних засобів (мови програмування, бібліотеки, інструменти для тестування API тощо).

– Середовище. Тест-план для автоматизованого тестування може містити вимоги до середовища, у тому числі необхідність налаштування спеціальних змінних середовища для правильного виконання тестів.

– Вимоги до програмного коду. Опис особливостей до форматування, оформлення, відправки на перевірку програмного коду автоматизованих тестових сценаріїв.

– Інтеграція з CI/CD. Опис особливостей роботи з системами керування версіями та CI/CD.

Об’єктом тестування є веб-ресурс Система електронного навчання ТНТУ по URL-адресі: «https://dl.tntu.edu.ua/».

Обсягом тестування є функціональні можливості Системи електронного навчання ТНТУ: логін, реєстрація, пошук курсів, завантаження файлів, відкриття та перехід по меню навігації тощо. Тестування виконувати шляхом способу Black Box, тобто повна відсутність інформації щодо внутрішнього керування програмним продуктом. Необхідно забезпечити покриття різного функціоналу, не концентруватись на одному елементі. Обмежити кількість ручних та автоматизованих сценаріїв до 20 одиниць.

Середовище виконання тестових сценаріїв: Production.

Для автоматизованого тестування використовуються засоби: мова програмування Python, бібліотека Selenium, засіб забезпечення процесу CI/CD Jenkins, засіб створення паралельних тестових сесій Selenoid, IDE Visual Studio Code.

За створення та виконання ручних та автоматизованих тестових сценаріїв, збір результатів тестування, створення звітів тощо відповідальний Іващенко Євгеній.

3.2 Налаштування тестового середовища для автоматизованого тестування

Для розробки автоматизованих тестових сценаріїв необхідно використовувати певну мову програмування та відповідну бібліотеку, а також забезпечити CI/CD та налаштування сесій.

У якості IDE для написання програмного коду було обрано Visual Studio Code. VS Code є одним з найпопулярніших інтегрованих середовищ розробки для програмістів і тестувальників завдяки своїм потужним можливостям автоматизації тестування [44]. Гнучкість та широка підтримка забезпечують різноманітні інструменти для створення та запуску автоматизованих тестів різних типів, включаючи одиничні, інтеграційні та функціональні тести.

Однією з ключових переваг VS Code є використання додаткових плагінів. Існує багато розширень спеціально призначених для підтримки автоматизації тестування. Одним з найважливіших є плагін Test Explorer UI, який надає можливість переглядати, запускати та отримувати результати виконання тестових сценаріїв (див. рисунок 3.1).

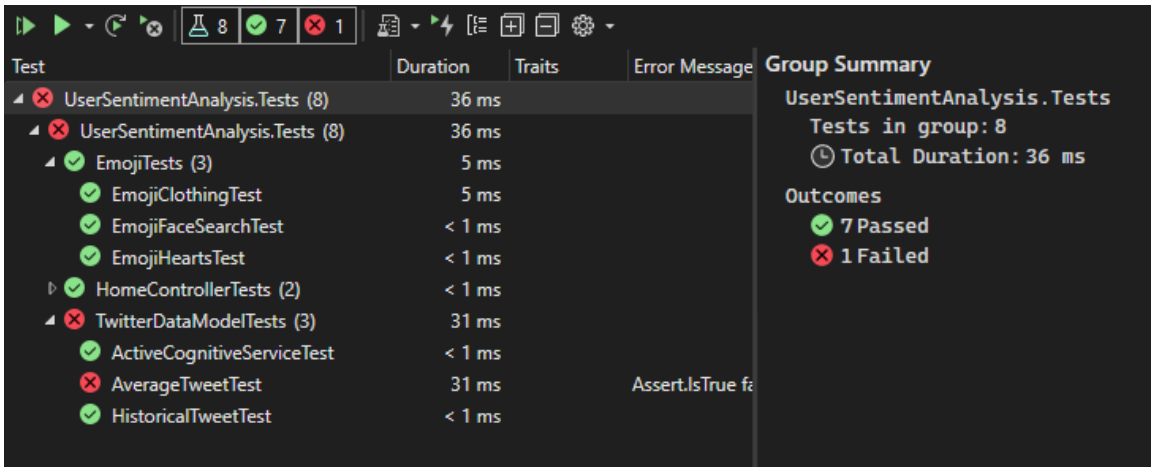


Рисунок 3.1 – Плагін TestExplorer UI у Visual Studio Code

Мовою програмування для автоматизації було обрано Python. Python є однією з найбільш потужних мов програмування для автоматизації тестування завдяки своїй простоті, гнучкості та значній підтримці спільноти [45]. Вона

здатна підтримувати різноманітні види тестування, включаючи функціональне, одиничне, інтеграційне тестування, тестування API тощо.

Значною перевагою мови програмування Python для автоматизації є можливість інтеграції з великою кількістю інструментів, наприклад, Selenium для тестування веб-ресурсів, Appium для мобільних застосунків, Requests для API тощо. Завдяки значній підтримці спільноти є можливість знайти розширену документацію, поради та найкращі практики для використання будь-якої бібліотеки.

Засобом автоматизації було обрано бібліотеку Selenium мови програмування Python. Selenium надає зручний API для написання функціональних/приймальних тестів з використанням Selenium WebDriver. Через Selenium API тестувальник отримує доступ до всіх функціональних можливостей Selenium WebDriver в інтуїтивно зрозумілий спосіб. Використання цього засобу дозволяє програмно взаємодіяти з веб-сторінками, виконувати дії (наприклад, натискання кнопок, введення тексту, перевірка вмісту тощо) та перевіряти результати за допомогою скриптів [46]. Selenium може керувати браузерами в автоматичному режимі, що дозволяє відтворювати різні сценарії тестування.

Selenium Bindings надають зручний API для доступу до Selenium WebDriver, таких як Firefox, Chrome, Remote тощо. Наразі підтримуються версії Python 3.5 та вище.

Основними функціями Selenium є:

– Запуск сесій у браузері. Можливе використання різних браузерів, таких як Chrome, Firefox, Safari тощо.

– Відтворення дій на веб-ресурсі. Автоматизовані сценарії можуть виконувати дії, які є аналогічні діям звичайного користувача, тобто натискання кнопок, зчитування та пошук тексту, ввід даних у поля, завантаження файлів, зміна розміру браузеру тощо.

– Перевірка результату. Selenium збирає та обробляє фактичний результат виконання дій та порівнює з попередньо запрограмованим очікуваним

результатом, за допомогою чого визначається успішність перевірки, а також час на виконання перевірки.

– Інтеграція з іншими засобами. Завдяки гнучкості можна інтегрувати різноманітні засоби, такі як підключення баз даних, генерація тестових даних, інтеграція з CI/CD.

