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

Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: «Розробка і впровадження сценарію автотестування на основі аналізу методів та інструментів автоматизації ПЗ»

Виконав: студент VI курсу, групи СПм-61
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва спеціальності)

_____ Дацко А.І..
(підпис) (прізвище та ініціали)

Керівник _____ Пастух О.А
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2025

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

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

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Петрик М.Р.
(підпис) (прізвище та ініціали)
« » 20___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Дацку Андрію Ігоровичу

1. Тема роботи Розробка і впровадження сценарію автотестування на основі аналізу методів та інструментів автоматизації тестування ПЗ

Керівник роботи Пастух Олег Анатолійович, д.т.н., проф. каф. ПІ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом по університету від « 27 » листопада 2025 року № 4/7-1045

2. Термін подання студентом роботи 13.12.2025

3. Вихідні дані до роботи наукові літературні джерела

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1 Аналіз предметної області . 2. Теоретична частина
3. Практична частина. 4. Охорона праці та безпека в надзвичайних ситуаціях

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Тема роботи. 2. Мета, задачі, об'єкт, предмет дослідження
3. Процес розробки та тестування ПЗ. 4. Види тестування ПЗ. .
5. Типи тестування знаннчя коду та рівні автоматизації..
6. Інструменти автоматизації тестування. 7. Різні браузери та їх рушії.
8. Порівняння різних платформ автоматизації тестування.
9. Додаткові використані технології. 10. Ініціалізація Node.js та Playwright.
11. Повний цикл прогону всього тесту.
12. Режим користувача Playwright.
13. Фрагмент вихідного коду авто-тесту.
14. Основні результати проведеного дослідження.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Осухівська Г.М., к. т. н., доц., зав. каф. КС		
Безпека в надзвичайних ситуаціях			

7. Дата видачі завдання _____ 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Аналіз предметної галузі дослідження	19.09–27.09.25	Виконано
2.	Обґрунтування актуальності дослідження	30.09–10.10.25	Виконано
3.	Огляд існуючих механізмів та засобів	11.10–28.10.25	Виконано
4.	Проведення дослідження механізмів та засобів опрацювання даних	29.10–15.11.25	Виконано
5.	Оформлення розділу «Аналіз предметної області»	16.11–28.11.25	Виконано
6.	Оформлення розділу «Теоретична частина»	29.11–05.12.25	Виконано
7.	Оформлення розділу «Практична частина»	06.12–10.12.25	Виконано
8.	Оформлення розділу «Охорона праці та безпека в надзвичайних ситуаціях»	05.12–12.12.25	Виконано
9.	Нормоконтроль	13.12–15.12.25	Виконано
10.	Перевірка на плагіат	13.12–17.12.25	Виконано
11.	Попередній захист роботи	18.12.25	Виконано
12.	Захист кваліфікаційної роботи	22.12.25	

Студент _____ Дацко А.І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Пастух О.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дацко Андрій Ігорович. Розробка і впровадження сценарію автотестування на основі аналізу методів та інструментів автоматизації тестування ПЗ. Кваліфікаційна робота освітнього ступеня «магістр». Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, група СПм-61 спеціальність 121 «Інженерія програмного забезпечення». Тернопіль, 2025. Сторінок 66, таблиць 1, рисунків 24, додатків 2, бібліографічних посилань 25.

Метою роботи є проведення докладного аналізу сучасних інструментів автоматизації тестування ПЗ та впровадження сценарію автотестування.

Об’єктом дослідження є веб-застосунок онлайн калькулятора Chrome app.

Предметом дослідження є процес автоматизації його тестування за допомогою інструмента Playwright. Методи дослідження включають: фундаментальні положення інженерії програмного забезпечення, моделювання архітектури, тестування функціональних компонентів.

У кваліфікаційній роботі представлені основні визначення, історія розвитку та виникнення, різні види та методи тестування, основні принципи та рівні автоматизації.

Проведено огляд сучасних інструментів автоматизації тестування програмного забезпечення, розглянуто принципи, типи, види, рівні тестування.

Здійснено аналіз популярних інструментів створення програмного забезпечення, автоматизації, середовищ для розробки та написання автотестів.

Реалізовано тестовий сценарій веб-застосунку «Калькулятор», за допомогою одного з розглянутих інструментів автоматизації.

Ключові слова: автотест, сценарій, тестувальник, Node.js, Playwright

ABSTRACT

Datsko Andrii Ihorovich. Development and implementation of an automated testing scenario based on analysis of software testing automation methods and tools. Master Thesis. Ternopil Ivan Puluj National Technical University, Department of Software Engineering, Group SPm-61, Specialty 121 "Software Engineering". Ternopil, 2025. Pages 66, tables 1, figures 24, annexes 2, bibliographic references 25.

The purpose of the work is to conduct a detailed analysis of modern software testing automation tools and implement an autotesting scenario.

The object of the study is the online calculator web application Chrome app.

The subject of the study is the process of automating its testing using the Playwright tool. The research methods include: fundamentals of software engineering, architecture modeling and testing of functional components.

The Thesis presents the main definitions, the history of development and emergence, various types and methods of testing, basic principles and levels of automation.

A review of modern software testing automation tools was conducted, the principles, types, types, and levels of testing were considered.

An analysis of popular software creation tools, automation, and environments for developing and writing autotests was carried out.

A test scenario for the "Calculator" web application was implemented using one of the considered automation tools..

Keywords: autotest, script, tester, Node.js, Playwright

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення.

СКВ – система контролю версій.

Тестування ПЗ – це процес перевірки відповідності реальної поведінки програми очікуваній, спрямований на виявлення дефектів, помилок та недоліків перед випуском продукту.

Фреймворк – програмна платформа, що визначає структуру програмного середовища.

ШІ – штучний інтелект.

API (англ. Application Programming Interface) – опис способів, якими одна комп'ютерна програма може взаємодіяти з іншою програмою.

CSS (агл. Cascading Style Sheets) – мова опису стилів та зовнішнього вигляду документів веб-сторінки.

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

HTML (англ. HyperText Markup Language) – мова гіпертекстової розмітки для перегляду веб-сторінок у браузері.

QA -інженер (англ. Quality Assurance - забезпечення якості) – спеціаліст з контролю за якістю інформаційного продукту на всіх етапах його розробки.

UI (англ. User Interface) – цифрові елементи, за допомогою яких користувач може взаємодіяти з цифровим продуктом.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Тестування ПЗ дає можливість оцінити ПЗ всіх стадіях його виробництва та, зрештою, дати оцінку якості реалізованого ПЗ. Методом тестування є процес виконання кроків і певної послідовності дій або ручним, або автоматизованим способом, з метою знаходження помилок і багів у продукті, що тестується.

Тестування є значущим етапом при розробці ПЗ, тому як кількість потенційних тестових прикладів навіть для найпростіших компонентів може здатися майже нескінченною. Щоб працювати ефективно та оптимізувати тимчасові витрати, тестувальники застосовують стратегії та перевіряють найважливіші функції. Вони намагаються створювати тести, які повторювали послідовність дій реального користувача, цим виявляючи помилки чи дефекти ПЗ на етапі використання.

Оптимальна оцінка якості ПЗ та передбачення можливих ризиків для користувачів визначаються ефективним тестуванням. Такий процес дає змогу одержати об'єктивні дані щодо працездатності та надійності виробів, а також їх здатність запобігати можливим збоям та помилкам в експлуатації.

Метою роботи є проведення докладного аналізу сучасних інструментів автоматизації тестування ПЗ та впровадження сценарію автотестування.

Для досягнення мети потрібно вирішити наступні завдання:

– дослідити історію появи тестування ПЗ;

– вивчити принципи, типи, види, рівні тестування;

– проаналізувати різні інструменти та середовища автоматизації тестування;

– виконати порівняння їх можливостей та властивостей і провести вибір інструменту та середовища написання автотестів;

– реалізувати тестовий сценарій досліджуваного калькулятора, за допомогою одного з розглянутих інструментів автоматизації.

Об’єкт дослідження: веб-застосунок онлайн калькулятора Chrome app.

Предмет дослідження: процес автоматизації його тестування за допомогою інструмента Playwright.

Наукова новизна роботи: запропоновано використати комбінацію спеціалізованого інструменту тестування та IDE для втілення сценарію автоматизації тестування.

Практичне значення одержаних результатів. Розроблений сценарій досліджуваного калькулятора може бути успішно використаний і поширений на інші веб-застосунки для їх автоматизованого тестування.

Апробація. Окремі результати дослідження доповідалися на XIV Міжнародна науково-практична конференція молодих учених та студентів «Актуальні задачі сучасних технологій» (м. Тернопіль, грудень 2025 р.) та XIII науково-технічній конференції «Інформаційні моделі, системи та технології» як опубліковані тези (м. Тернопіль, грудень 2025 р.).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

В цьому розділі буде проаналізовано предметну область, описана коротка історія становлення тестування ПЗ.

Необхідно виконати постановку завдання та навести основні цілі виконуваного дослідження.

Також буде описано власне поняття тестування як такого та сім базових його принципів.

1.1 Історія виникнення тестування ПЗ

Відповідно до історії, тестування зародилося з появою перших написаних програм. Після Другої світової війни вчені Манчестерського університету Том Кілберн та Фредерік Вільямс були творцями першого ПЗ, яке з'явилося в 1947 році, що виконувало математичні обчислення, використовуючи машинний код.

У контексті історії тестування варто також згадати і першу помилку в ПЗ. 9 вересня 1947 року група інженерів і програмістів, які працювали над комп'ютером Mark II, зіткнулася з проблемою в роботі. Коли вони аналізували причину збою, виявили, що в одному з реле машини забилася міль. Вона застрягла між контактами реле, викликаючи коротке замикання та неправильну роботу комп'ютера. Цей інцидент був детально задокументований і став першим "багом" в історії комп'ютерів.

Відділення напряму тестування ПЗ відбулося у 1950-х роках, тоді головним правилом було покриття всього коду тестами. Пізніше ПЗ ставало лише складнішим, і стало ясно, що вичерпне тестування вже неможливе. Вперше про тестування згадав Джозеф Джуран у його книзі «Посібник з контролю якості», завдяки чому став батьком тестування. Також у теорії управління якістю ПЗ, слід згадати роботу письменника Філіпа Б. Кросбі, який виділив три основні етапи: контроль, планування та модернізація. Цей підхід став основним у галузі розробки ПЗ.

Також, у 1957 році Чарльз Л. Бейкер у своїй рецензії на книгу Дена Маккракена «Програмування цифрових комп'ютерів» зробив істотний внесок, виділяючи окремі аспекти тестування ПЗ від процесів налагодження. Ця подія вважається значним моментом у розвитку методологій та методів перевірки ПЗ.

У 1960-х роках увага зосереджувалась на концепції "вичерпного" тестування, яке вимагало застосування всіх шляхів у коді або всіх можливих вхідних даних. Однак було зазначено, що подібний підхід неможливо застосувати в повному обсязі через величезну кількість потенційних вхідних даних, безліч шляхів виконання програми та складнощів у виявленні проблем в архітектурі та специфікаціях. В результаті "вичерпне" тестування було відкинуто як теоретично неможливе.

У 1960-ті і 1970-ті роки з'явився метод, який отримав назву "доказ коректності". Однак його використання вимагало значних часових витрат, тоді як процес тестування системи залишався недостатньо точним.

З початку 1970-х років майже одночасно існували два ключові варіанти тестування ПЗ. Перший варіант представляв тестування як доказ працездатності програми у певних умовах. Цей метод дозволяв переконатися, що програма виконує поставлені перед нею завдання та відповідає всім вимогам. Другий варіант, навпаки, орієнтувався виявлення непрацездатності програми у певних умовах. Отже, тестування дозволяло виявити ситуації, коли програма працює некоректно. Ці два підходи не суперечать одне одному, а навпаки, доповнюються.

Приблизно через десять років після появи цих методів тестування почалося не в кінці циклу розробки, а упродовж усього процесу. Це дозволяло виявляти проблеми та помилки відразу чи практично відразу після їх виникнення. У разі ж, якщо тестування проводилося наприкінці циклу розробки і було виявлено помилку, була потрібна значна кількість зусиль і часу на пошук кореня проблеми.

Також на початку 1970-х років тестування ПЗ визначалося як процес, спрямований на підтвердження коректності роботи продукту чи перевірку його правильної функціональності. У рамках зароджуваної галузі програмної інженерії, верифікація ПЗ розглядалася, як доказ його правильності, проте на практиці цей

підхід вимагав багато часу і не був всебічний. Ефективність методу доказу правильності була оскаржена, і його було відкинуто як неефективний спосіб тестування.

У наступні роки тестування ПЗ почало розглядатися як процес, спрямований на виявлення помилок, а не на підтвердження правильної роботи. Успішним вважався тест, який виявляв раніше невідомі проблеми. Цей підхід протилежний попередньому та акцентував увагу на виявленні та усуненні недоліків у ПЗ. Таким чином, два визначення тестування були «парадоксом тестування», де, з одного боку, воно підтверджувало правильність роботи продукту, а з іншого — виявляло його недоліки. Однак другий підхід виявився продуктивнішим, оскільки дозволяв активно працювати над поліпшенням якості ПЗ, не ігноруючи його недоліки.

Наприкінці 1970-х був розроблений новий метод тестування ПЗ, спрямований на виявлення помилок у системі, які до цього моменту залишалися непоміченими. Цей метод можна розглядати як поліпшення попереднього, при якому приділяли увагу лише подальшому видаленню виявлених помилок. Основний принцип роботи таких методів тестування полягає в тому, що при виявленні помилок викликається програма та проводяться додаткові тести, щоб переконатися, що помилки справді є проблемами ПЗ, а не результатом помилок у процесі тестування.

З 80-х років тестування ПЗ проходило стадію активного розвитку, набуваючи відомих та актуальних форм і концепцій, які застосовуються і до сьогодні. У цей час тестування ПЗ досягло нового рівня, з'явилися нові методології, стандарти та інструменти, які сприяють оптимізації перевірки. До речі, слід зазначити, що з 1983 по1987 року основну увагу фахівців було зосереджено на оцінці та вимірі якості ПЗ. Тестування значно покращило впевненість у працездатності ПЗ. Тестувальники продовжували свою роботу доти, доки не досягали прийнятного рівня, при якому кількість виявлених помилок знижувалася. Цей метод переважно застосовувався до об'ємного ПЗ.

У 1980-ті роки було впроваджено нову практику у процесі тестування ПЗ - запобігання дефектам. Цей підхід, відомий як "проєктування тестів", значно

просунувся вперед, порівняно з попередніми методами. Він застосовувався до програми лише після її повної розробки та охоплював тестування вимог, дизайну та коду програми, а також перевірку ефективності самих тестів. Пізніше було розроблено інструменти автоматизованого тестування, що дозволяють комп'ютерам виконувати більше тестів, що значно підвищило ефективність процесу тестування ПЗ.

Налагодження тривалий час була основним способом перевірки ПЗ і залишалася такою протягом двох десятиліть. До 1980-х років команди розробників почали проводити тестування програм у реальних умовах, не обмежуючись лише пошуком та усуненням помилок. Цей підхід відкрив шлях до більш широкого погляду на тестування, що включає не тільки процес налагодження, а й поліпшення якості програмного продукту в цілому.

Згодом тестування продовжувало еволюціонувати, розширюючись та доповнюючись. З 1988 по 2000 роки з'явився новий підхід, при якому тести були орієнтовані на демонстрацію відповідності ПЗ його специфікації, виявлення помилок та запобігання виникненню дефектів. У цей час визначення методів тестування стало ключовим. В останнє десятиліття XX століття також стало поширеним дослідницьке тестування, коли тестувальник вивчав і намагався якнайглибше зрозуміти досліджуване ПЗ, прагнучи виявити більше помилок.

У 1980-х з'явилися перші програмні засоби для проведення тестування у автоматизованому режимі. Прогнозувалося, що ПК здатен виконати значно більше число тестів, на противагу людині, і виконає він це надійніше значно. На початку такі інструменти були надто простими і не володіли змогою написання скриптових сценаріїв.

У 1990-х роках спостерігався перехід від простого тестування до складнішого. Тести торкалися дедалі більше етапів, і виник такий напрям, як "забезпечення якості " (quality assurance). Воно включає різноманітні етапи, такі як план, проєкт, формування, підтримання та проведення тестів. Саме тоді починають створюватися різноманітні програмні продукти для підтримання процесу власне тестування: досконаліші середовища автоматизації із перспективою написання

скриптів та формування звітів, а також програми для управління тестами, ПЗ для, властиво, навантаження тестування. У такий спосіб тестування перейшло на новий рівень, сприяючи розвитку методологій та появі ефективних інструментів управління процесом тестування та автоматизації, які, і стали спадкоємцями сьогоднішніх інструментів автоматизації.

У середині 1990-х років із прогресом Інтернету і створенням значного числа веб-застосунків надзвичайну популярність стало отримувати «гнучке тестування» (за подібністю із гнучкими методологіями програмування). Через такий підхід і виникла тісна співпраця між усіма членами команди розробки. Команди тестування та розробки стали все більш багатофункціональними, а тестувальники та програмісти тісно співпрацюють один з одним, обмінюючись знаннями та досвідом. Іноді вони можуть доповнювати та замінювати один одного, що сприяє максимально швидкій та продуктивній роботі. Також не варто забувати і про дизайн-тестування, даний метод був досить популярний у цей час і призначався для перевірки ПЗ на етапах планування, проєктування, розробки та підтримки.

Ще однією не менш важливою подією стала поява ємнісного тестування. При цьому виді тестування ПЗ навантажували та завантажували спеціально розробленими інструментами. ПЗ тестували на стійкість до відмов та можливість швидко обробляти велику кількість запитів в обмежений час. Це стало переломним моментом в історії тестування, даний вигляд залишається актуальним і в даний час, і називається тестуванням навантаження.

В наш час все більше уваги приділяється автоматизації тестування. Автоматизація – це найважливіший елемент сучасного тестування, який прискорює та підвищує ефективність тестових процесів. Використання автоматизованих засобів дозволяє людині прискорити ручні монотонні тести. Автоматизація дає змогу позбутися певних рутинних операцій і значно пришвидшити реалізацію тестів, значні ресурси можуть йти власне на оновлення самих тестів. Прогрес автоматизації вивів якість на новий рівень, зникла необхідність перевіряти ті самі тестові сценарії після невеликих змін у коді програми.

У світі використання ШІ стає дедалі популярнішим і тестування не

залишилося осторонь цієї тенденції. Використання ШІ для процесу тестування дає безліч переваг. ШІ забезпечує налаштування тестів, виконання тестових сценаріїв щоразу з однаковою точністю та виключає людський фактор. ШІ може використовувати менше ресурсів для точного пошуку помилок. Тестування може проводитися в найкоротші терміни за декількома сценаріями та різними умовами одночасно, що значно зменшує час, витрачений на тести, отже, знизить витрати на розробку. Все більше використання ШІ покращить обсяг тестування програмних продуктів і дозволить виконувати тести в різних умовах і сценаріях. Це дозволить тестувальникам більш ефективно виявляти та виправляти помилки.

1.2 Постановка завдання та цілі роботи

Необхідно дослідити засоби автоматизованого тестування для проведення автоматизованого тестування веб-застосунку калькулятора.

Калькулятор, доступний за посиланням calculator.apps.chrome, є веб-застосунком для виконання різних математичних розрахунків. Він має простий і зручний інтерфейс, що підтримує основні арифметичні операції, та складніші складні функції, зокрема тригонометричні, логарифми та піднесення до степеня. Калькулятор доступний безпосередньо з браузера, що робить його зручним для швидких розрахунків без встановлення додаткового ПЗ.

Головним завданням кваліфікаційної роботи є реалізація тестового сценарію досліджуваного калькулятора, за допомогою одного з розглянутих інструментів автоматизації.

Початкові вимоги до проведення тестування:

– користувача авторизовано в ПК;

– усі додаткові програми, які можуть впливати на роботу тестування, закриті;

– браузер закрито.

Стандартне найпростіше завдання:

– запустити браузер Google Chrome;

– зайти на сайт https://calculator.apps.chrome/ ;
– за допомогою кнопок виконати найпростішу арифметичну дію;
– порівняти очікуваний та фактичний результати;
– закрити браузер.

Для прикладу використовується задача, найбільш проста і одночасно, наближена на кшталт завдань, що виконуються в реальних авто-тестах. Це завдання максимально наближене до реального. І головним критерієм вибору такого тестового сценарію є комерційна таємниця ПЗ, що розробляється.

1.3 Визначення тестування

Тестування – це діяльність, яка здійснюється для оцінки та покращення якості ПЗ [1]. Тестування ґрунтується на виявленні дефектів, багів або проблем у програмній системі. Також тестування торкається всіх сфер розробки ПЗ, як показано на рисунку 1.1 [2].

Тестування створює незалежну платформу для оцінки, засновану на конкретних параметрах та вимогах, що сприяє попередженню потенційних проблем та підвищенню впевненості у якості ПЗ, що розробляється.

Тестування ПЗ включає процес виконання окремих компонентів ПЗ або цілих систем з метою оцінки їх відповідності певним критеріям і вимогам. Ці критерії включають:

– дотримання вимог, встановлених під час проєктування і створення ПЗ;
– коректна обробка різних типів вхідних даних;
– виконання функцій у межах прийнятного часу;
– забезпечує зручність використання для кінцевого користувача;
– можливість встановлення та коректної роботи в різних операційних середовищах;
– досягнення загального бажаного результату, що відповідає потребам усіх зацікавлених сторін;
– ефективне забезпечення безпеки даних та захисту від загроз;

– мінімізація помилок та виявлення потенційних уразливостей;

– сумісність з іншими програмними продуктами та системами;

– відповідність стандартам та регулятивним вимогам;

– гнучкість і масштабованість для адаптації до умов і вимог, що змінюються.

Рисунок 1.1 – Оточення тестування

Основна мета тестування - не тільки знайти і виявити помилки та недоліки, але й забезпечити високу якість і надійність програмного продукту, що розробляється, перед його впровадженням та використанням споживачем.

Тестування ПЗ є ітеративним процесом, оскільки виправлення однієї помилки може призвести до виявлення інших, складніших проблем або навіть викликати появу нових. Тестувальники вибирають стратегії, які найкраще відповідають функціям та вимогам програми або програми. Ці стратегії можуть містити такі тестування:

– ручне;
– автоматизоване;
– «білої та чорної скриньки»;
– інтеграції, систем, виробництва та безпеки.

Зрештою добре спланований та виконаний процес тестування підвищує якість ПЗ та знижує ймовірність його збоїв під час роботи.

Баги є помилками, дефектами чи недоліками ПЗ, які блокують систему або перешкоджають коректній роботі. Баги здатні породжуватися різноманітними причинами, як то помилки кодування, неграмотне планування, складністю сумісності і ін. [3].

При розгляді тестування також варто згадати і різні етапи зі створення та, відповідно, тестування програмних продуктів, показано на рисунку 1.2.

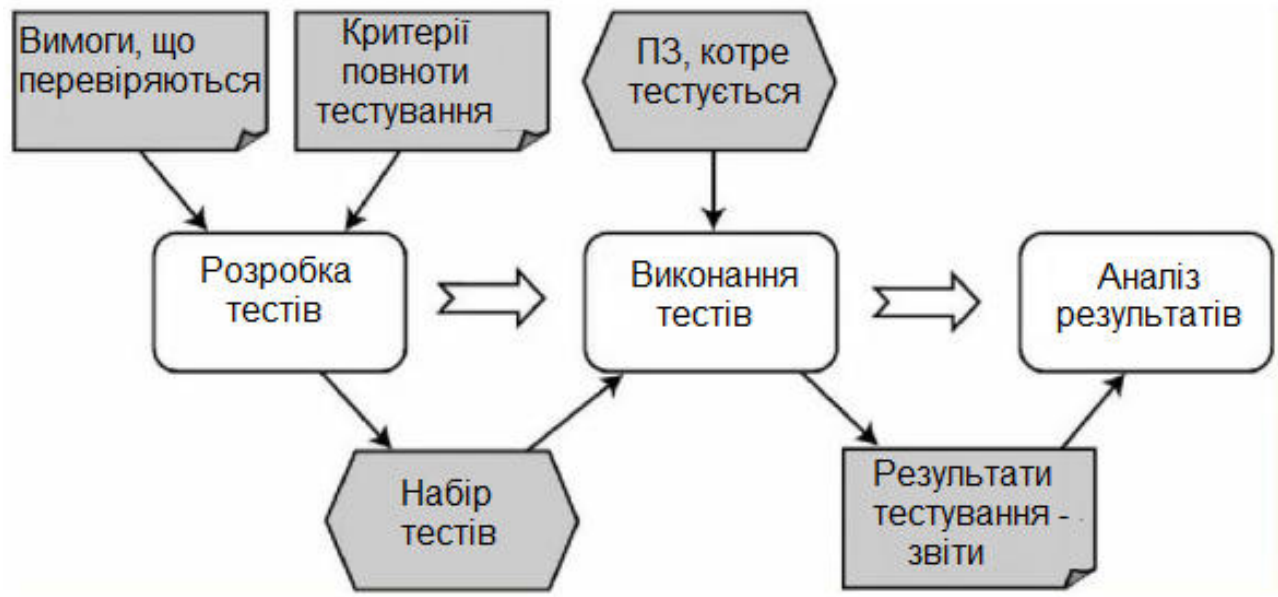

Рисунок 1.2 –Процес розробки та тестування ПЗ

Опрацювання вимог. Потрібно ознайомитися із вимогами, котрі ставить замовник ПЗ. Обговорити, як має власне виглядати фінальний продукт.

Формування тестувальної стратегії. Треба виконати оцінку періоду тестування, визначення середовища для здійснення тестування, узгодження всієї інформації, котра була одержана під час попереднього етапу.

Написання документації тестування. Створення сценаріїв, котрі дають змогу перевірити функціонал розроблюваного ПЗ.

Тестування прототипу. Тестуються основні функціональні можливості ПЗ, коригуються цілі, шляхом додавання необхідних функцій.

Основне тестування. Проходить загальна перевірка ПЗ.

Стабілізація. Тут проходять роботи над видаленням багів коду.

Експлуатація. Здійснюється легке тестування, виправлення помилок, які знайшов кінцевий користувач.

Автоматизоване тестування ПЗ - це процес тестування, в якому основні функції та етапи виконання тесту, такі як запуск, ініціалізація, виконання, аналіз та надання кінцевих результатів користувачеві виконуються автоматично за допомогою інструментів автоматизації [4].

При цьому інструмент для автоматизованого тестування - це програма, яка використовується для створення, налагодження та виконання тестових сценаріїв та аналізу результатів їх виконання.

Перевірка програмних систем є важливою частиною динамічної верифікації, де використовується обмежена кількість тестів для перевірки ПЗ. У цьому вся процесі вибираються тестові приклади зі стандартних моделей поведінки, типових цій галузі застосування, і перевіряється їх відповідність очікуваному поведінці системи.

1.4 Принципи тестування ПЗ

В галузі тестування ПЗ існують сім базових принципів, які відіграють ключову роль при розробці та оцінці якості програмних продуктів:

1. Невичерпність тестування: Цей принцип підкреслює, що, незважаючи на зусилля, повне охоплення всіх можливих варіантів та сценаріїв тестування неможливе через нескінченну множину потенційних сценаріїв використання та станів системи.

2. Виявлення дефектів: Тестування спрямоване виявлення дефектів і

невідповідностей, а чи не доказ їх відсутності. У процесі тестування виявляються конкретні дефекти чи невідповідності, підкреслюючи, що відсутність виявлених дефектів гарантує відсутність помилок у системі.

3. Ілюзія безпомилковості: Важливо розуміти, що навіть при виявленні та виправленні певної кількості дефектів неможливо гарантувати абсолютну бездоганність програмного продукту через природу ПЗ та обмеження часу та ресурсів, доступних для тестування.

4. Раннє тестування: Цей принцип наголошує на необхідності розпочати тестування на ранніх етапах життєвого циклу розробки, що дозволяє виявляти та виправляти дефекти на більш ранніх етапах, що економить час та ресурси.

5. Кластеризація дефектів: Існує тенденція до групування дефектів у певних модулях або компонентах системи, часто звана "кластеризацією дефектів". Це може бути обумовлено складністю та заплутаністю певних ділянок коду або ефектом ланцюгової реакції при внесенні змін.

6. Тестування залежить від контексту: Цей принцип підкреслює, що методи та підходи до тестування повинні відповідати контексту та особливостям конкретного типу програмного продукту, включаючи його цілі, тип, команду розробників, доступні інструменти, терміни та очікуваний рівень якості.

7. Парадокс пестициду. Принцип вказує на необхідність постійного оновлення та модифікації тестових сценаріїв та кейсів, оскільки застосування одного і того ж набору тестів протягом тривалого часу здатне спричинити втрати їх ефективності у виявленні нових дефектів.

1.5 Висновок до першого розділу

Досліджено предметну область, зокрема наведена історія появи тестування ПЗ.

Виконано постановку задачі та наведено основні цілі дослідження. Описано саме поняття тестування та сім базових його принципів.

У сучасному світі розробка ПЗ успішно розвивається. З'являються все більш

складні бібліотеки та програми, що вимагають відповідних змін та розвитку в галузі тестування. З огляду на швидкі темпи розвитку технологій потрібно постійно покращувати існуючі підходи та методи тестування, а також створювати нові способи забезпечення якості ПЗ. Важливо рухатися за поточними тенденціями і прагнути постійного розвитку у цій галузі.

2 ТЕОРЕТИЧНА ЧАСТИНА

Для проведення подальшого дослідження необхідно описати види і різні типи тестування. Особливу увагу варто приділити трьом рівням автоматизації тестування (Unit-тести, функціональні тести, UI тести).

Потрібно провести докладний аналіз основних програмних інструментів для автоматизації тестування, таких як Playwright, Puppeteer, Cypress, Selenium Webdriver.

2.1 Види тестування

У рамках тестування виділяють множину видів та методів тестування. Кожен вид тестування спрямований на перевірку певного функціоналу. На рисунку 2.1 наведено схему основних видів тестування.

Функціональне тестування — це метод, у якого тести створюються з урахуванням технічного завдання, те, з чим кінцевий споживач взаємодіятиме напряму. Цей підхід до тестування ПЗ не включає докладне вивчення внутрішньої структури програми, характерне для методу «білої скриньки», а сконцентрований на перевірці вхідних даних та аналізі вихідних результатів [2].

Функціональне тестування описує очікувану поведінку системи, спираючись на аналіз специфікацій функціональності компонента чи всієї системи загалом. Воно не обмежується тестуванням окремих функцій чи методів, модуля чи класу, а скоріше оцінює попередньо певну поведінку програмного елемента або системи у загальному. Функціональне тестування аналізує наперед визначену поведінку і базується на працездатності системи загалом.