Для реалізації CI/CD процесу було обрано Jenkins. Jenkins є одним з найпопулярніших інструментів для автоматизації процесу Continuous Integration/Continuous Delivery. Використання цього засобу у поєднанні з Selenium дозволяє автоматизувати процес виконання тестів, забезпечуючи швидке та ефективне тестування веб-додатків

Завдяки цьому засобу забезпечується автоматичне збирання, тестування та розгортання коду при кожному внесенні змін у репозиторій. За допомогою Jenkins можна налаштувати регулярні запуски тестів на власному сервері або хмарному сервісі.

Крім того, Jenkins дозволяє масштабувати процес тестування, наприклад, запускаючи тести паралельно на різних платформах або середовищах. Це допомагає прискорити час виконання тестів та забезпечити більш ефективне використання ресурсів.

На рисунку 3.2 зображено процес виконання автоматизованих тестів за допомогою Jenkins.

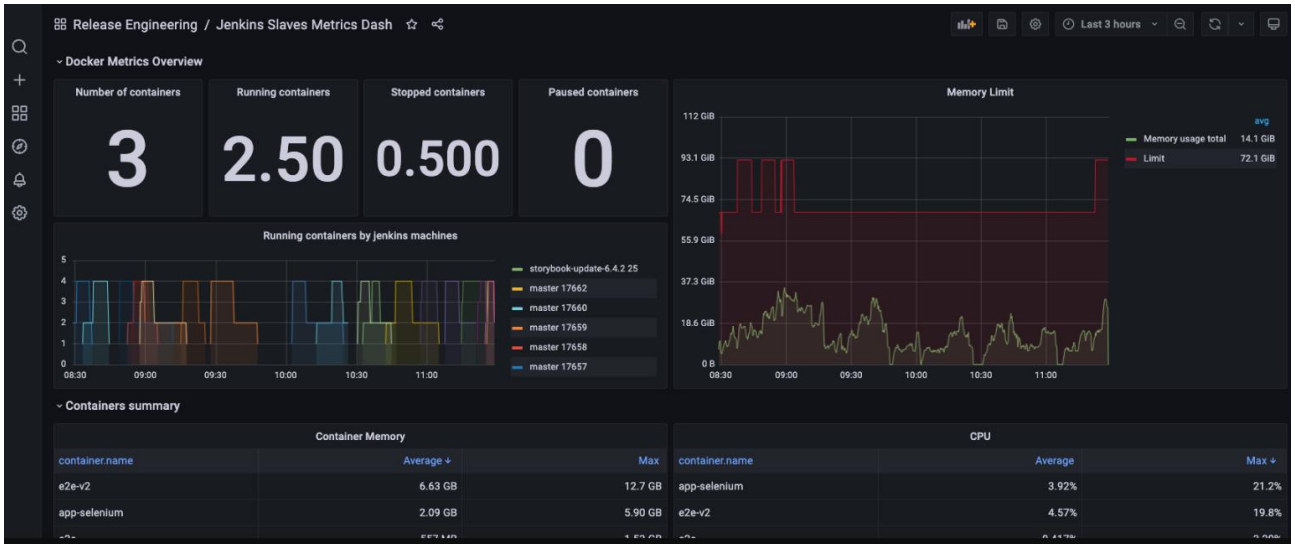


Рисунок 3.2 – UI засобу Jenkins

Для паралельного виконання автоматизованих тестів використовується Selenoid. Selenoid є потужним інструментом для автоматизації тестування, заснованим на Selenium, який надає зручність і ефективність у виконанні веб-тестів у середовищі з контейнерами. Використання Selenoid у поєднанні з Selenium дозволяє швидко та ефективно запускати тести на різних браузерах і платформах, забезпечуючи надійність і масштабованість автоматизованого тестування.

Однією з головних переваг Selenoid є те, що він використовує контейнеризацію для запуску браузерів. Кожен браузер запускається у власному Docker контейнері, що дозволяє ізолювати тестове середовище. Це забезпечує чистоту тестів і уникнення конфліктів між різними тестами. Docker контейнери, на яких працює Selenoid, дозволяють швидко запускати тести та масштабувати виконання тестів паралельно на різних браузерах і платформах. Це дозволяє прискорити час виконання тестів та забезпечує ефективне використання ресурсів.

Таким чином, за допомогою зв’язування Selenoid із засобом Jenkins автоматизовані тести будуть виконуватись ізольовано в окремих койтейнерах, завдяки чому буде забезпечено ізольованість та паралельність виконання, що сприяє зменшенню кількості помилок, ефективнішому використання ресурсів та пришвидшенню роботи.

3.3 Створення та реалізація тестових сценаріїв і скриптів

Для виконання ручного тестування та створення автоматизованих сценаріїв необхідно мати чеклист або тест-кейси.

У контексті створення та реалізації тестових сценаріїв і скриптів в інженерії тестування програмного забезпечення, чеклисти та тест-кейси є ключовими інструментами, які допомагають забезпечити повноту, якість та систематичний підхід до тестування продукту

Чеклист є структурованим списком тестових сценаріїв у форматі заголовку, тобто без детального опису виконуваної перевірки, лише узагальнений опис сценарію.

Тест-кейси представляють собою конкретні сценарії або тестові випадки, які описують детальну послідовність дій для виконання певного тесту.

Текст першого параграфа. Текст першого параграфа. Текст першого параграфа. Розділ, параграф та пункт не повинен закінчуватись рисунком, таблицею, лістингом, маркованим або нумерованим списком.

Таким чином, чеклисти є більш швидким, простим та ефективним способом задокументувати перевірки. У свою чергу, тест-кейси є більш громіздкими, їхнє створення займає більше часу, але вони є значно детальнішими. Підхід з використанням тест-кейсів є зручним, якщо з ними працюють декілька осіб, або функціонал є складним для розуміння, вимагає складних математичних операцій тощо.

Оскільки задачею є поверхневе тестування ресурсу Система електронного навчання ТНТУ, тоді для створення тестових сценаріїв використаємо підхід чеклистів. У таблиці 3.1 наведено чеклист функціональних перевірок до ПЗ Система електронного навчання ТНТУ.

Таблиця 3.1 – Чеклист функціональних перевірок

ID	**Опис перевірки**	**Статус**
1	**2**	**3**
1	Вхід у систему з коректними даними	Не почато
2	Можливість реєстрації з коректними даними	Не почато
3	Вхід у систему з неіснуючими даними	Не почато
4	Вихід з акаунту користувача	Не почато
5	На головній сторінці відображаються курси користувача	Не почато
6	Перехід на сторінку курсу	Не почато
7	Пошук курсів за назвою	Не почато
8	Реєстрація на курс натисканням кнопки «Enroll»	Не почато

Продовження таблиці 3.1

1	2	3
9	Перехід на головну сторінку зі сторінки курсу	Не почато
10	Відкриття меню навігації на сторінці курсу	Не почато
11	Перехід на сторінку з меню навігації	Не почато
12	Скасувати реєстрацію на курс кнопкою «Unenroll»	Не почато
13	Завантаження файлу з файлообміннику	Не почато
14	Вивантаження файлу на скриньку завдань	Не почато
15	Перехід на сторінку з тестами курсу	Не почато
16	Можливість під’єднання до онлайн-конференції	Не почато
17	Редагування та збереження даних у профілі	Не почато
18	Зміна мови інтерфейсу	Не почато
19	Перехід на поштову скриньку користувача	Не почато
20	Перехід на навчальний план	Не почато

Наступним чином необхідно автоматизувати перевірки з таблиці 3.1 за допомогою мови програмування Python та Selenium.