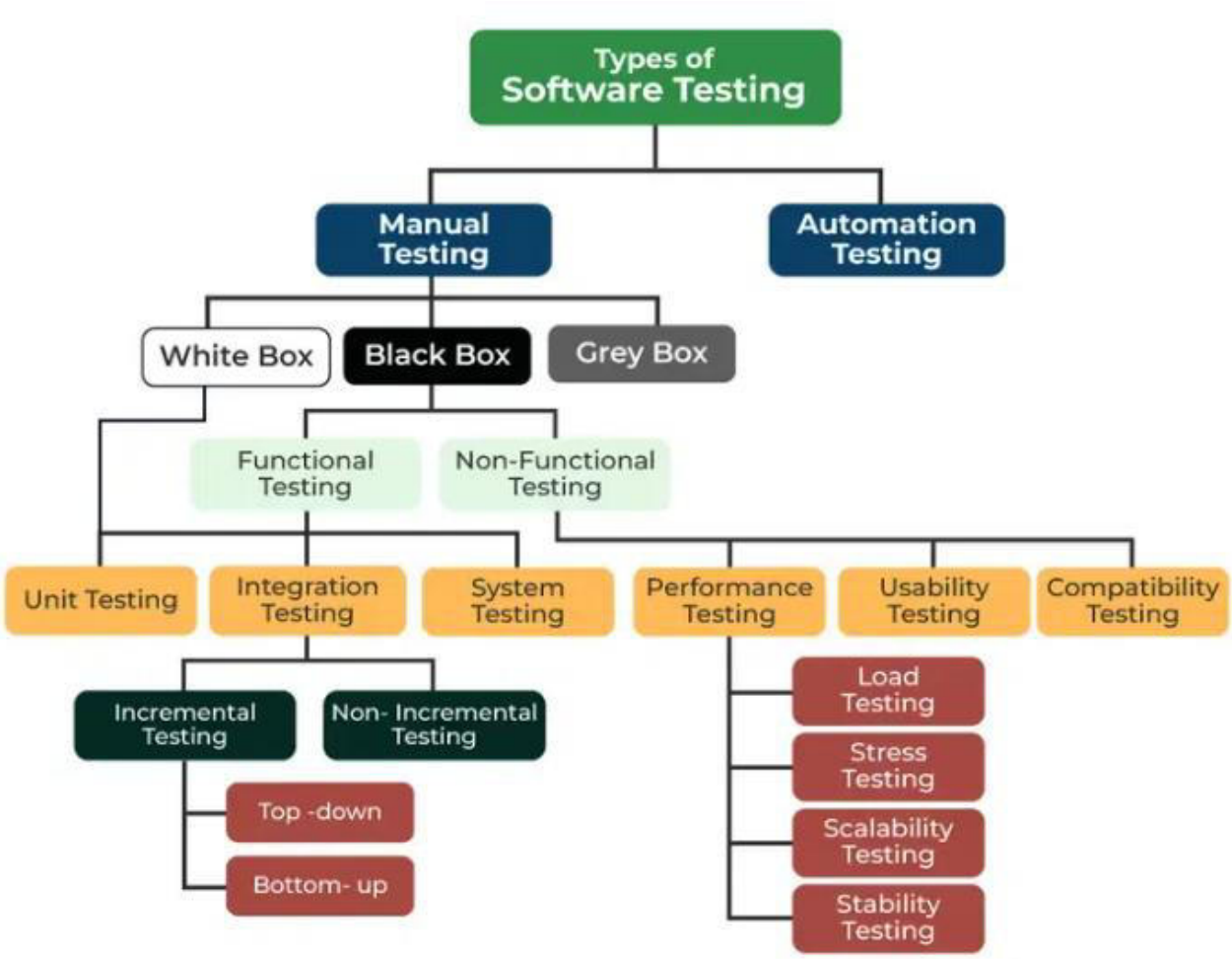

Рисунок 2.1 – Види тестування

Функціональне тестування включає:

– працездатність програми як функції;
– коректність роботи програми;
– взаємодія програми з іншими системами чи компонентами;
– відповідність прийнятим специфікаціям та правилам;
– захищеність від загроз та уразливостей;
– тестування продуктивності.

У сфері розробки ПЗ, тестування продуктивності здійснюється з метою оцінки роботи системи, що тестується, з точки зору її оперативності та стабільності при певних навантаженнях. Цей вид тестування також дозволяє досліджувати, вимірювати та перевіряти інші якісні характеристики системи, такі як її масштабованість, надійність та ефективне використання ресурсів.

Існують два основні підходи до проведення тестування продуктивності ПЗ:

– ґрунтується на робочому навантаженні, при якому ПЗ піддається тестуванню в умовах, що відповідають різним сценаріям використання;

– базується на бета-тестуванні, при якому система тестується реальними кінцевими користувачами в реальних умовах експлуатації.

Розглянемо види тестування продуктивності:

Навантажувальне тестування. Є однією з базових форм перевірки продуктивності ПЗ. Цей вид тестування спрямований на аналіз поведінки компонентів або системи під навантаженням, що збільшується. Під навантаженням розуміється очікувана кількість одночасних користувачів, які звертаються до додатку, і виконують певну кількість операцій у певний період. Навантажувальне тестування спрямовано визначення часу відгуку важливих операцій бізнес-процесів програми. У ході випробувань також контролюються бази даних, сервери застосунків та інше обладнання, які впливають на роботу ПЗ.

Стрес-тестування. Спрямоване на аналіз працездатності та стійкості системи в умовах екстремальних навантажень. Цей вид тестування розглядається як ефективний метод оцінки здатності системи впоратися зі збільшенням навантаження до критичних значень, забезпечуючи при цьому збереження доступності та коректну обробку поставлених завдань за умов, що перевищують типові експлуатаційні параметри. Завдяки стрес- тестуванню відбувається виявлення вразливостей, що забезпечують додатковий захист критично важливих компонентів ПЗ.

Тестування стабільності. Застосовується з метою оцінки стійкості програмного застосунку перебувати під передбачуваним навантаженням протягом тривалого інтервалу часу. Крім того, для даного виду тестування здійснюється моніторинг використання оперативної пам'яті з метою виявлення деградації продуктивності, що проявляється у зниженні швидкості обробки інформації, збільшенні часу реакції та відгук програми на дані, що вводяться.

Конфігураційне тестування. Постає як типовий спосіб оцінки продуктивності, проте у ньому є значна відмінність із іншими способами. Тут основна увага приділяється впливу різноманітних конфігурацій випробуваного ПЗ,

різних пристроїв та операційних систем. У цьому контексті важливо не тільки визначити, як застосунок буде працювати в ідеальних умовах, але і як він поводитиметься при різних налаштуваннях його оточення.

Юзабіліті – тестування. Є методом перевірки ПЗ, яке використовується для того, щоб зрозуміти чи ергономічно випробуване ПЗ чи ні. Тобто тестування дозволяє відповісти на запитання: чи "зручно" користуватися даним додатком кінцевому користувачеві? Перевірка ергономічності визначається однією або кількох об'єктах тестованої системи, тоді як взаємодія людини і комп'ютера, формує універсальні стандартні принципи.

Тестування безпеки. Спеціалізується на забезпеченні безпеки ПЗ. У контексті загальної загрози системам ПЗ від боку зловмисників, таких як хакери, конкуренти та інші, дане тестування покликане проводити перевірку стійкості ПЗ, що тестується, перед впливом шкідливих атак і програм. Цей вид тестування необхідний для детектування вразливостей і забезпечення захисту ПЗ від небажаних впливів.

Тестування локалізації. Випробування локалізації відображає процес тестування ПЗ на відповідність різним мовам, культурним особливостям, а також регіональним та місцевим контекстам. Крім того, враховується вплив менталітету користувачів на локалізацію, що потребує включення відповідних сценаріїв та логічних ланцюжків у додатках. Таке тестування відіграє ключову роль у забезпеченні адаптації ПЗ до різних культурних та локальних контекстів, що зрештою сприяє підвищенню зручності використання та рівня задоволеності користувачів.

Тестування сумісності. Пов'язане з інтеграцією випробуваного ПЗ з іншими системами або окремими застосунками, є процесом перевірки спільної роботи цих систем. Воно спрямоване на виявлення потенційних конфліктів та збоїв, які можуть виникнути при взаємодії між компонентами ПЗ та зовнішніми системами. Цей вид тестування необхідний для забезпечення ефективної роботи всієї системи загалом і запобігання можливих негативних наслідків від неправильної інтеграції компонентів.

2.2 Типи тестування

Розглянемо типи тестування знання коду (рисунок 2.2):

– “чорної скриньки”;

– “білої скриньки”;

– "сірої скриньки".

Основна мета тестування методом “чорної скриньки” полягає у дослідженні невідомих аспектів роботи системи. Це стосується, перш за все, її внутрішньої будови. При цьому тестувальник, всупереч традиційному уявленню про тестування, бере на себе роль типового кінцевого користувача, проводячи тестування, спираючись на технічні специфікації, а також інші супутні документи, що деталізують функціональні аспекти об'єкта, що тестується.

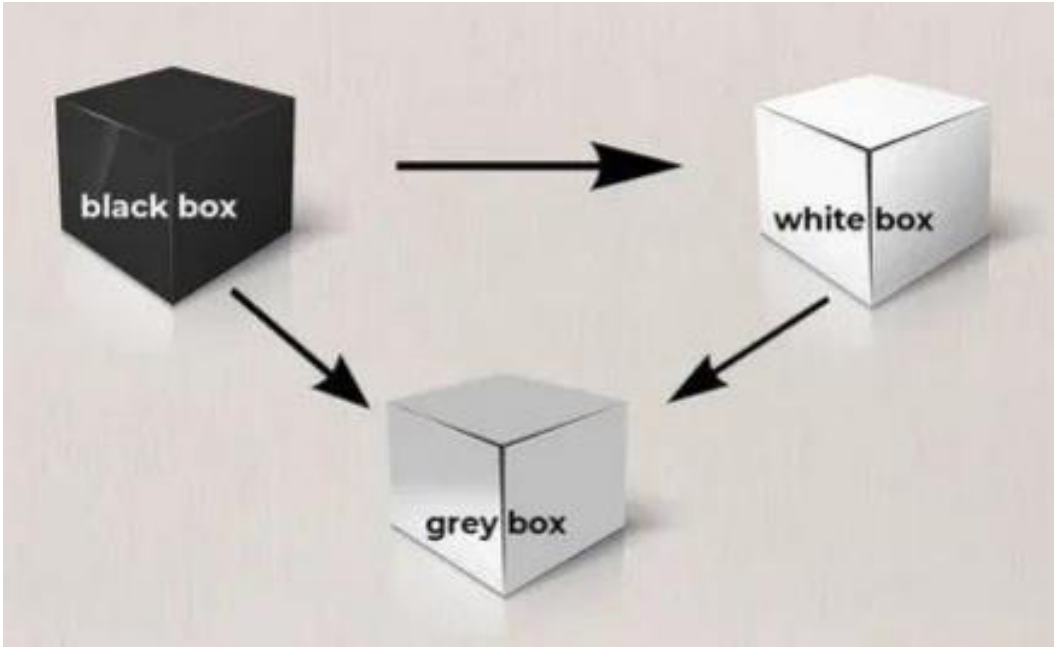

Рисунок 2.2 – Типи тестування за знанням коду

Отже, процес розробки ідей тестування відбувається з урахуванням очікуваних моделей поведінки користувачів. Цей підхід часто визначається як поведінковий, оскільки тестувальник прагне відтворити типові сценарії взаємодії з продуктом виходячи за межі суто технічних аспектів його

функціонування.

До плюсів цього типу можна віднести:

– об'єктивність. Тестування "чорної скриньки" дозволяє оцінювати систему виключно за її зовнішньою поведінкою, що робить процес об'єктивнішим. Перспектива користувача - тестувальники можуть емулювати дії звичайних користувачів, що допомагає виявити проблеми, з якими вони можуть зіткнутися в процесі використання продукту;

– простота. Таке тестування не вимагає глибокого знання внутрішньої будови системи, що робить його доступним для значного числа спеціалістів.

Варто вказати і недоліки такого тестування:

– обмежене охоплення. Оскільки тестування проводиться виключно на основі зовнішніх характеристик системи, деякі внутрішні дефекти можуть залишитися непоміченими;

– недолік деталізації. Тестування "чорної скриньки" може втратити тонкі нюанси функціонування системи, які можуть бути важливими для її коректної роботи;

– залежність від документації. Ефективність цього методу тестування залежить від якості та повноти технічної документації про продукт. У разі її відсутності чи неповноти виникають складності розробки тест-кейсів;

– обмежений контроль над тестовим процесом. Тестувальники можуть обмежуватися лише тими сценаріями, які передбачені у специфікаціях чи інших документах, та пропускати важливі аспекти функціонування системи, які не були документовані.

“Біла скринька” є повним антиподом до чорної. В процесі тестування чорної скриньки потрібно запуск програми та спостереження за її діями, у той час як при використанні методу білої скриньки цього не потрібно. Достатньо вивчити код програми.

Тестування методом “білої скриньки” (також відоме як прозоре, відкрите, скляне тестування або структурне тестування) є методикою перевірки ПЗ, яка передбачає, що тестувальнику відома внутрішня структура, будова та реалізація

системи При використанні цього методу вибираємо вхідні значення, спираючись на знання про код, який їх оброблятиме. Крім того, є уявлення про те, яким має бути результат цієї обробки. Розуміння всіх особливостей тестованої програми та її реалізації є обов'язковим застосування цієї методики. Тестування «білої скриньки» передбачає глибоке занурення у внутрішній пристрій системи, виходячи поза її зовнішніх інтерфейсів.

Переваги цього методу:

– аналіз коду дозволяє виявляти приховані за інтерфейсом проблеми з ПЗ;

– тестування починається на ранніх етапах розробки, що дозволяє поліпшити якість продукту;

– має розгалужену систему метрик, зібрати і проаналізувати які можна з легкістю автоматизувати;

– сприяє написанню якісного коду та додаткового проходження етапу код-ревью;

– багато підходів даного методу підтверджують свою ефективність, ґрунтуючись на суворих технічних принципах.

Недоліки:

– зростають вимоги до навичок та вмінь тестувальників;

– аналізує роботу програми без реального середовища виконання, ігноруючи його вплив;

– аналізує функціонування програми поза контекстом реальних сценаріїв, у яких його використовують користувачі.

“Сіра скринька” зі свого боку поєднує у собі два методу описані раніше. У нашому розпорядженні лише обмежене уявлення про внутрішню будову програми. Є доступ до деяких аспектів внутрішньої структури та алгоритмів роботи ПЗ для створення найбільш ефективних тест-кейсів. Однак сам процес тестування виконується з використанням методу чорної скриньки, тобто з погляду користувача.

Цей підхід до тестування також відомий як метод напівпрозорої скриньки: є

доступ до деяких аспектів, але не до всіх.

2.3 За ступенем автоматизації

Розглянемо найбільш важливий розділ тестування в рамках даної роботи: ручне та автоматизоване.

Ручне тестування має на увазі ручне виконання перевірок без використання автоматизованих інструментів. Цей вид тестування є найпростішим – низькорівневим та не потребує спеціальних знань [5].

Однак перед прийняттям рішення про автоматизацію необхідно провести серію ручних тестів для оцінки можливості застосування автоматизації. Потрібно витратити значні зусилля на ручне тестування, щоб переконатися у доцільності автоматизації. Важливо пам'ятати, що повністю автоматизувати тестування неможливо в принципі, тому ручне тестування необхідне у певних випадках.