Важливою частиною автоматизації тестових сценаріїв є можливості засобу до пошуку елементів різними способами та імітація різних дій користувача.

Selenium дозволяє виконувати пошук елементів на веб-сторінці за допомогою наступних вказівників:

– Назва класу (By.CLASS_NAME).
– Селектор CSS (By.CSS_SELECTOR).
– ID (By.ID).
– Точний текст (By.LINK_TEXT).
– Частковий текст (By.PARTIAL_LINK_TEXT).
– Назва (By.NAME).
– Тег (By.TAG_NAME).
– XPATH (By.XPATH).

Завдяки значній кількості способів пошуку тестувальник зможе знайти будь-який елемент, оскільки зазвичай елементи веб-сторінки мають лише деякі параметри, наприклад ID та Class Name.

Імітація взаємодії з веб-елементами є важливим аспектом при роботі з Selenium та Python. Оволодіння цими взаємодіями має фундаментальне значення для створення ефективних і повноцінних тестових скриптів за допомогою Selenium і Python. Розглянемо основні можливості імітації користувацьких дій у Selenium: [47]

– Натискання на елементи (кнопки та посилання).

– Ввід даних у поля.

– Очищення тексту.

– Отримання тексту.

– Взаємодія з випадаючими списками, клітинками з прапорцями та перемикачами.

– Робота зі спливаючими вікнами та попередженнями.

– Робота мишки: перетягування (механіка Drag and Drop), подвійне натискання, довге натискання, зміщення по координатам, переміщення до конкретних координат тощо.

– Робота клавіатури: імітація натискання будь-якої клавіші на клавіатурі, а також комбінацій клавіш.

– Робота з вікнами: вибір цільового вікна, перехід на іншу та нову вкладку у браузері.

Крім того, важливим є реалізація очікування при автоматизації тестових сценаріїв. Сьогодні більшість веб-додатків використовують техніку AJAX. Коли сторінка завантажується браузером, елементи на ній можуть завантажуватися через різні проміжки часу. Це ускладнює визначення місцезнаходження елементів: якщо елемент ще не присутній в DOM, функція locate згенерує виключення ElementNotVisibleException. Використовуючи очікування, ми можемо вирішити цю проблему. Очікування забезпечує певний проміжок часу між діями, що виконуються – здебільшого між локалізацією елемента або будь-якою іншою операцією з ним.

Selenium Webdriver надає два типи очікувань – явні та неявні. Явне очікування – це код, який визначає очікування настання певної умови, перш ніж продовжувати виконання коду. Крайнім випадком цього є time.sleep(), який встановлює умову на точний проміжок часу для очікування. Існує декілька зручних методів, які використовуються для визначення умови спрацювання явного очікування:

– Відповідність заголовку.
– Заголовок містить певний текст.
– Наявність певного елементу.
– Видимість певного елементу.
– Вибір певних елементів.
– Стан елементів.
– Поява спливаючого вікна.

В свою чергу, неявне очікування вказує WebDriver опитувати DOM протягом певного часу при спробі знайти будь-який елемент, який є недоступним одразу.

Крім того, Selenium має можливість опрацьовувати помилки та надавати інформацію щодо їхньої появи. До таких помилок відносяться: ElementClickInterceptedException, ElementNotVisibleException, InvalidArgument Exception, JavascriptException, NoSuchDriverException, NoSuchWindowException, TimeoutException тощо. [48]

Необхідно автоматизувати першу перевірку з чеклиста, який наведено у таблиці 3.1.

Перш за все, необхідно імпортувати модуль Webdriver з бібліотеки Selenium для автоматизації тестового сценарію за допомогою коду, оскільки саме цей модуль керує браузером:

```
from selenium import webdriver
```

Також необхідно імпортувати модуль By з бібліотеки Selenium для реалізації можливості пошуку елементів на веб-сторінці за допомогою вказівників:

```
from selenium.webdriver.common.by import By
```

Необхідні ініціалізувати драйвер, який буде керувати браузер, та вказати цільовий URL веб-сайту:

```
driver = webdriver.Chrome()
     driver.get("https://dl.tntu.edu.ua/")
```

За допомогою DevTools знаходимо ID полів, які використовуються для прийому логіну та паролю користувачів. На рисунку 3.3 зображено використання DevTools у браузері Google Chrome.

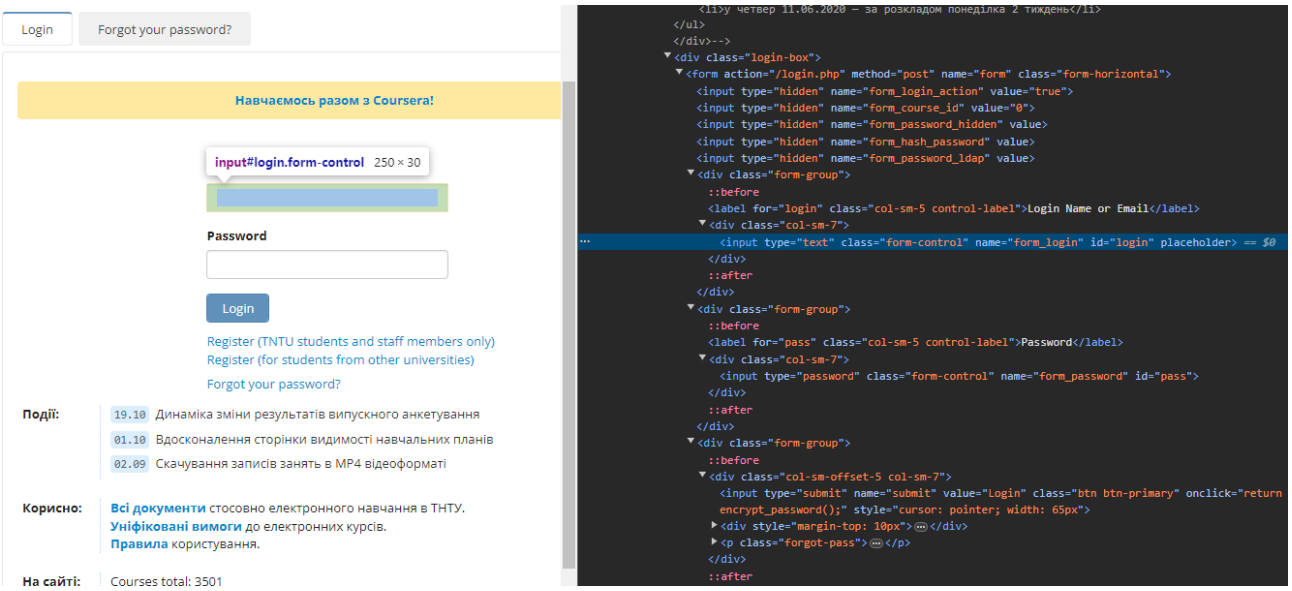


Рисунок 3.3 – Пошук властивостей полів за допомогою DevTools

Передаємо знайдені ID полів у драйвер, а також за допомогою функції send_keys передаємо валідні дані для входу у Систему електронного навчання ТНТУ.

Програмний код передачі ID полів та даних для входу у систему наведено у лістингу 3.1.

Лістинг 3.1 – Програмний код передачі даних у поля

```
# Знаходимо елементи для входу
    username_input = driver.find_element(By.ID, "login")
    password_input = driver.find_element(By.ID, "pass")

    # Введення коректних облікових даних
    username_input.send_keys("jeka200011505@gmail.com")
    password_input.send_keys("VeryStrongPassword33212")
```

За допомогою DevTools знаходимо XPATH для кнопки увійти та натискаємо її:

```
    login_button = driver.find_element(By.XPATH, "/html/body/div/
div/div[2]/div[1]/div[4]/div[2]/form/div[3]/div/input")
    login_button.click()
```

За допомогою функції assert знаходимо текст "You have logged in successfully.". Якщо автоматизований тест знайде цей текст на наступній сторінці, то він буде вважатись успішним, в іншому випадку буде виявлено дефект.

У лістингу 3.2 наведено програмний код автоматизованого тестового сценарію виконання перевірки «Вхід у систему з коректними даними»:

Лістинг 3.2 – Програмний код перевірки «Вхід у систему з коректними даними»:

```
from selenium import webdriver
    from selenium.webdriver.common.by import By

    driver = webdriver.Chrome()

    driver.get("https://dl.tntu.edu.ua/")

    username_input = driver.find_element(By.ID, "login")
    password_input = driver.find_element(By.ID, "pass")

    username_input.send_keys("jeka200011505@gmail.com")
    password_input.send_keys("123123321")

    login_button          =          driver.find_element(By.XPATH,
"/html/body/div/div/div[2]/div[1]/div[4]/div[2]/form/div[3]/div/in
put")
    login_button.click()

    driver.implicitly_wait(5)
```

```
assert "You have logged in successfully." in driver.page_source
driver.quit()
```

На рисунку 3.4 зображено виконання автоматизованого тестового сценарію «Вхід у систему з коректними даними»

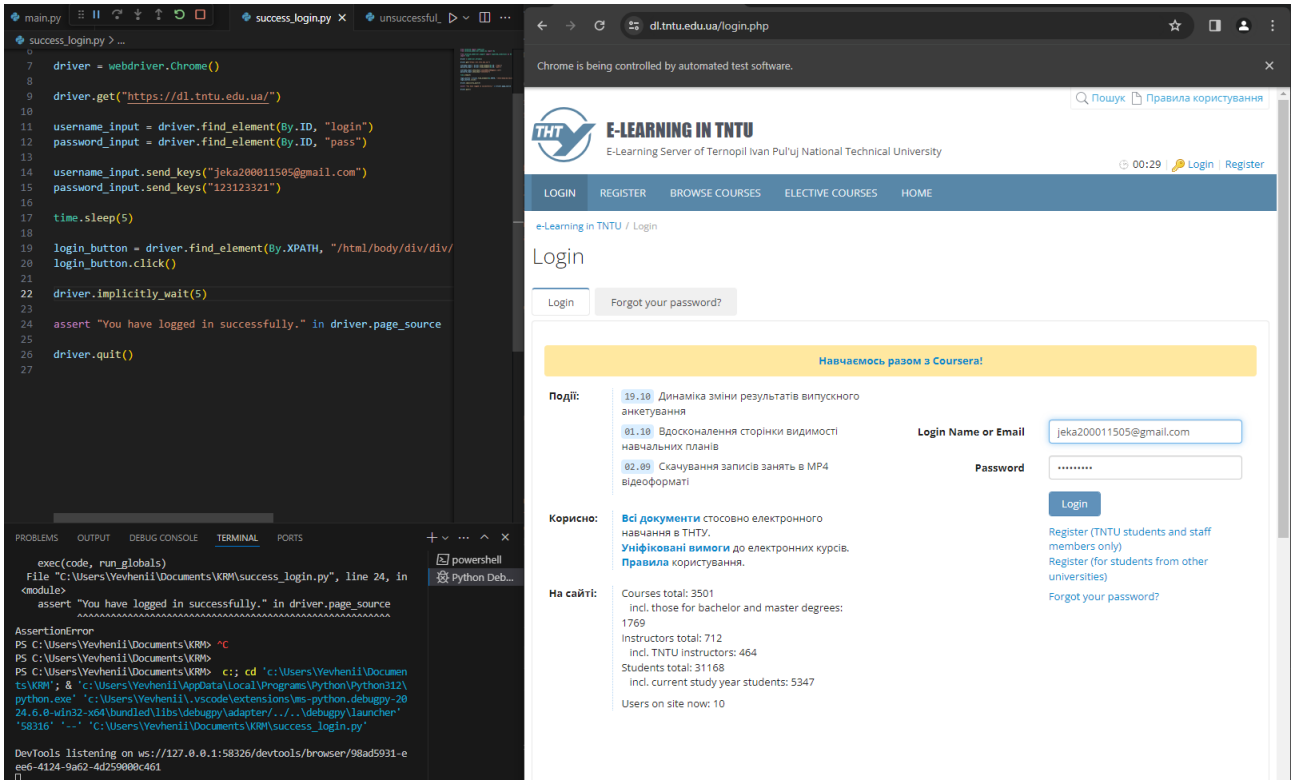


Рисунок 3.4 – Виконання тестового сценарію

Програмний код автоматизованих сценаріїв перевірок 2-20 наведено у Додатку Б.

3.4 Аналіз ефективності використання ручного та автоматизованого підходів для функціонального тестування

Необхідно визначити, який підхід є більш ефективним для функціонального тестування: ручний чи автоматизований. Необхідно визначити час на створення тестових сценаріїв, витрати часу на повторне використання створених тестових сценаріїв, оцінити гнучкість тестових сценаріїв для

подальшої підтримки тестів та загальні умови для використання одного з двох підходів.

Порівняємо витрати часу на створення та виконання ручних та автоматизованих тестових сценаріїв. Час на створення чеклистів та тест-кейсів не враховуємо, оскільки вони необхідні як для ручного, так і автоматизованого тестування.

При створенні та виконанні тестових сценаріїв для Системи електронного навчання ТНТУ було виміряно витрачений час. Таким чином, на виконання одного ручного тестового сценарію було витрачено в середньому 3 хвилини, а на створення на виконання автоматизованого тестового сценарію – 22.5 хвилини.