На жаль, для використання засобів автоматизації потрібні спеціалізовані знання та професійна підготовка. Повна автоматизація також недосяжна, тому в деяких випадках у тест-кейсах є і ручне тестування.

Розглянемо основні переваги при використанні автоматизованого тестування:

– виняток людського фактору. Це важлива перевага, оскільки жоден із нас не застрахований від можливих помилок. Автоматизований тест виконується без ризику здійснення випадкових перепусток або помилок;

– зменшення часу виконання тестів. Тест-скрипти не вимагають зіставлення дій з інструкціями чи документацією, що робить процес швидким та ефективним;

– скорочення витрат на обслуговування тестів. Підтримка та аналіз автоматизованих тестів вимагають значно менше ресурсів, ніж ручне тестування (рисунок 2.3);

– гнучкість у формуванні звітів. Є можливість налаштувати формат та надсилання звітів відповідно до наших потреб та вимог;

– можливість запуску тестів у час. Це дозволяє розробникам займатися іншими завданнями, доки тести виконуються автоматично.

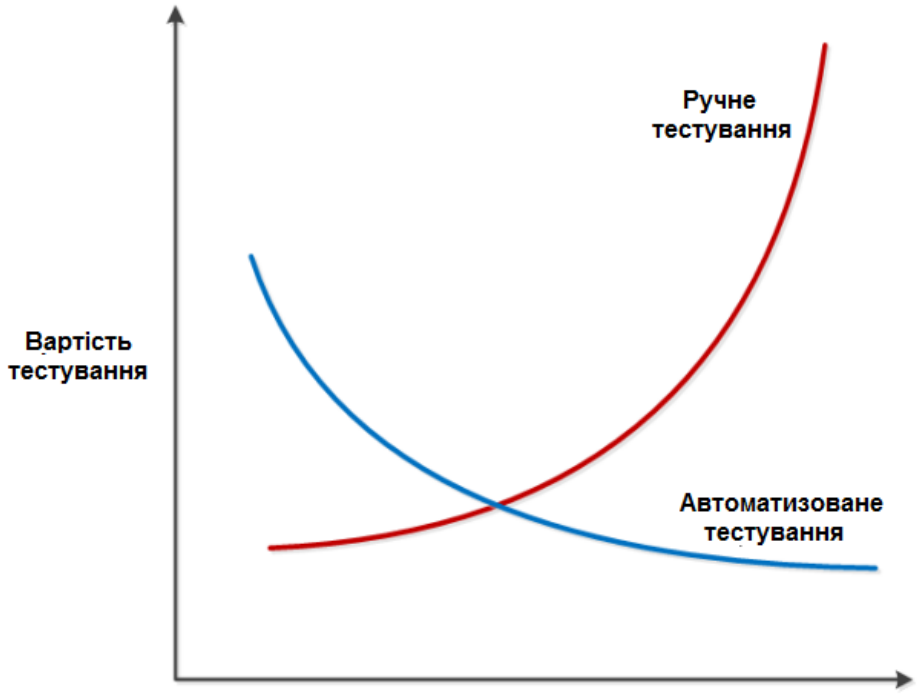

Рисунок 2.3 – Кількість проведених тестів

Однак існують і деякі недоліки:

– одноманітне виконання тестів може призвести до упущення деяких деталей, які були виявлені при ручному тестуванні;

– збільшення витрат на обслуговування через часті зміни в застосунку, що тестується. Значні матеріальні витрати на створення автотестів;

– висока вартість інструментів для автоматизації, особливо під час використання ліцензійного програмного забезпечення;

– необхідність додаткового ПЗ для розширення функціоналу або поліпшення процесу автоматизації.

2.4 Три рівні автоматизації тестування

Для покращення якості продукту, використовують 3 рівні автоматизації тестування. Розглянемо докладніше кожен із цих рівнів (рисунок 2.4):

– Unit-тести;
– функціональні тести;
– тестування інтерфейсу користувача (UI тести).

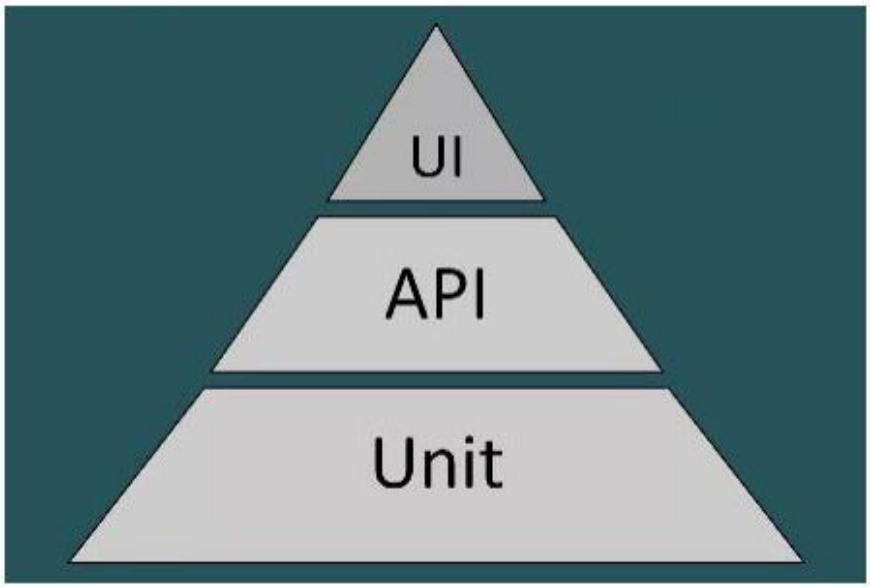

Рисунок 2.4 – Три рівні автоматизації тестування

1. На етапі ініціалізації продукту, фундаменті піраміди автоматизації розробники створюють Unit-тести (або компонентні/модульні перевірки). Цей етап роботи, базис автоматизаційної піраміди, задає основу якісного продукту. Тестувальники, поряд із програмістами, також можуть взяти участь у створенні та налагодженні, пропрацювавши тестові сценарії для оцінки функціональності коду. Надання доступу до тестів на ранніх стадіях розробки та постійне оновлення набору перевірок суттєво знижує ризик виникнення критичних проблем у проєкті, а виправлення знайдених проблем набагато простіше та дешевше.

2. Шар функціонального тестування відіграє важливу роль у забезпеченні якості ПЗ. Функціональне тестування проводиться для перевірки функціональності програми. Воно дозволяє переконатися в тому, що програма працює коректно і виконує функції, що відповідають вимогам користувачів та замовника. Цей вид тестування перевіряє роботу бізнес-логіки програми, незалежно від його інтерфейсу користувача. Це особливо важливо у випадках, коли частина функціональності програми доступна лише через API або back-end.

3. Шар тестування через інтерфейс користувача (UI -тести): На цьому рівні тестувальник перевіряє інтерфейсну частину програми взаємодіючи з продуктом як кінцевий користувач. Такі тести дозволяють знаходити помилки у верстці, у зручності взаємодії та інтуїтивності використання програми.

2.5 Інструменти автоматизації тестування

Існує безліч інструментів і засобів для автоматизації перевірки ПЗ. Вони починаються від мануального кодування сценаріїв до застосування спеціалізованих фреймворків та інтегрованих середовищ розробки.

Одним із найпоширеніших варіантів автоматизації тестування є створення та виконання автоматизованих тестів із застосуванням інструментів. Ці тести можуть зачіпати майже всі типи тестування: функціональні, безпеки, стресостійкості та сумісності програми [6].

Далі розглянемо кілька ключових інструментів для автоматизації, що надають тестувальникам і розробникам широкий спектр можливостей для розробки надійних і ефективних сценаріїв перевірки. Завдяки потужним платформам та інтуїтивно зрозумілим інструментам з графічним інтерфейсом можна спростити та прискорити процес перевірки, підвищити якість та надійність існуючих технологій, а також скоротити час, що витрачається на створення та підтримку всіх перевірок.

2.5.1 Playwright

Playwright — бібліотека, створена командою Microsoft і заснована на асинхронному підході. Вона дозволяє тестувати веб-програми на різних платформах, надає простий та інтуїтивно зрозумілий API для написання тестів, який допомагає легко взаємодіяти з елементами веб-сторінки та виконувати різні дії: кліки, заповнення форм, скролінг та інші [7]. Логотип наведено на рисунку 2.5. Далі розглянемо кросплатформенність різних браузерів.

Рисунок 2.5 – Логотип Playwright

При вивченні структури таких браузерів, як Safari, Chrome та Firefox, можна побачити деякі відмінності у роботі. Це пов’язано із тим, що кожен браузер використовує певні рушії. Наприклад, якщо ви подивитеся на серце браузера Safari, ви побачите, що всередині знаходиться рушій WebKit. Також, якщо ви поглянете на Chrome, то побачите, що його начинки засновані на рушії Chromium. А ядром Firefox є рушій Gecko, як показано на рисунку 2.6.

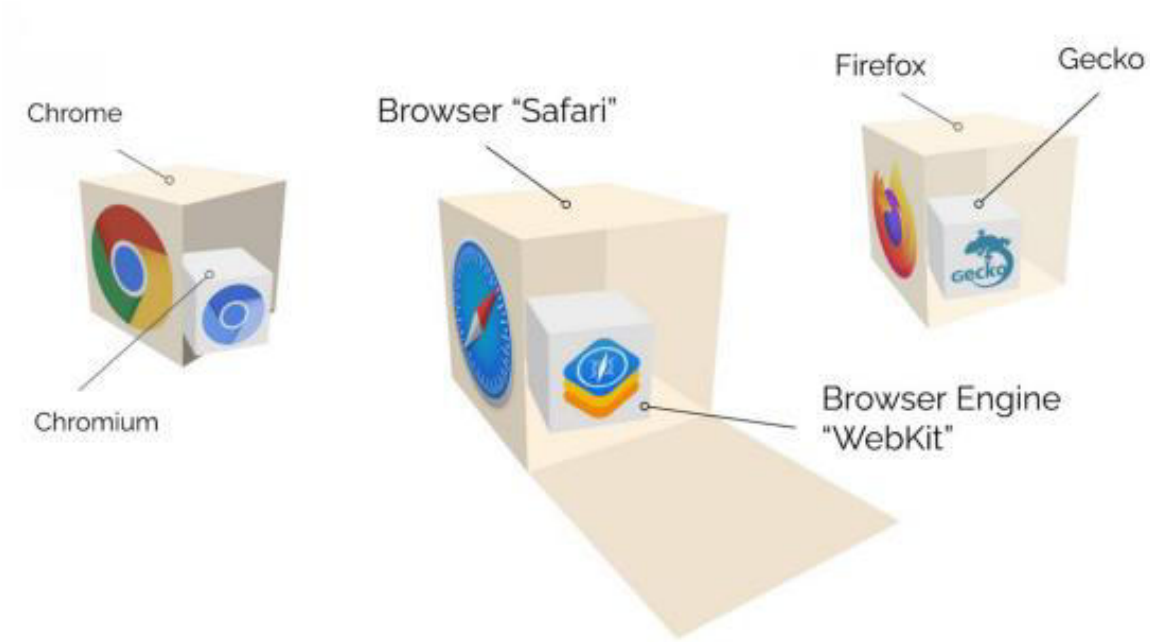

Рисунок 2.6 – Різні браузери та їх рушії

Ці рушії є важливими компонентами, що визначають спосіб відображення веб-сторінок та взаємодії з ними. Ці функції включають HTML і CSS, які безпосередньо впливають на сприйняття користувачем веб сторінок і їх функціональність. JavaScript надає зручний та потужний API для керування вашим браузером та взаємодії з веб-сторінками.

Playwright пропонує своїм користувачам широкий спектр функцій, що дозволяють проводити більш швидке та ефективне наскрізне тестування:

– Codegen. Ця функція дозволяє писати тестові сценарії, не використовуючи код. Функція codegen автоматично записує всі ваші дії, коли ви взаємодієте з браузером;

– інспектор Playwright надає можливість спостерігати за ходом тестування послідовно у реальному режимі, вибирати та редагувати локатори, а також переглядати результати перевірок працездатності у відповідних журналах;

– перегляд трасування, є можливість переміщення між етапами тестування та вивчення того, що відбулося на кожному з них за допомогою скріншотів та журналів;

– автоматичне очікування. Ця функція включає виконання відповідних перевірок для кожного елемента перед виконанням будь-якої дії з ним. Дія буде виконана тільки після того, як система переконається, що всі елементи поводяться належним чином;

– візуалізація та запис результатів тестування. Playwright підтримує скріншоти та запис екрана, які можна переглядати разом із звітами;

– можливість візуального тестування інтерфейсу користувача, яка включає тестування макетів, дизайнів, стилів та інших елементів користувальницького інтерфейсу;

– широкі можливості інтеграції з різними системами сторонніх виробників, включаючи сервери CI \ CD, такі як Jenkins, CircleCI, Azure Pipeline та інші;

– підтримка паралельного виконання тестів дозволяє збільшити швидкість тестування, оскільки тестувальники можуть запускати тести одночасно у кількох браузерах та кількох пристроях.

Всі згадані особливості стали причиною швидкого зростання популярності Playwright.

2.5.2 Puppeteer

Це бібліотека API на базі Node.js, створена для автоматизації браузера Google Chrome серед Chromium. Логотип представлено на рисунку 2.7. Цей інструмент дозволяє перехоплювати процеси вашого веб-браузера безпосередньо та керувати ними через програмний інтерфейс. Це дозволяє автоматизувати великий набір дій, які зазвичай виконуються реальним користувачем у браузері. Цей інструмент допомагає спростити розробку та тестування веб-застосунків [8].

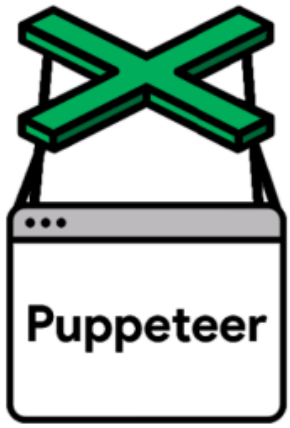

Рисунок 2.7 – Логотип Puppeteer

Принцип функціонування фреймворку Puppeteer показано на рисунку 2.8

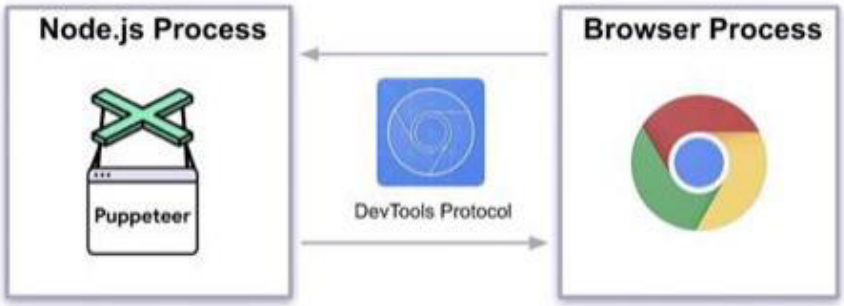

Рисунок 2.8 – Принцип роботи Puppeteer

Переваги:

– багатий API для роботи з мережними запитами та їх перехопленнями;
– наявність можливості суттєвого прискорення тестів за допомогою

Puppeteer Browser Context;

– наявність API для тестування програми в режимі офлайн та ін.

Недоліки:

– за фактом підтримка роботи тільки з одним браузером - Chomium;

– немає вбудованого механізму паралелізації тестів;

– необхідність підключення додаткових бібліотек для повноцінного, всеосяжного тестування.