В подальшому, ці тестові сценарії будуть виконуватись значну кількість разів в рамках регресії – перевірка старого функціоналу на відсутність нових дефектів, які могли виникнути при зміні програмного коду для нового функціоналу. Вважатимемо, що для повторного використання тестового сценарію для ручного тестування буде витрачатись 3 хвилини на одну перевірку, а для автоматизованого – 0.5 хвилини. На рисунку 3.5 зображено моделювання ситуації.

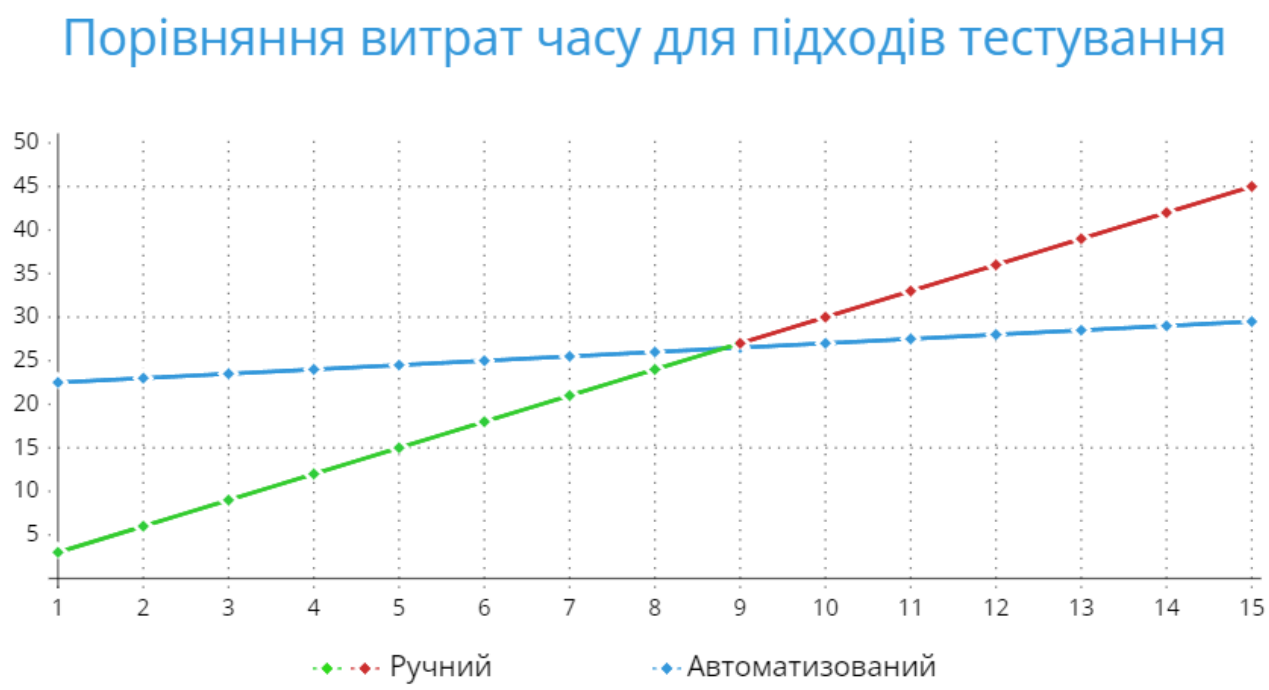


Рисунок 3.5 – Порівняння витрат часу

Таким чином, згідно до рис. 3.5, ручне тестування стає неефективним на 9му повторенні тестового сценарію. В реальних умовах один тестовий сценарій може бути повторено десятки разів.

Варто зазначити, що у цій ситуації не враховано час на підтримку тестових сценаріїв, адже підтримка автоматизованих перевірок витрачає значно більше часу, ніж редагування ручних.

Отже, ручний підхід для функціонального тестування варто використовувати у нестабільних етапах проєкту (наприклад, початковий етап, коли вимоги ще не стабільні), для перевірки тестів, які виконуються рідко, а також для тестування сценаріїв, де необхідна людська точка зору (наприклад, тестування UI/UX). Крім того, ручне тестування варто застосовувати у ситуаціях, коли час або бюджет є строго обмеженими.

Автоматизоване тестування варто використовувати для регресійного тестування, у проєктах з великою кількістю різноманітного функціоналу, для тестування інтеграції та API та для виконання тестових сценаріїв, які є критичними і не допускаються помилки.

3.5 Висновок до третього розділу

В третьому розділі кваліфікаційної роботи освітнього рівня «Магістр» описано елементи тест-плану, подано налаштування середовища для автоматизованого тестування та створено чеклист для тестування Системи електронного навчання ТНТУ.

Крім того, в третьому розділі оглянуто основні принципи використання бібліотеки Selenium для мови програмування Python та автоматизовано тестові сценарії з чеклиста. На основі цього проведено аналіз ефективності використання часу для виконання функціонального тестування ручним та автоматизованим підходами.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Ергономічні вимоги до організації робочого місця оператора

Дотримання ергономічних вимог робочого місця оператора є важливим аспектом для забезпечення комфорту, продуктивності та здоров'я тестувальника програмного забезпечення. [49]

Раціональне планування робочого місця має забезпечувати: найкраще розміщення знарядь і предметів праці, не допускати загального дискомфорту, зменшувати втомлюваність працівника, підвищувати його продуктивність праці. На рисунку 4.1 зображено основні ергономічні вимоги до проектування робочого місця в системі «людина – техніка – виробниче середовище».

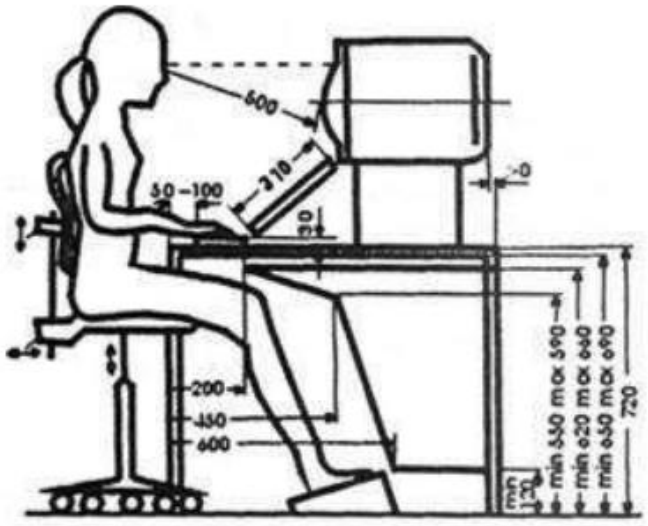


Рисунок 4.1 – Ергономічні вимоги робочого місця оператора

Організація робочого місця включає:

– Облік психофізіологічної сумісності виконавця і засобів праці.

– Аналіз антропометричних характеристик людини для вибору ергономічно-обґрунтованого робочого положення і робочих зон.

– Раціональне компонування робочого місця.

– Облік факторів зовнішнього середовища, у тому числі соціальнопсихологічного аспекту.