2.5.3 Cypress

Побудований на базі Node.js і доступний у вигляді npm -модуля, Cypress надає можливість написання автоматизованих тестів лише мовами Java Script та Type Script. Процеси Cypress та Node.js постійно комунікують між собою, синхронізуються, виконують тестові завдання залежно один від одного. Використовуючи Cypress, можна підміняти мережевий трафік браузера та відстежувати відповіді сервера на запити браузера під час тестування. Великим плюсом фреймворку можна вважати велику кількість інтернет-спільноти його користувачів і велику кількість інструкцій та статей від самих розробників платформи, докладна документація. Також платформа дозволяє реалізовувати автоматичне очікування появи елементів веб -сторінок. До недоліків Cypress часто відносять швидкість його роботи, він показує найменші показники продуктивності в порівнянні з аналогічними інструментами. Логотип наведено на рисунку 2.9.

Рисунок 2.9 – Логотип Cypress

Переваги:

– надає чудову документацію і не потребує додаткового налаштування

для встановлення залежностей та бібліотек;

– скріншоти та відеозапис. При виконанні автоматизованих тестів Cypress робить знімки екрану, щоб фахівці QA або розробники могли побачити, що сталося на конкретному етапі, навівши курсор на певні команди в журналі команд та виправити дефекти;

– функції Spies, Stubs та Clocks дозволяють фахівцям QA поведінку відповідей сервера, функцій або таймерів та перевіряти їх;

– підтримка кросс-браузерного тестування. Cypress забезпечує підтримку браузерів Firefox та Edge, завдяки чому команди можуть виконувати тести у цих браузерах.

Cypress виконує тести в режимі реального часу та надає візуальний зворотний зв'язок для внесення покращень [9].

2.5.4 Selenium Webdriver

Впровадження Selenium підтримується продуцентами найбільш популярних браузерів. Їм вдається адаптовувати браузери з метою тіснішої інтеграції із Selenium, часом ж більше того, втілюють вбудовану підтримку. Вигляд логотипу показаний на рисунку 2.10.

Рисунок 2.10 – Лого Selenium

Selenium підтримує і десктопні, і мобільні браузери, також дає змогу створювати сценарії автотестів властиво на будь-якій мові програмування. Із використанням Selenium вдіється влаштовувати розподілені стенди, котрі

містять сотні комп'ютерів із різноманітними браузерами та операційними системами, ба, на додачу, втілювати сценарії у хмарах. WebDriver - це інструмент для автоматизованого тестування веб-додатків, зокрема, для перевірки того, що програма працює відповідно до очікувань. Цей інструмент проєктувався таким чином, щоб мати зручний програмний інтерфейс (API), що дозволяє підвищити читання та спростити підтримку тестів.

Схема обміну даними у Selenium WebDriver наведена на рисунку 2.11.

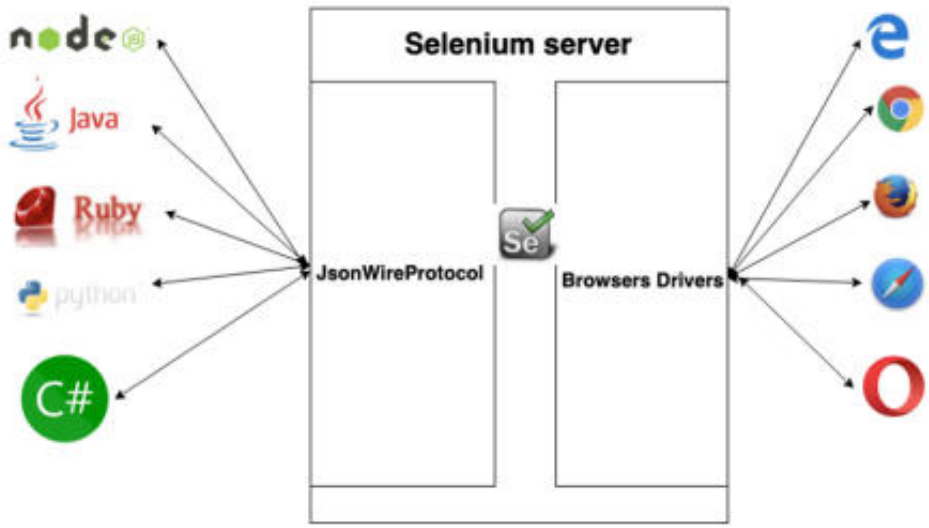

Рисунок 2.11 – Схема обміну даними у Selenium WebDriver

WebDriver API не прив'язаний до жодних тестових фреймворків, Це дозволяє використовувати будь-які фреймворки модульного тестування [10].

2.6 Порівняння та вибір інструментів

При зіставленні ключових інструментів автоматизації було зроблено висновки та складено таблицю 2.1.

Таблиця 2.1 – Порівняння різних платформ автоматизації тестування

Можливості платформи	**Платформа**			
	Selenium	**Puppeteer**	**Cypress**	**Playwright**
Підтримувані мови програмування	Java, C#, PHP, Ruby, Perl, Python	Java, C#, Python, JavaScript, TypeScript	JavaScript, TypeScript	Java, C#, Python, JavaScript, TypeScript
Можливість заміни мережного трафіку	–	–	+	+
Використовувані операційні системи	Для запуску Safari потрібна MacOC, Для запуску IE необхідний Windows	Windows, MacOS, Linux	Windows, MacOS, Linux	Windows, MacOS, Linux
Автоматична генерація коду	З використанням сторонніх плагінів	З використанням сторонніх плагінів	+	+
Підтримка Chromium	+	+	+	+
Підтримка Webkit	+	-	-	+
Підтримка Firefox	+	-	+	+
Спільнота користувачів	+	+	+	+

Selenium — це універсальний інструмент для автоматизації тестування веб-застосунків, що підтримує безліч мов програмування та різних браузерів. Його архітектура заснована на WebDriver, що дозволяє йому взаємодіяти із браузерами безпосередньо через API. Але це робить налаштування та конфігурацію більш складними. Selenium широко інтегрований у різні CI/CD системи, що робить його популярним у корпоративних середовищах розробки. Хоча, продуктивність Selenium може бути не дуже високою при виконанні складних і об'ємних тестів[10].

Puppeteer. Цей фреймворк від Google використовується для автоматизації

тестування веб-інтерфейсів. Робота з платформою передбачає використання браузерів лише сімейства Chromium (Google Chrome, Opera, Vivaldi). Авто-тести, реалізовані за допомогою Pupeteer , можна запускати на 3 операційних системах (Windows, Mac, Linux). Puppeteer пропонує високу продуктивність та простоту у вивченні, але головною перевагою є швидкість та стабільність його роботи в порівнянні з іншими платформами, оскільки фреймворк звертається безпосередньо до браузера через Chrome DevTools Protocol [8].

Cypress орієнтований на тестування сучасних фронтенд-додатків Його архітектура дозволяє легко писати та запускати тести, але підтримка обмежена лише мовами Java Script та Type Script Cypress легко інтегрується із сучасними CI/CD платформами та інструментами для тестування. Головною перевагою є те, що платформа дозволяє реалізовувати автоматичне очікування та появи елементів web –сторінок [9].

Playwright створено лише у 2020 році командою Microsof, яка розробила Puppeteer. Він розширює його можливості, підтримуючи не лише Chrome, а й Firefox та WebKit. Інструмент надає потужний та інтуїтивно зрозумілий API для крос-браузерного тестування, зберігаючи високу продуктивність та стабільність. Playwright легко інтегрується з різними системами CI/CD та підтримує безліч мов програмування, включаючи TypeScript, JavaScript, Python, C# та Java. Це робить його універсальним та зручним інструментом для сучасних проєктів [7]. Також до плюсів фреймворку можна віднести наявність інструменту codegen, що дозволяє створювати скрипти для авто тестів автоматично без написання коду вручну. До недоліків Playwright можна віднести малу кількість навчальних матеріалів та невелику інтернет- спільноту, оскільки платформа є відносно новою.

Таким чином, порівнюючи ці чотири інструменти, можна зробити висновок, що Playwright є найбільш збалансованим та потужним продуктом. Його підтримка багатьох браузерів, висока продуктивність, інтуїтивно зрозумілий інтерфейс і простота налаштування забезпечують найкращі можливості для автоматизації тестування веб-додатків. Таким чином, Playwright є найбільш вдалим варіантом для створення системи автоматизованого тестування калькулятора.

2.7 Висновок до другого розділу

Були описані види та типи тестування. Особливу увагу приділено трьом рівням автоматизації тестування.

Докладно проаналізовані чотири програмні інструменти для автоматизації тестування (Playwright, Puppeteer, Cypress, Selenium Webdriver). Після порівняльного аналізу їз можливостей та характеристик вибір зроблено на користь програмного продукту Playwright.

3 ПРАКТИЧНА ЧАСТИНА

У цій частині кваліфікаційної роботи розділі необхідно описати додаткові технології, котрі застосовуються для втілення процесу автоматизованого тестування. Основна увага приділена редактору коду, СКВ та серверній платформі.

Здійснено практичне втілення ініціалізації проєкту з повним циклу прогону усього тесту.

3.1 Додаткові технології

Для кращої розробки та тестування програмних продуктів потрібне застосування спеціалізованих програмних засобів та інструментів. Найпопулярнішими є IDE – набори програм, покликані спростити процес розробки ПЗ або автоматизувати тестування. Також обов'язковою частиною будь-якої розробки є СКВ, в якій відбувається збереження історії змін і наявна можливість повернутися до минулих варіантів коду. І останніми невід'ємними інструментами є засоби управління проєктами, такі як Jira або Tolredo, у яких команда розробки відстежує та розподіляє обов'язки щодо розробки або тестування програмного продукту.

3.1.1 Редактор коду

Visual Studio Code є потужним ПЗ для розробки, призначене для використання в IDE, що характеризується багатофункціональністю і володіє великим набором інструментів, спрямованих на підвищення продуктивності програмістів та фахівців в галузі автоматизованого тестування.

Інтерфейс програми представлений на рисунку 3.1 [11].

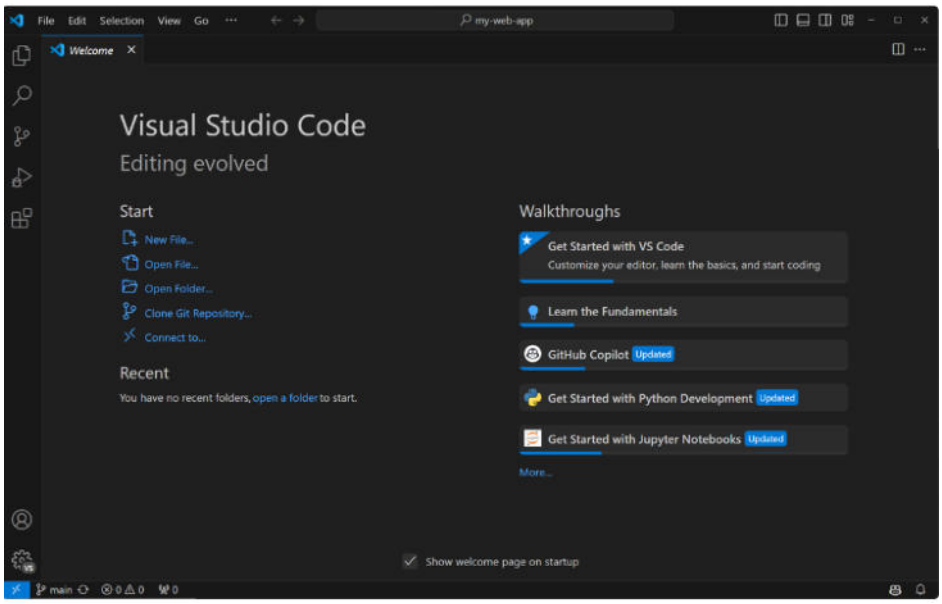

Рисунок 3.1 – Інтерфейс програми Visual Studio Code

Удосконалені можливості рефакторингу дозволяють проводити швидкі та ефективні зміни у вихідному коді програм, а дизайн IDE орієнтований на максимізацію продуктивності, звільняючи розробників від повсякденних рутинних операцій.

3.1.2 Система контролю версій

Крім IDE, широко застосовуються СКВ. Такі системи (Version Control Systems) є програмними рішеннями, призначеними для оптимізації роботи з інформацією, що динамічно модифікується. Вони забезпечують можливість зберігання безлічі версій тих самих файлів чи документів, дозволяють повертатися до попереднім станам даних, і навіть визначати авторство і часові рамки внесених змін.

СКВ поширені виключно у розробці ПЗ для збереження вихідних кодів проєктованої програми. Також їх успішне застосування можливе в інших сферах, де є необхідність в управлінні великим обсягом даних, що постійно змінюються.

У тому числі, такі системи успішно використовуються у САПР, як правило як складові систем керування даними щодо виробу. Керування версіями успішно застосовується у інструментах конфігураційного керування.

Одними із найпопулярніших СКВ залишаються комплекси інструментів Git. Система сконструйована як комплекс програм, котрі демонстративно розроблені беручи до уваги їх застосування у сценаріях створення ПЗ. Вона дає змогу зручно писати спеціалізовані СКВ. Git дозволяє швидкий поділ і , властиво, злиття гілок різних версій одного коду, містить інструменти для візуалізації та навігації із нелінійної історії створення коду. Візуальне розгалуження проєкту розробки головної гілки master показано на рисунку 3.2.

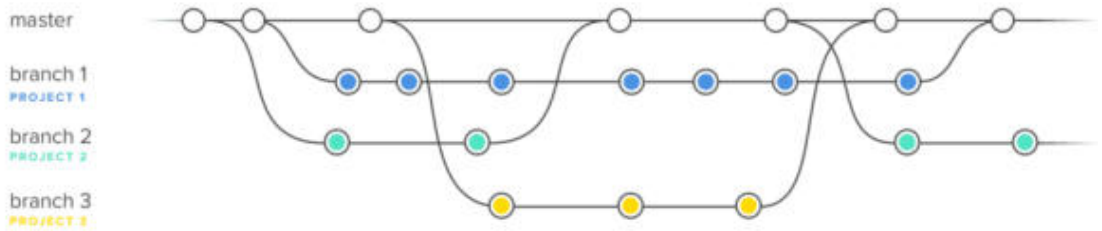

Рисунок 3.2 – Зміна вітки master

Git визначає для кожного розробника або спеціаліста із автоматизованого тестування локальну репліку усієї історії створення коду, усі зміни зберігаються та відтворюються від одного сховища до іншого.

Інструменти GitLab та GitHub широко поширені по всьому світу в командах з розробки, ці інструменти використовують понад 3 мільйони підприємств та організацій. Інтерфейс представлений на рисунку 3.3.

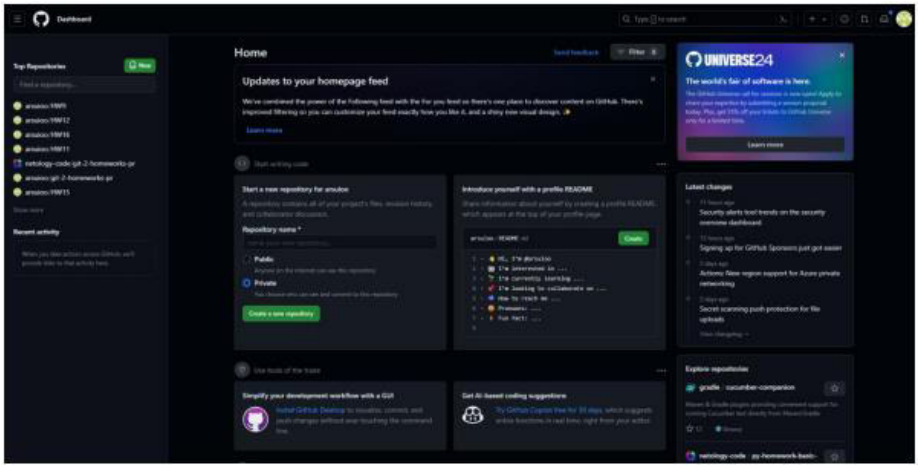

Рисунок 3.3 – Стартове вікно Github

Забезпечується віддалений доступ до сховищ Git через git-daemon, SSH- або HTTP -сервером. TCP -сервіс git-daemon включений до набору програм Git і є спільно із SSH самим поширеним та найнадійнішим методом доступу. Попри ряд обмежень, метод HTTP вельми популярний у контрольованих мережах, оскільки дає змогу застосовувати наявні конфігурації фільтрів мережі [12].

3.1.3 Node.js

Фреймворк Node.js є серверною платформою, котра функціонує на рушію Google Chrome - V8, котрий здатний компілювати JavaScript код у машинний код. Логотип представлений на рисунку 3.4.

Рисунок 3.4 – Логотип NodeJS

Node.js використовує архітектуру подієво-орієнтовану модель і неблокуючу ввід / вивід архітектуру, що робить його легковаговим і ефективним. Це не фреймворк і не бібліотека, це середовище виконання JavaScript.

В рамках середовища виконання Node.js відбувається виконання програмного коду, написаного мовою програмування JavaScript. Це означає, що значна кількість розробників, які вже володіють JavaScript і застосовують його в браузерному середовищі, здатні також розробляти як серверний, так і клієнтський код, не вдаючись до освоєння нового інструменту для занять серверною розробкою.

У контексті мови програмування застосовуються єдині концепції як у браузерному, так і у серверному середовищі. Додатково, Node.js надається можливість швидко переходити на застосування нових стандартів ECMAScript в ході їх впровадження в даній платформі. Цей процес не вимагає очікування оновлення браузерів користувачів, оскільки Node.js є серверним оточенням, яким повністю керує розробник.

Таким чином, нові можливості мови стають доступними відразу після встановлення версії Node.js, яка підтримує ці стандарти [13].

Процес встановлення вищезазначених інструментів, не є складними, всі необхідні компоненти можна завантажити на офіційних сайтах і встановити на персональний комп'ютер. Перейдемо до створення та ініціалізації проєкту.

3.2 Ініціалізація проєкту

Для початку необхідно створити файл із проєктом та проініціалізувати пакет node.js. Вводимо в консоль команду: npm init, що створює файл packagejson у папці проєкту. Цей файл містить важливу інформацію, таку як назва проєкту, його версія, опис, скрипти, залежності та дані автора.

Ініціалізація проєкту наведена на рисунку 3.5.

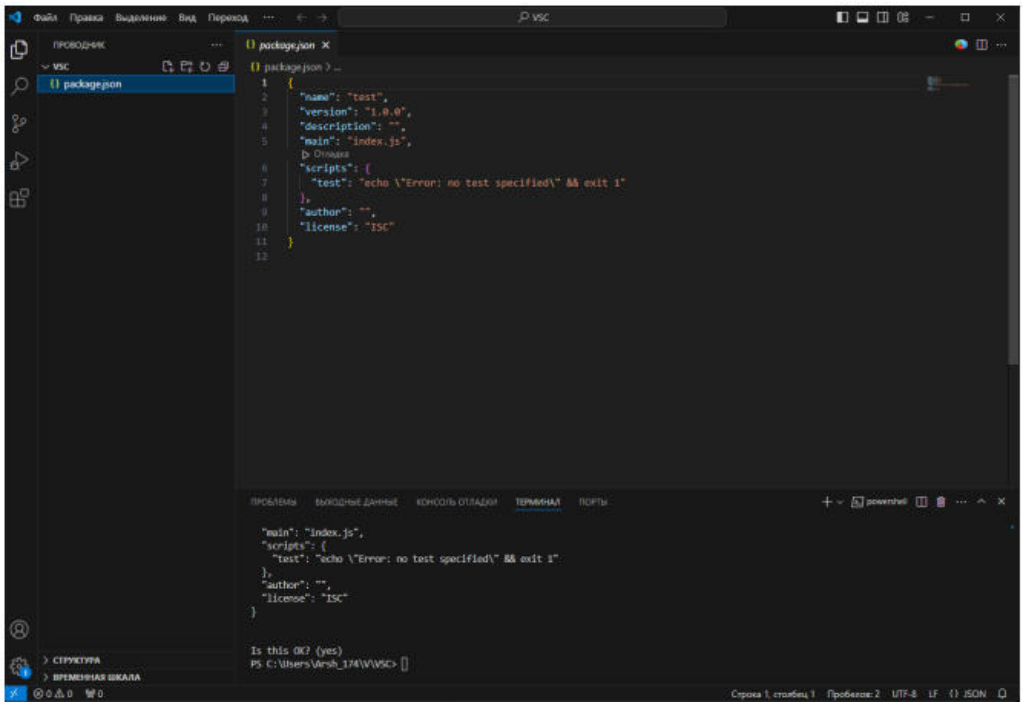

Рисунок 3.5 – Ініціалізація Node.js

Наступним етапом встановимо у створений проєкт Playwright за допомогою команди npm init playwright@latest. Таким чином встановляться всі необхідні залежності, браузери та папки з конфігураціями для налаштування тестів, що наведено на рисунку 3.6.

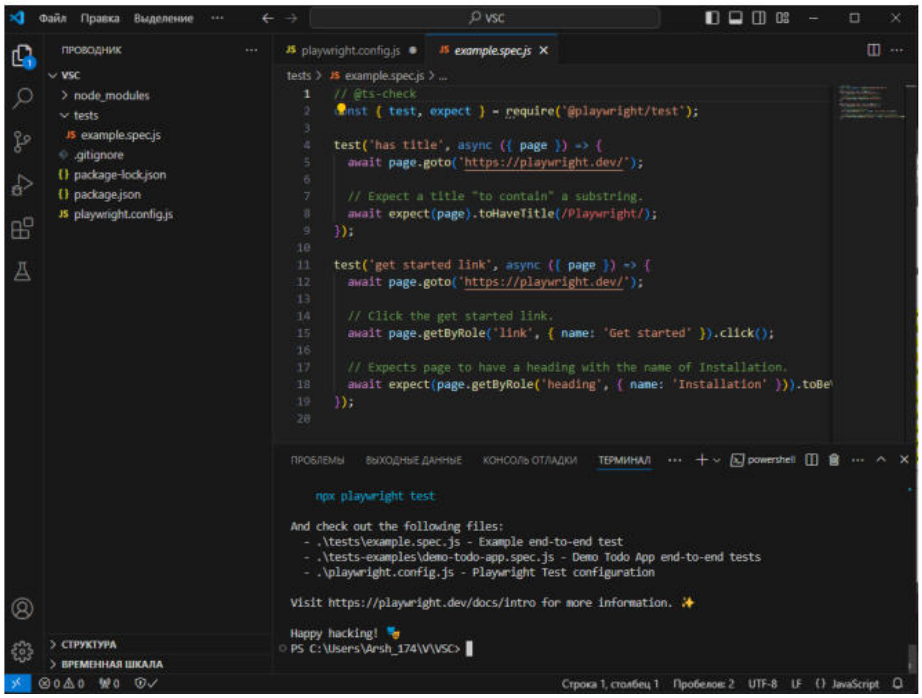

Рисунок 3.6 – Ініціалізація Playwright

На цьому базове налаштування середовища закінчено, далі приступаємо до написання коду авто-тесту та його запуску.

3.3 Повний цикл прогону всього тесту

При переході на сайт програми бачимо рядок введення та виведення отриманого результату, кнопки, що відповідають за символи, арифметичні дії та додаткові функції (рисунок 3.7).

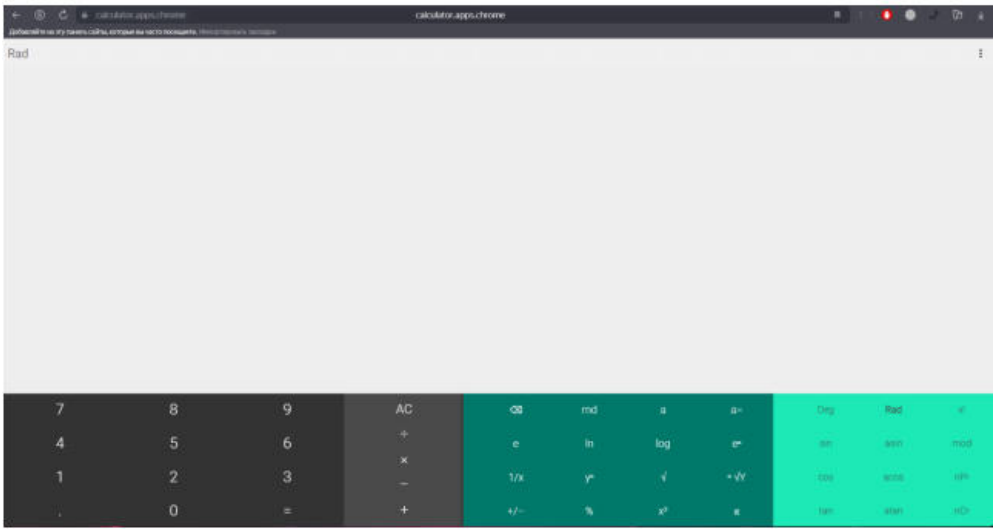

Рисунок 3.7 - Веб-застосунок калькулятора

Для пошуку елементів на сторінці потрібні спеціалізовані засоби для розробників у браузері. Розглянемо їх на прикладі Google Chrome. Вони називаються DevTools (F12). За допомогою селектора знаходимо унікальні локатори кожної необхідної кнопки (рисунок 3.8).

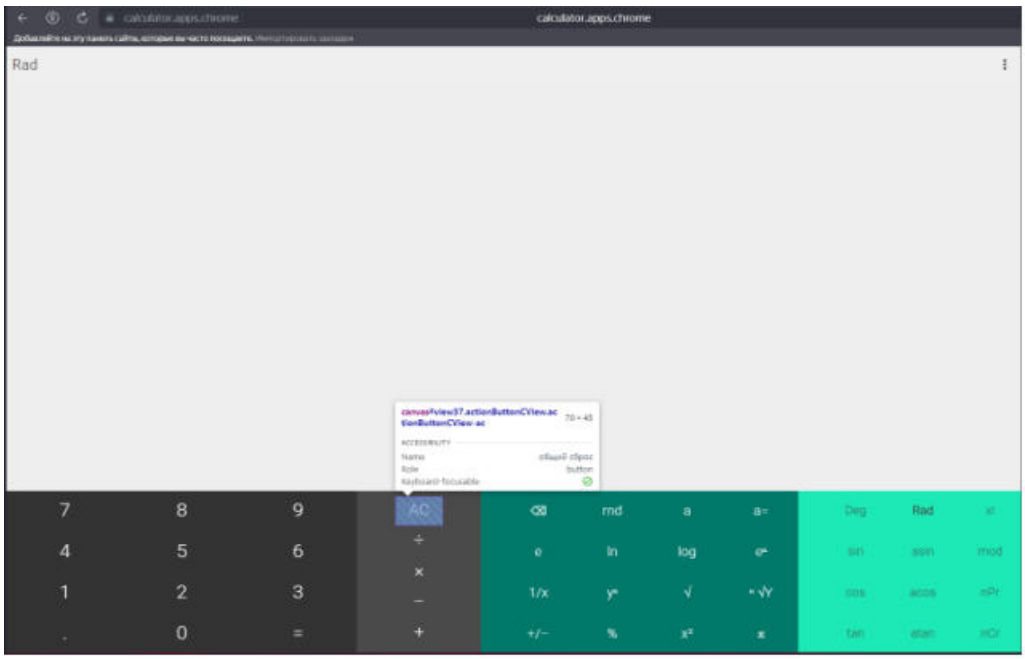

Рисунок 3.8 - Веб-застосунок Калькулятор

На вкладці Elements можна побачити повне дерево HTML тегів та елементів (рисунок 3.9). Для тестів важливо, щоб був лише один елемент взаємодії. Якщо одному локатору будуть відповідати кілька елементів, то тест або взаємодіятиме з першим з них, або відбудеться помилка і весь тест буде провалений.

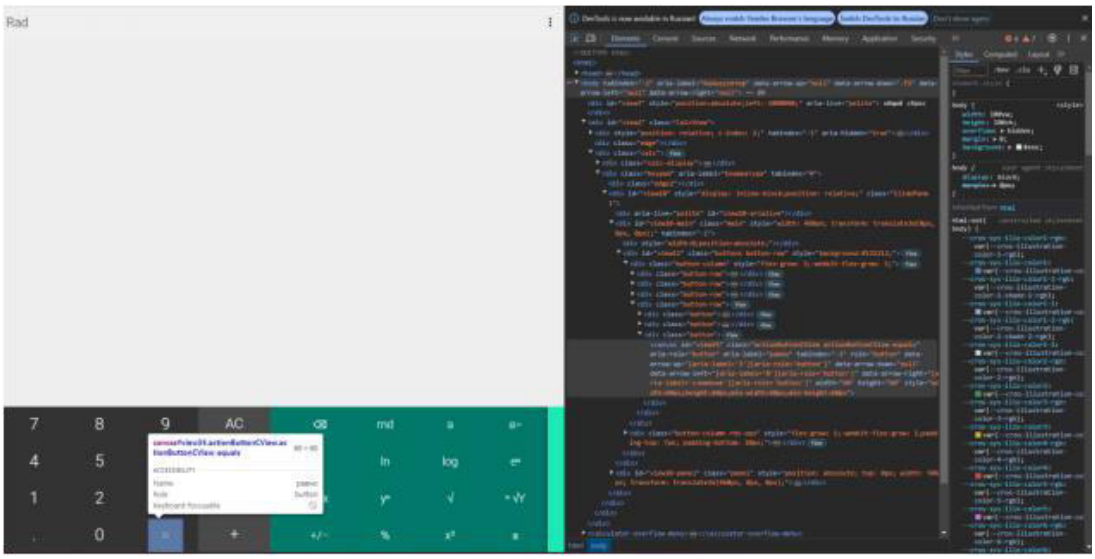

Рисунок 3.9 - Повне дерево HTML тегів та елементів веб-застосунку калькулятора

Для ініціалізації написаного тесту на JavaScript використовуємо команду npx playwright test. Команда забезпечує автоматизовану взаємодію з браузером та

запуск тесту . На рисунку 3.10 наведено результати проходження 6 тестів, для їх виконання знадобилося 3,6 секунди. Що вказує на високу ефективність та швидкодію автоматизованого тестування. Усі 6 тестів пройшли успішно, що говорить про відсутність критичних помилок у додатку, що тестується.