Ергономічні вимоги пов'язані зі створенням для людини оптимальних умов праці, що роблять її високопродуктивною та надійною і водночас забезпечують людині необхідні зручності, зберігаючи сили, здоров'я та працездатність.

Наведемо рекомендації для забезпечення ергономічних вимог організації робочого місця оператора:

– Потрібно забезпечити комфортну та підтримуючу спинку стільця, яка підтримує природну кривизну хребта. Також він повинен мати регульовану висоту та підлокітники для оптимальної позиції сидіння.

– Необхідно розмістити робочий стіл на тій висоті, щоб оператор міг зручно дістатися до клавіатури, миші та монітора. Також варто забезпечити достатню робочу площу для розташування всіх необхідних засобів праці.

– Потрібно обрати ергономічну клавіатуру та мишу, які забезпечують комфортну позицію рук і запобігають напруженню м'язів. Клавіатура повинна бути плоскою та мати підставку для зап'ястя.

– Монітор потрібно розмістити на такій висоті, щоб верхній край екрану був на рівні очей. Це допоможе уникнути напруження шиї та зменшити втому очей. Рекомендується також використовувати монітор з антибліковим покриттям та налаштуваннями яскравості і контрастності

– Потрібно забезпечити оптимальну організацію робочого простору, щоб зменшити зусилля та час, потрібний для доступу до необхідних матеріалів. Варто розмістити часто використовувані предмети, такі як ручки або накопичувачі даних, у зручно доступних місцях.

Варто також враховувати індивідуальні потреби та вподобання працівника, а також специфічні вимоги конкретної роботи, щоб створити найкращі умови для комфортної та продуктивної роботи.

4.2 Організація контролю умов праці

Організація контролю умов праці є критичним аспектом для забезпечення безпеки, здоров'я та добробуту працівників, в тому числі тестувальників програмного забезпечення. Контроль є загальною функцією управління,

пов'язаною з виконавською дисципліною. У області охорони праці контроль розглядається як спеціальна функція, пов'язана з перевірками стану для подальшого вироблення управлінських рішень. [50]

Основними принципами організації контролю є:

– Контроль має бути направлений на попереджуючу ідентифікацію небезпечних і шкідливих чинників.

– Перевірки мають бути стимулюючим чинником в підвищенні безпеки і умов праці, а не каральною акцією.

– Процедура перевірок має бути систематичною.

– Перевірки слід проводити там, де вірогідність появи небезпеки найбільша.

– У необхідних випадках слід привертати незалежних фахівців-експертів.

– Контроль, як правило, не повинен порушувати виробничий процес.

– До перевірок слід привертати представників нижчої ланки контролю, уповноважених трудового колективу по питаннях охорони праці і працівників.

– В ході перевірки при виявленні порушень слід давати пояснення про можливі їх наслідки.

– Має бути забезпечена гласність обговорення результатів перевірок і залучення до рішення питань безпеки широкого круга громадськості.

– Обов'язковість виконання заходів, що стосуються усунення виявлення порушень.

Важливою частиною контролю умов праці є розробка і впровадження процедур безпеки та здоров'я, які включають у себе інструкції з безпеки, навчання персоналу правилам та процедурам, пов'язаним з уникненням травм та професійними захворюваннями. Забезпечення необхідного обладнання для особистої захисту, наприклад, респіраторів чи спеціального одягу, також є важливою частиною контролю умов праці.

Об’єктами контролю умов праці є:

– Будівлі, споруди, приміщення виробничого і іншого призначення, устаткування та технологічні процеси.

– Проєктна, виробничо-технічна, санітарно-гігієнічна, облікова і дозвільна документація.

– Чинники виробничого середовища і трудового процесу, що визначають умови праці.

– Засоби індивідуального і колективного захисту, засоби зв'язку.

– Нормативно-правові акти.

– Умови праці жінок, інвалідів і неповнолітніх.

– Знання працівників нормативно-правових актів з охорони праці, технічної документації, прийомів безпечного ведення робіт, сигналів, план ліквідації аварій, запасних виходів, правил поведінки при аваріях.

– Пільги і компенсації за шкідливі, небезпечні і особливі умови праці.

– Режим праці і відпочинку працівників.

Ефективний контроль умов праці передбачає систематичний аналіз усіх аспектів робочого процесу. Важливо забезпечити, щоб контроль умов праці був постійним і регулярним процесом, який враховує зміни в законодавстві та вимоги в сфері безпеки та здоров'я.

Успішна організація контролю умов праці залежить від залучення працівників до процесу і відкритості для їхніх пропозицій та відгуків. Проактивне вирішення питань, пов'язаних з умовами праці, сприяє створенню здорової та продуктивної робочої атмосфери, яка сприяє зростанню якості та ефективності працівників.

4.3 Підвищення стійкості роботи підприємств приладобудівної галузі у воєнний час

Підвищення стійкості роботи об’єктів народного господарства, зокрема підприємств приладобудівної галузі, у воєнний час є однією із основних задач цивільної оборони України.

Під стійкістю роботи підприємств приладобудівної галузі розуміють їх здатність за умов дії надзвичайних ситуацій виробляти продукцію в запланованих обсязі та номенклатурі, а при одержанні слабких чи середніх

руйнувань або порушенні постачання сировини відновлювати своє виробництво в мінімально короткі терміни. Щоб забезпечити нормальну роботу промислових об’єктів будівництва та скоротити можливі матеріальні втрати під час воєнного часу необхідно виконати комплекс різних заходів, які забезпечили б їхнє функціонування.

Здатність об’єкта підприємства приладобудівної галузі випускати продукцію залежить від захисту і нормального функціонування основних елементів сучасного виробництва:

– Виробничий персонал.

– Будинки та споруди з технологічним устаткуванням.

– Система постачання енергією, водою, паливом, устаткуванням та ремонтною базою.

– Система виробничих і кооперативних зв’язків з іншими об’єктами.

На стійкість роботи об’єктів приладобудівної галузі впливають наступні фактори: [51]

– Надійність захисту робітників від дії вражаючих факторів.

– Здатність промислового комплексу протистояти дії вражаючих факторів.

– Надійність систем постачання об’єкта сировиною для виробництва видів продукції.

– Стійкість системи управління виробництвом та цивільною обороною в надзвичайних ситуаціях.

– Готовність об’єкта до проведення рятувальних дій або робіт по відновленню виробництва.

– Захищеність об’єкта від дії вторинних вражаючих факторів.

При вирішенні проблеми підвищення стійкості роботи підприємств приладобудівної галузі, а також інших об’єктів народного господарства, керуються єдиними принциповими положеннями:

– Завчасне проведення заходів цивільного захисту, спрямованих на зниження можливих втрат та руйнувань у разі застосування з боку противника

зброї масового ураження і на створення умов для швидкого відновлення виробництва після часткового руйнування.

– Комплексний підхід в розробці і здійсненні заходів забезпечення стійкості роботи для всіх напрямків діяльності підприємства.