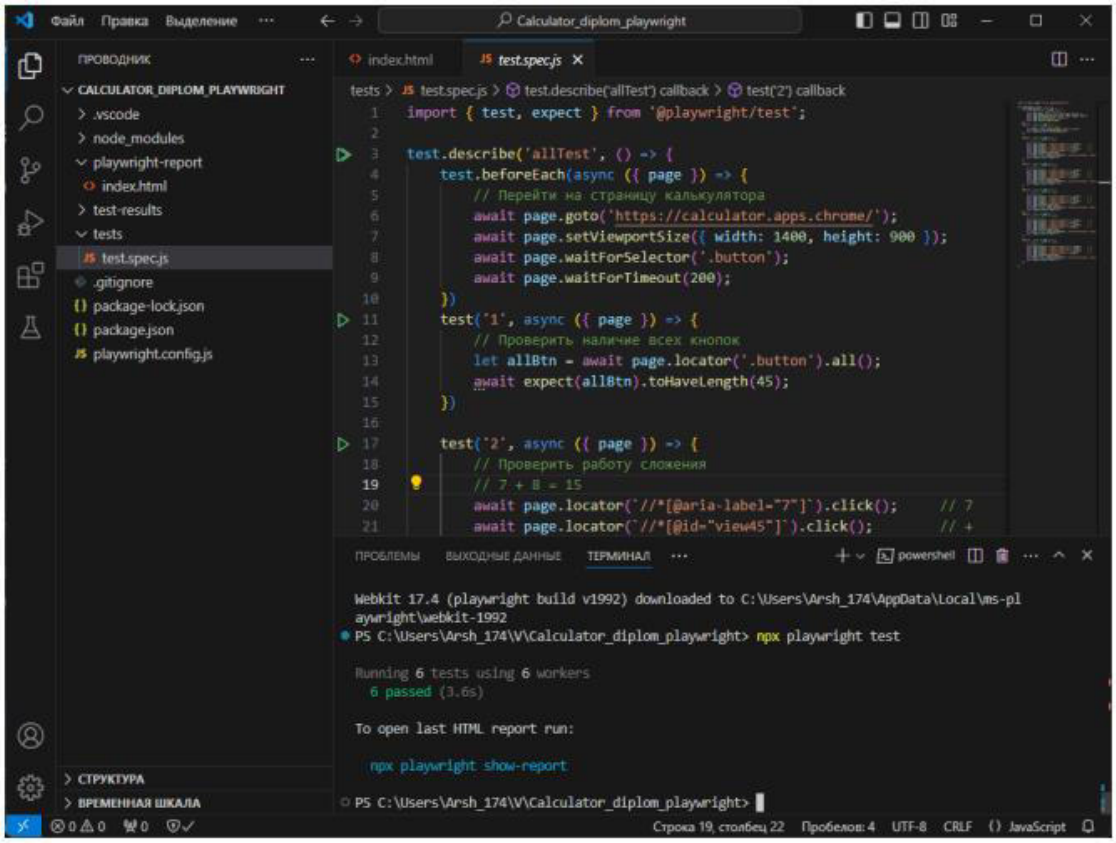

Рисунок 3.10 - Проходження тесту

Також Playwright має можливість запускати тести в режимі користувача за допомогою npx playwright test - ui. Режим користувача дозволяє побачити всі дії, які виконує написаний код, дії виконуються автоматично, але з можливістю спостерігати в реальному часі процес виконання, що показано на рисунку 3.11.

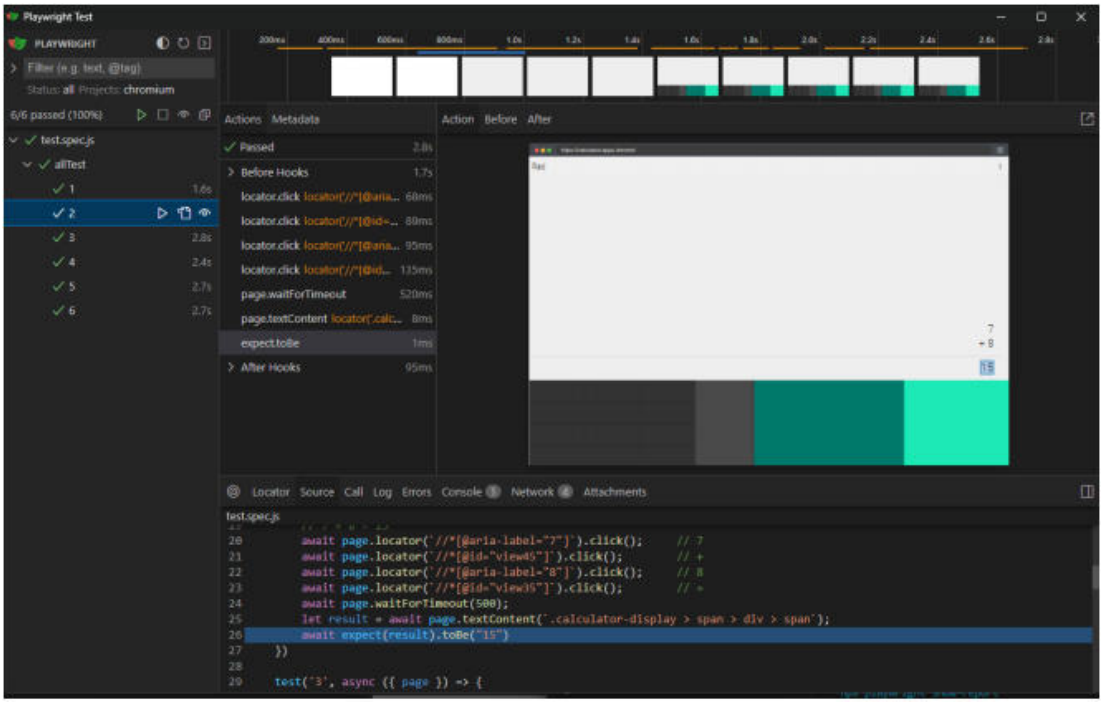

Рисунок 3.11 - Режим користувача Playwright

Це одна з ключових особливостей, яка спрощує розробку та налагодження автоматизованих тестів.

3.4 Висновок до третього розділу

У цьому розділі були описані додаткові технології, які використовуються для проведення автоматизованого тестування (редактор коду, СКВ, серверна платформа). Виконана практична реалізація ініціалізації проєкту та повного циклу прогону всього тесту.

Тестування веб-застосунку калькулятора з використанням Playwright показало високу ефективність і надійність автоматизації тестів. Успішне проходження всіх тест-кейсів свідчить про коректну реалізацію функціональності калькулятора та його готовність до використання.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ

В цьому розділі необхідно проаналізувати важливі питання, котрі стосуються охорони праці та безпеки в надзвичайних ситуаціях.

4.1 Охорона праці

Метою кваліфікаційної роботи магістра є проведення докладного аналізу сучасних інструментів автоматизації тестування ПЗ та впровадження сценарію автотестування [24]. Оскільки, проведення такого виду передбачає застосування комп’ютерної техніки, зокрема ПК та периферійних пристроїв, то обов’язковим є дотримання вимог з охорони праці і техніки безпеки.

Для ефективної і безпечної роботи колективу працівників з розробки автотестів ПЗ, необхідно організувати безпечні умови праці. При цьому керівник організації несе безпосередню відповідальність за порушення нормативно-правових актів з охорони праці [25]. Окрім цього, на робочих місцях працівників необхідно забезпечити дотримання вимог, затверджених Наказом Мінсоцполітики від 14.02.2018 за № 207 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями». Згідно Вимог приміщення, де розміщені робочі місця операторів, крім приміщень, у яких розміщені робочі місця операторів великих ЕОМ загального призначення (сервер), мають бути оснащені системою автоматичної пожежної сигналізації відповідно до цих вимог;

– переліку однотипних за призначенням об'єктів, які підлягають обладнанню автоматичними установками пожежогасіння та пожежної сигналізації, затвердженого наказом Міністерства України з питань надзвичайних ситуацій та у справах захисту населення від наслідків Чорнобильської катастрофи від 22.08.2005 N 161, зареєстрованого в Міністерстві юстиції України 05.09.2005 за N 990/11270 (НАПБ Б.06.004-2005);

– Державних будівельних норм "Інженерне обладнання будинків і споруд.

Пожежна автоматика будинків і споруд", затверджених наказом Держбуду України від 28.10.98 N 247 (далі - ДБН В.2.5-56:2014, з димовими пожежними сповіщувачами та переносними вуглекислотними вогнегасниками.

В інших приміщеннях допускається встановлювати теплові пожежні сповіщувачі. Приміщення, де розміщені робочі місця операторів, мають бути оснащені вогнегасниками, кількість яких визначається згідно з вимогами ДСТУ 4297:2004 «Пожежна техніка. Технічне обслуговування вогнегасників». Загальні технічні вимоги і з урахуванням граничнодопустимих концентрацій вогнегасної рідини відповідно до вимог НАПБ А.01.001-2014. Приміщення, в яких розміщуються робочі місця операторів сервера загального призначення, обладнуються системою автоматичної пожежної сигналізації та засобами пожежогасіння відповідно до вимог ДБН В.2.5-56:2014, НАПБ А.01.001-2014 і вимог нормативно-технічної та експлуатаційної документації виробника. Проходи до засобів пожежогасіння мають бути вільними.

Лінія електромережі для живлення комп’ютера та периферійних пристроїв повинні бути виконаними як окрема групова трипровідна мережа шляхом прокладання фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується для заземлення (занулення) електроприймачів. Не допускається використовувати нульовий робочий провідник як нульовий захисний провідник. Нульовий захисний провідник прокладається від стійки групового розподільного щита, розподільного пункту до розеток електроживлення. Не допускається підключати на щиті до одного контактного затискача нульовий робочий та нульовий захисний провідники.

Площа перерізу нульового робочого та нульового захисного провідника в груповій трипровідній мережі має бути не менше площі перерізу фазового провідника. Усі провідники мають відповідати номінальним параметрам мережі та навантаження, умовам навколишнього середовища, умовам розподілу провідників, температурному режиму та типам апаратури захисту, вимогам документу НПАОП 40.1-1.01-97.

У приміщенні, де одночасно експлуатуються понад п’ять комп’ютерів, на

помітному, доступному місці встановлюється аварійний резервний вимикач, який може повністю вимкнути електричне живлення приміщення, крім освітлення. Комп’ютери повинні підключатися до електромережі тільки за допомогою справних штепсельних з'єднань і електророзеток заводського виготовлення.

У штепсельних з'єднаннях та електророзетках, крім контактів фазового та нульового робочого провідників, мають бути спеціальні контакти для підключення нульового захисного провідника. Їхня конструкція має бути такою, щоб приєднання нульового захисного провідника відбувалося раніше, ніж приєднання фазового та нульового робочого провідників. Порядок роз’єднання при відключенні має бути зворотним. Не допускається підключати комп’ютери до звичайної двопровідної електромережі, в тому числі – з використанням перехідних пристроїв. Електромережі штепсельних з'єднань та електророзеток для живлення комп’ютерної техніки повинні бути виконаними за магістральною схемою, по 3-6 з'єднань або електророзеток в одному колі. Штепсельні з'єднання та електророзетки для напруги 12 В та 42 В за своєю конструкцією мають відрізнятися від штепсельних з'єднань для напруги 127 В та 220 В. Штепсельні з'єднання та електророзетки, розраховані на напругу 12 В та 42 В, мають візуально (за кольором) відрізнятися від кольору штепсельних з'єднань, розрахованих на напругу 127 В та 220 В.

При підвищенні ефективності контролю доступу в приміщення, де для забезпечення безпеки мешканців, співробітників і збереження майна використовуються ДС, важливим, з точки зору охорони праці, є забезпечення достатньої величини природного та штучного освітлення, які визначені у НПАОП 0.00-7.15-18. Організація робочого місця фахівця із дослідження методів та програмно-апаратних засобів оптимізаційних процесів на основі ГА повинна забезпечувати відповідність усіх елементів робочого місця та їх розташування ергономічним вимогам ДСТУ 8604:2015 «Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. Загальні ергономічні вимоги».

Таким чином, у результаті аналізу вимог щодо охорони праці користувачів комп’ютерів, визначено особливості організації робочих місць, вимог з

електробезпеки, природного та штучного освітлення для ефективної і безпечної роботи фахівців із проведення докладного аналізу сучасних інструментів автоматизації тестування ПЗ та впровадження сценарію автотестування.

4.2. Функціонування державної системи спостереження, збирання, оброблення та аналізу інформації про стан довкілля під час надзвичайних ситуацій мирного та воєнного часу

Моніторинг довкілля – це система спостереження, збирання та аналізу інформації про ситуацію, що може скластись під час надзвичайних ситуацій мирного та воєнного часу [26]. Також це система спостереження за визначеними об’єктами, явищами та процесами з метою оперативного оцінювання їх стану, виявлення результатів впливу на них зовнішніх чинників та прийняття відповідних управлінських рішень (ДСТУ 3891:2013) (див. ДСТУ 7295:2013).

Моніторинг потенційно небезпечних об’єктів це спостереження, контролювання за зміною параметрів технологічних режимів з метою збирання, збереження, передавання та аналізування інформації щодо поточного стану потенційно небезпечних об’єктів, наявності та кількості порушень вимог безпеки, відпрацювання рекомендацій щодо проведення 98 робіт із запобігання та ліквідування техногенних надзвичайних ситуацій та їх наслідків (ДСТУ 7295:2013).

Моніторинг джерел надзвичайних ситуацій це система спостереження за об’єктами, які можуть бути джерелами надзвичайних ситуацій, що має на меті виявлення небезпеки, збирання, узагальнення та аналізування оперативної інформації стосовно стану об’єктів моніторингу та розроблення науково-обґрунтованих рекомендацій щодо проведення заходів із запобігання та ліквідування надзвичайних ситуацій (ДСТУ 7295:2013).

Моніторинг довкілля – це систематичні спостереження і контролювання, які проводять регулярно, за єдиною програмою для оцінювання стану довкілля, аналізування процесів, які відбуваються в ньому і своєчасне виявлення тенденцій

його змінювання (ДСТУ 7295:2013).

Моніторинг надзвичайних ситуацій (НС) – система спостереження за об’єктами, які можуть бути джерелами надзвичайних ситуацій, що має на меті виявлення небезпеки, збирання, узагальнення та аналізування оперативної інформації щодо об’єктів моніторингу та розроблення науково обґрунтованих рекомендацій щодо проведення заходів із запобігання та ліквідування НС.

Моніторинг небезпечних явищ та процесів це система спостереження та контролювання за розвитком небезпечних та стихійних природних явищ і процесів, чинниками, які спричинюють їх формування та розвиток, аналізування, збереження та передавання інформації щодо виявлення тенденцій їх змінювання, розроблення комплексу заходів щодо запобігання природним надзвичайним ситуаціям та ліквідування їх наслідків. Небезпечні природні явища і процеси підрозділяють на геофізичні, геологічні, гідрологічні, метеорологічні, медико-біологічні та пожежі в природних екосистемах (ДСТУ 7295:2013).

Моніторинг пожеж в екосистемах це спостереження, контролювання, збирання, аналізування, збереження та передавання інформації щодо 99 пожежної небезпеки в природних екосистемах (умов погоди, стану горючих матеріалів, інших пожежонебезпечних чинників), з метою своєчасного планування та здійснення заходів щодо запобігання виникненню і ліквідування пожеж та їх наслідків (ДСТУ 7295:2013).

Моніторинг радіаційної безпеки це спостереження і контролювання рівня радіоактивного забруднення місцевості, повітря, води, продовольства, об’єктів господарювання, дозових навантажень на населення з метою прийняття оперативних рішень щодо запобігання виникненню та ліквідування надзвичайних ситуацій та їх наслідків (ДСТУ 7295:2013).

Моніторинг хімічної небезпеки це спостереження, контролювання, збирання, аналізування, збереження та передавання інформації щодо визначення ступеня і характеру хімічного забруднення довкілля, санітарногігієнічний нагляд за дотриманням установлених нормативів з метою виявлення джерела надходження небезпечних хімічних речовин, запобігання виникненню та ліквідування

надзвичайних ситуацій та їх наслідків (ДСТУ 7295:2013).