– Узгодження заходів з територіальними і військовими органами управління.

Заходи з підвищення стійкості плануються з урахуванням місцевих умов, ступеня важливості об’єкта, його географічного положення, економічної доцільності проведення заходів. Крім того, при проведені заходів з ЦЗ потрібно враховувати і внутрішні фактори, що впливають на стійкість: розмір виробництва, виду продукції, що випускається, чисельність працівників, рівень їх дисциплінованості і компетентності, особливості технології виробництва, системи постачання виробництва сировиною, технічною і питною водою, газо- та електроенергією.

Існують наступні шляхи і способи підвищення стійкості роботи підприємств приладобудівної галузі:

– Забезпечення надійного захисту робітників і службовців: укриття робітників і службовців, які продовжують роботу на об'єкті у воєнний час.

– Проведення евакуації робітників, службовців і членів їх сімей та забезпечення їх життєдіяльності.

– Захист основних виробничих фондів об'єкта від ураження, підвищення стійкості будівель, споруд проти впливу ударної хвилі чи світлового випромінювання.

– Укриття найбільш уразливого обладнання в захисних пристроях (шатрах, камерах, конусах).

– Часткову зміну технології виробництва.

– Підготовка паливно-енергетичного господарства до роботи у воєнний час.

– Розосередження запасів найбільш уразливого обладнання, приладів, сировини.

– Встановлення виробничих контактів з дублерами постачальниками, необхідних для безперебійної роботи об'єкта.

– Підвищення надійності та оперативності управління виробництвом.

– Проведення підземних кабельних ліній зв'язку до всіх елементів об'єкта.

– Підготовка до виконання робіт по відновленню об'єкта у воєнний час, планування відновлювальних робіт за кількома варіантами.

Підвищення стійкості роботи об’єкта приладобудівного господарства у воєнний час досягається завчасним проведенням комплексу інженерно-технологічних, технологічних і організаційних заходів, спрямованих на максимальне зниження впливу вражаючих факторів зброї масового ураження і створення умов для швидкої ліквідації наслідків.

Підготовка до відновлення порушеного виробництва здійснюється завчасно і передбачає планування відбудовних робіт по декількох варіантах: підготовку ремонтних бригад, створення необхідного запасу матеріалів і устаткування, надійний його захист.

4.4 Висновок до четвертого розділу

В четвертому розділі кваліфікаційної роботи описано ергономічні вимоги до організації робочого місця оператора та основні принципи і об’єкти організації контролю умов праці, що сприяє зростанню якості, продуктивності та задоволеності працівників.

Крім того, у четвертому розділі визначено фактори та способи підвищення стійкості роботи підприємств приладобудівної галузі у воєнний час, що досягається завчасним проведенням комплексу інженерно-технологічних, технологічних і організаційних заходів.

ВИСНОВКИ

Проведене дослідження ефективності використання різних підходів до тестування ПЗ визначило, що автоматизоване тестування дозволяє зменшити час, необхідний для виконання тестових сценаріїв, підвищує швидкість виявлення помилок і забезпечує більшу точність результатів порівняно з ручним тестуванням. З іншого боку, ручне тестування залишається необхідним для певних аспектів, таких як тестування користувацького досвіду та валідація інтерфейсу. Отже, оптимальний підхід полягає у поєднанні обох методів з урахуванням специфіки проекту та потреб бізнесу.

Результати дослідження також підкреслили необхідність правильної стратегії підготовки та підтримки автоматизованих тестів для забезпечення їхньої ефективності та стійкості в майбутньому. Важливо також враховувати витрати на впровадження автоматизованого тестування та його підтримку, щоб забезпечити оптимальний баланс між якістю та вартістю процесу тестування.

В першому розділі кваліфікаційної роботи освітнього рівня «Магістр»:

– Визначено сфери застосування та цілі тестування ПЗ.

– Розглянуто основні принципи використання підходів тестування.

– Висвітлено особливості використання автоматизованого тестування як програмного коду.

– Досліджено сучасні засоби автоматизації тестування.

В другому розділі кваліфікаційної роботи:

– Описано принципи тестування ПЗ.

– Досліджено класифікацію видів тестування.

– Подано порівняльний опис використання підходів до нефункціонального тестування.

– Запропоновано оцінку ефективності виконання функціонального тестування ручним та автоматизованим підходами.

В третьому розділі кваліфікаційної роботи:

– Розроблено тест-план для виконання дослідження ефективності тестування.

– Налаштовано середовище для автоматизованого тестування ПЗ.

– Створено чек-лист тестових сценаріїв.

– Розроблено автоматизовані тестові сценарії за допомогою мови програмування Python та Selenium.

– Проаналізовано оцінку ефективності виконання ручного та автоматизованого підходів.

У розділі «Охорона праці та безпека в надзвичайних ситуаціях» висвітлено ергономічні вимоги до організації робочого місця оператора, описано організацію контролю умов праці та наведено способи підвищення стійкості роботи приладобудівної галузі у воєнний час.

ПЕРЕЛІК ДЖЕРЕЛ

1. Chen, Xingru, Muhammad Usman, and Deepika Badampudi. "Understanding and evaluating software reuse costs and benefits from industrial cases—A systematic literature review." Information and Software Technology (2024): 107451.

2. Bodnarchuk, Ihor, et al. "Adaptive method for assessment and selection of software architecture in flexible techniques of design." 2018 IEEE 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT). Vol. 1. IEEE, 2018.

3. Hooda, Itti, and Rajender Singh Chhillar. "Software test process, testing types and techniques." International Journal of Computer Applications 111.13 (2015).

4. “Тестування програмного забезпечення: типи, види та застосування” FoxmindEd, foxminded.ua/. Дата доступу 14 травня 2024.

5. Dhore, Prasad, et al. "Brief Review On Different Manual Software Testing Approaches & Procedure." Journal of Pharmaceutical Negative Results (2023): 455-464.

6. Feshchuk, Inna. “Manual Testing Guide for Beginners.” QA Madness, www.qamadness.com/manual-testing-guide-for-beginners/. Дата доступу 14 травня 2024.

7. Sneha, Karuturi, and Gowda M. Malle. "Research on software testing techniques and software automation testing tools." 2017 international conference on energy, communication, data analytics and soft computing (ICECDS). IEEE, 2017.

8. “Автоматизоване тестування.” QALight, qalight.ua/baza-znaniy/avtomatizovane-testuvannya/. Дата доступу 14 травня 2024.

9. Contan, Andrei, Catalin Dehelean, and Liviu Miclea. "Test automation pyramid from theory to practice." 2018 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR). IEEE, 2018.

10. Ateşoğulları, Dilara, and Alok Mishra. "Automation testing tools: a comparative view." (2020).

11. Фіалка, Є. В. "Дослідження та порівняння фреймворків для автоматизованого тестування вебзастосунків." (2024).

12. Pelivani, Elis, and Betim Cico. "A comparative study of automation testing tools for web applications." 2021 10th Mediterranean Conference on Embedded Computing (MECO). IEEE, 2021.

13. Wardhan, Harshita, and Suman Madan. "Study On Functioning Of Selenium TestingTool." International Research Journal of Modernization in Engineering Technology and Science Www. Irjmets. Com@ International Research Journal of Modernization in Engineering (2021).

14. Mobaraya, Fatini, and Shahid Ali. "Technical Analysis of Selenium and Cypress as functional automation framework for modern web application testing." 9th International Conference on Computer Science. 2019.

15. Tjandra, Suhatati, Indra Maryati, and Joshua Theopilus. "Automated software testing for multi platform applications using katalon." (2021).

16. Kharchenko, Aleksandr, et al. "The Survey of Global Software Design Processes." 2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T). IEEE, 2021..

17. Dimoski, Davor, et al. "Testing RESTful APIs–Use Case: RESTful API for Solving Multidimensional Time–Independent Schrödinger Equation." (2022).

18. Котлярчук, Д. В., О. В. Романюк. Аналіз методів тестування інтерфейсу користувача. Diss. ВНТУ, 2020.

19. “Автоматизуємо тестування: коли, навіщо і кому це потрібно.” Newline.tech, newline.tech/test-automation-when-why-and-who-needs-it_uk/. Дата доступу 14 травня 2024.

20. “Друкарня – українська блог платформа.” Друкарня, drukarnia.com.ua/articles/avtomatichne. Дата доступу 14 травня 2024.

21. Юніт-тест проти інтеграційного тесту: в чому різниця?” CSS Code.org, uk.css-code.org/8222626-unit-test-vs-integration-test-whats-the-difference. Дата доступу 14 травня 2024.

22. “Unit Testing vs. Integration Testing: 5 Key Differences and Why You Need Both.” Codefresh, codefresh.io/learn/unit-testing/unit-testing-vs-integration-testing-5-key-differences-and-why-you-need-both. Дата доступу 14 травня 2024.

23. Winkler, Dietmar, Kristof Meixner, and Stefan Biffl. "Towards flexible and automated testing in production systems engineering projects." 2018 IEEE 23rd international conference on emerging technologies and factory automation (ETFA). Vol. 1. IEEE, 2018.

24. Bagare, Praveen, and Ruslan Desyatnikov. "Test Data Management in Software Testing Life Cycle-Business Need and Benefits in Functional, Performance, and Automation Testing." (2018).

25. Lindholm, David. "Economics of Test Automation: Test case selection for automation." (2019).

26. Guan, Dayu. Manual to automated testing. Diss. Open Access Te Herenga Waka-Victoria University of Wellington, 2014.

27. Reddy, G. C. “Drawbacks of Manual Testing.” Software Testing, www.gcreddy.com/2021/10/drawbacks-of-manual-testing.html. Дата доступу 14 травня 2024.

28. “Pros and Cons of Automated Testing.” Uilicious.com, uilicious.com/blog/pros-cons-automated-testing/. Дата доступу 14 травня 2024.

29. Готович, В. А., and А. В. Мачужак. "Застосування методології CI/CD для автоматизації процесів тестування та розгортання програмного забезпечення." Матеріали XI Міжнародної науково-практичної конференції молодих учених та студентів „Актуальні задачі сучасних технологій “ (2022): 131-132.

30. Serhiy, Nemchynskyy. “Принципи тестування: їх концепції та підходи.” FoxmindEd, foxminded.ua/pryntsypy-testuvannia/. Дата доступу 14 травня 2024.

31. Kharchenko, Alexander, Ihor Bodnarchuk, and Vasyl Yatcyshyn. "The method for comparative evaluation of software architecture with accounting of trade-offs." American Journal of Information Systems 2.1 (2014): 20-25.

32. Іщук, В. І., and Боднарчук, І. О. "Сертифікація програмного забезпечення на основі моделі якості." Збірник тез доповідей VI Міжнародної науково-технічної конференції молодих учених та студентів „Актуальні задачі сучасних технологій “ 2 (2017): 73-74.

33. Hromyak, Roman, and Vasyl Nemish. "Estimation of the structural ρ parameter for a number of structural materials." Вісник Тернопільського національного технічного університету 112.4 (2023): 67-72.

34. “Нефункціональне тестування ПЗ - Q & A.” Qalearning, qalearning.com.ua/theory/lectures/material/nonfunctional-testing/. Дата доступу 14 травня 2024.

35. “Види тестування, пов’язані зі змінами. Кросбраузерність. - Q & A.” Qalearning, qalearning.com.ua/theory/lectures/material/regression-testing. Дата доступу 14 травня 2024.

36. “White/Black/Grey Box-тестування.” QALight, qalight.ua/baza-znaniy/white-black-grey-box-testuvannya/. Дата доступу 14 травня 2024.

37. “Non Functional Testing - a Detailed Overview.” Testsigma Blog, testsigma.com/blog/non-functional-testing/. Дата доступу 14 травня 2024.

38. Takanen, Ari, et al. Fuzzing for software security testing and quality assurance. Artech House, 2018.

39. “Most Important Software Test Metrics.” QAwerk, qawerk.com/blog/most-important-software-test-metrics. Дата доступу 14 травня 2024.

40. “11 Test Automation Metrics and Their Pros & Cons.” Sealights, www.sealights.io/regression-testing/11-test-automation-metrics-and-their-pros-cons. Дата доступу 14 травня 2024.

41. “Test Plan.” QALight, qalight.ua/baza-znaniy/test-plan-2/. Дата доступу 14 травня 2024.

42. Лавінський, Г. В. "Автоматизовані системи обробки економічної інформації." К.: Вища школа (1995).

43. Гром’як, Роман, and Ігор Боднар. Проектування інформаційного забезпечення систем обробки економічної інформації. ТНЕУ, 2009.

44. Quoy, Lucia. “Visual Studio vs Visual Studio Code: What’s the Key Difference?” DistantJob, distantjob.com/blog/visual-studio-vs-visual-studio-code/. Дата доступу 14 травня 2024.

45. Srinath, K. R. "Python–the fastest growing programming language." International Research Journal of Engineering and Technology 4.12 (2017): 354-357.

46. Gojare, Satish, Rahul Joshi, and Dhanashree Gaigaware. "Analysis and design of selenium webdriver automation testing framework." Procedia Computer Science 50 (2015): 341-346.

47. “Selenium Python Tutorial” GeeksforGeeks, www.geeksforgeeks.org/selenium-python-tutorial/. Дата доступу 14 травня 2024.

48. “Selenium with Python — Selenium Python Bindings 2 Documentation.” Readthedocs.io, selenium-python.readthedocs.io/. Дата доступу 14 травня 2024.

49. Єсінова, Ніна Ігорівна. "Економіка праці та соціально-трудові відносини." (2012).

50. Голінько, В. І. "Контроль умов праці." (2018).

51. Гасило, Ю. А., et al. "Охорона праці в галузі та цивільний захист: навч. посіб." Кам’янське: ДДТУ (2017).