Збір та аналіз інформації про стан довкілля під час мирного та воєнного стану дає можливість приймати оперативні рішення для адекватного реагування на ситуацію [26].

4.3. Висновки до четвертого розділу

В цьому розділі проаналізовано важливі питання охорони праці та безпеки в надзвичайних ситуаціях, висвітлено питання функціонування державної системи спостереження, збирання, оброблення та аналізу інформації про стан довкілля під час надзвичайних ситуацій мирного та воєнного час.

ВИСНОВКИ

На сьогоднішній день веб-застосунки продовжують розвиватися, а кількість використовуваних технологій, як і різноманітність дефектів, тільки збільшуватися. У цій роботі було проведено аналіз технік та методологій їх тестування веб-застосунків.

Крім того, було виявлено, що і ручне, і автоматизоване тестування вигідніше застосовувати на початковому етапі розробки нової програми, для мінімізації виявлення помилок на наступних етапах, і зменшення вартості розробки в цілому.

Додатково було проведено дослідження сутності автоматизованого тестування веб-застосунків. Після цього були розроблені сценарії автоматизованого тестування вже існуючого функціоналу калькулятора.

Ці дії були спрямовані на усунення помилок для забезпечення якості програмного продукту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Manfred Baumgartner, Richard Seid and 4 more. Test Automation Fundamentals: A Study Guide for the Certified Test Automation Engineer Exam. Rocky Nook. 2022. 330 p.

2. Types of Software Testing. URL: https://www.geeksforgeeks.org/software-testing/types-software-testing/ (дата звертання: 20.11.2025).

3. Titus Winters, Tom Manshreck, Hyrum Wright. Software Engineering at Google: Lessons Learned from Programming Over Time, O'Reilly Media, 2020,599 p.

4. 3 рівні автоматизації тестування. URL: https://testmatick.com/uk/3-rivni-avtomatyzacziyi-testuvannya/ (дата звертання: 23.11.2025).

5. Автоматизоване тестування. URL: http://automated-testing. info/ (дата звертання: 23.11.2025).

6. Ручне і автоматизоване тестування. URL: http://qalight.com.ua/baza-znanij/ruchnoe-i-avtomatizirovannoe.html (дата звертання: 24.11.2025).

7. Документація Playwright. URL: https://playwright.dev/ (дата звертання: 21.11.2025).

8. Документація Puppeteer. URL: https://pptr.dev/ (дата звертання: 21.11.2025).

9. Документація Cypress. URL: https://www.cypress.io/ (дата звертання: 21.11.2025).

10. Документація Selenium. URL: http://docs. seleniumhq.org// (дата звертання: 21.11.2025).

11. Virtual Studio Code. URL: https://code.visualstudio.com/ (дата звертання: 21.11.2025).

12. Git. URL: https://git-scm.com/ (дата звертання: 26.11.2025).

13. NodeJS. URL: https://nodejs.org/en/ (дата звертання: 27.11.2025).

14. Дацко А.І. Види тестування програмного забезпечення // Інформаційні моделі, системи та технології: Праці XIII наук.-техн. конф. Тернопіль, 17-18 грудня 2025 р. с. 86.

15. Дацко А.І Типи тестування знання коду // XIV Міжнародна науково-практична конференція молодих учених та студентів «Актуальні задачі сучасних технологій», Тернопіль, 11-12. Грудня 2025 р. с. 250 -251.

16. Svyatoslav Kulikov. Software testing. Base course, 3rd Edition/ EPAM Systems, 2022.-278p.

17. Dorothy Graham, Rex Black, Erik van Veenendaal. Foundations of Software Testing ISTQB Certification: 4th Edition. Cengage Learning. 2024, 288 p.

18. Stefanyshyn, V. , Stefanyshyn, I. , Pastukh, O. , Yatsyshyn, V. , Yakymenko. Accuracy of software and hardware of computer systems for human-machine interaction, I. CEUR Workshop Proceedings, 2024, 3842, pp. 178–183.

19. Stefanyshyn, I. , Pastukh, O., Stefanyshyn, V. , Baran, I. , Boyko, I. Robustness of AI algorithms for neurocomputer interfaces based on software and hardware technologies CEUR Workshop Proceedings, 2024, 3742, pp. 137–149.

20. Petryk, M. , Bachynskyi, M. , Brevus, V. , Mudryk, I. , Mykhalyk, D. Analysis technology of neurological movements considering cognitive feedback influences of cerebral cortex signals CEUR Workshop Proceedings, 2022, 3309, pp. 45–54.

21. Boyko, I. , Petryk, M. , Mudryk, I. , Stoianov, Y., Tsupryk, H. Mathematical Model of the Capacitor Based on Zeolite Material. Proceedings - International Conference on Advanced Computer Information Technologies, ACIT, 2021, pp. 45–48.

22. Boyko, I. , Petryk, M. , Mykhailyshyn, R . Excitons in resonant tunnelling structures based on AlN/GaN/AlN/AlGaN/AlN nitride: spectral dependences and intensities of interband optical transitions. Ukrainian Journal of Physical Optics, 2022, 23(3), pp. 180–191.

23. M. R. Petryk, A. Khimich, M. M. Petryk, and J. Fraissard, “Experimental and computer simulation studies of dehydration on microporous adsorbent of natural gas used as motor fuel,” Fuel, Vol. 239, 1324–1330 (2019). https://doi.org/https://doi.org/10.1016/j.fuel.2018.10.134

24. Методичні вказівки до виконання кваліфікаційної роботи магістра для здобувачів спеціальності 121 – Інженерія програмного забезпечення, всіх форм

навчання / укладачі Михалик Д.М., Цуприк Г.Б., Бревус В.М., Мудрик І.Я. – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. – 44 с.

25. Заікіна Д., Глива В. Основи охорони праці та безпека життєдіяльності. 2019. URL: https://doi.org/10.31435/rsglobal/001 (дата звертання: 10.12.2025).

26. Безпека в надзвичайних ситуаціях. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання / укл.: Стручок В. С. Тернопіль: ФОП Паляниця В. А., 2022. 156 с.

ДОДАТКИ

Додаток А

Апробація результатів

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя
Маріборский університет (Словенія)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Краківський економічний університет (Польща)
Вроцлавський економічний університет (Польща)
Університет «Опольська Політехніка» (Польща)
Національний університет «Полтавська політехніка імені Юрія Кондратюка»
Вінницький національний аграрний університет
Львівський національний університет ім. І. Франка
Головне управління Пенсійного фонду в Тернопільській області
Наукове товариство ім. Шевченка
Тернопільський обласний комунальний інститут післядипломної педагогічної освіти
Сумський державний педагогічний університет
Запорізький національний університет

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник

тез доповідей

XIV Міжнародної науково-технічної конференції молодих учених та студентів

11-12 грудня 2025 року

УКРАЇНА
ТЕРНОПІЛЬ – 2025

УДК 004.415.53
А.І. Дацко, студент гр. СПм-61
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ТИПИ ТЕСТУВАННЯ ЗНАННЯ КОДУ

A.I. Datsko, st.group SPm-61

TYPES OF CODE KNOWLEDGE TESTING

Розглянемо типи тестування знання коду (рис.1) [1]: “чорної скриньки”; “білої скриньки”; "сірої скриньки".

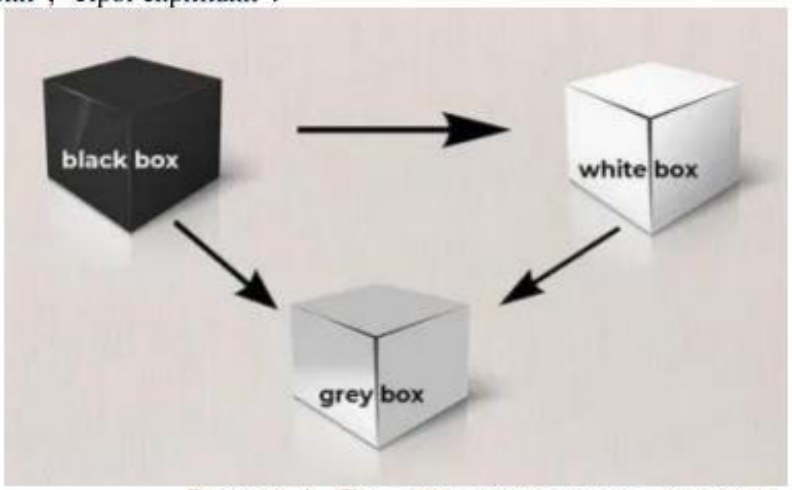

Рисунок 1 - Типи тестування за знанням коду

Основна мета тестування методом “чорної скриньки” полягає у дослідженні невідомих аспектів роботи системи. Це стосується, перш за все, її внутрішньої будови. При цьому тестувальник, всупереч традиційному уявленню про тестування, бере на себе роль типового кінцевого користувача, проводячи тестування, спираючись на технічні специфікації, а також інші супутні документи, що деталізують функціональні аспекти об'єкта, що тестується.

Отже, процес розробки ідей тестування відбувається з урахуванням очікуваних моделей поведінки користувачів. Цей підхід часто визначається як поведінковий, оскільки тестувальник прагне відтворити типові сценарії взаємодії з продуктом виходячи за межі суто технічних аспектів його функціонування [2].

До плюсів цього типу можна віднести:

– об'єктивність. Тестування "чорної скриньки" дозволяє оцінювати систему виключно за її зовнішньою поведінкою, що робить процес об'єктивнішим. Перспектива користувача - тестувальники можуть емулювати дії звичайних користувачів, що допомагає виявити проблеми, з якими вони можуть зіткнутися в процесі використання продукту;

– простота. Таке тестування не вимагає глибокого знання внутрішньої будови системи, що робить його доступним для значного числа спеціалістів.

Варто вказати і недоліки такого тестування [1]:

– обмежене охоплення. Оскільки тестування проводиться виключно на основі зовнішніх характеристик системи, деякі внутрішні дефекти можуть залишитися непоміченими;

– недолік деталізації. Тестування "чорної скриньки" може втратити тонкі нюанси функціонування системи, які можуть бути важливими для її коректної роботи;

– залежність від документації. Ефективність цього методу тестування залежить від якості та повноти технічної документації про продукт. У разі її відсутності чи неповноти виникають складності розробки тест-кейсів;

– обмежений контроль над тестовим процесом. Тестувальники можуть обмежуватися лише тими сценаріями, які передбачені у специфікаціях чи інших документах, та пропускати важливі аспекти функціонування системи, які не були документовані.

"Біла скринька" є повним антиподом до чорної. В процесі тестування чорної скриньки потрібно запуск програми та спостереження за її діями, у той час як при використанні методу білої скриньки цього не потрібно. Достатньо вивчити код програми.

Тестування методом "білої скриньки" (також відоме як прозоре, відкрите, скляне тестування або структурне тестування) є методикою перевірки ПЗ, яка передбачає, що тестувальнику відома внутрішня структура, будова та реалізація системи При використанні цього методу вибираємо вхідні значення, спираючись на знання про код, який їх оброблятиме. Крім того, є уявлення про те, яким має бути результат цієї обробки. Розуміння всіх особливостей тестованої програми та її реалізації є обов'язковим застосування цієї методики. Тестування «білої скриньки» передбачає глибоке занурення у внутрішній пристрій системи, виходячи поза її зовнішніх інтерфейсів [2].

Переваги цього методу:

– аналіз коду дозволяє виявляти приховані за інтерфейсом проблеми з ПЗ;

– тестування починається на ранніх етапах розробки, що дозволяє поліпшити якість продукту;

– має розгалужену систему метрик, зібрати і проаналізувати які можна з легкістю автоматизувати;

– сприяє написанню якісного коду та додаткового проходження етапу код-ревью;

– багато підходів даного методу підтверджують свою ефективність, грунтуючись на суворих технічних принципах.

Недоліки:

– зростають вимоги до навичок та вмінь тестувальників;

– аналізує роботу програми без реального середовища виконання, ігноруючи його вплив;

– аналізує функціонування програми поза контекстом реальних сценаріїв, у яких його використовують користувачі.

"Сіра скринька" зі свого боку поєднує у собі два методу описані раніше. У нашому розпорядженні лише обмежене уявлення про внутрішню будову програми. Є доступ до деяких аспектів внутрішньої структури та алгоритмів роботи ПЗ для створення найбільш ефективних тест-кейсів. Однак сам процес тестування виконується з використанням методу чорної скриньки, тобто з погляду користувача.

Цей підхід до тестування також відомий як метод напівпрозорої скриньки: є доступ до деяких аспектів, але не до всіх.

Література

1. Всі відомі типи тестування (100+ штук). URL: https://dou.ua/forums/topic/40666/ (дата звертання: 12.11.2025).

2. Види тестування та відмінності між ними. URL: https://qagroup.com.ua/publications/vydy-testuvannya-ta-vidminnosti-mizh-nymy/ (дата звертання: 12.11.2025).

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

XIII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

17-18 грудня 2025 року

ТЕРНОПІЛЬ
2025

УДК 004.415.53
А. Дацко
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ВИДИ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

UDC 004.415.53
A. Datsko

TYPES OF SOFTWARE TESTING

У рамках тестування програмного забезпечення (ПЗ) виділяють множину видів та методів тестування. Кожен вид тестування спрямований на перевірку певного функціоналу. На рис. 1 наведено схему основних видів тестування [1].

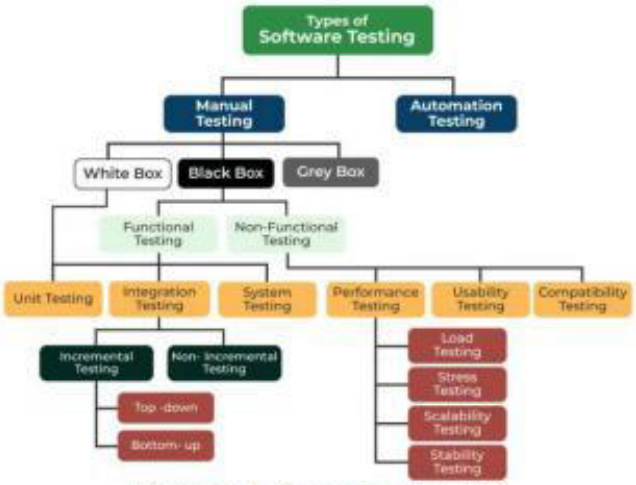

Рисунок 1. Види тестування ПЗ

Функціональне тестування – це метод, у якого тести створюються з урахуванням технічного завдання. Цей підхід до тестування ПЗ не включає докладне вивчення внутрішньої структури програми, характерне для методу «білої скриньки», а сконцентрований на перевірці вхідних даних та аналізі вихідних результатів [1]. Таке тестування описує очікувану поведінку системи, спираючись на аналіз специфікацій функціональності компонента чи всієї системи загалом. Воно не обмежується тестуванням окремих функцій чи методів, модуля чи класу, а скоріше оцінює попередньо певну поведінку програмного елемента або системи у загальному.

Функціональне тестування включає:працездатність програми як функції; коректність роботи програми; взаємодія програми з іншими системами чи компонентами; відповідність прийнятим специфікаціям та правилам; захищеність від загроз та уразливостей; тестування продуктивності.

У сфері розробки ПЗ, тестування продуктивності здійснюється з метою оцінки роботи системи, що тестується, з точки зору її оперативності та стабільності при певних навантаженнях. Цей вид тестування також дозволяє досліджувати, вимірювати та перевіряти інші якісні характеристики системи, такі як її масштабованість, надійність та ефективне використання ресурсів.

Існують два основні підходи до проведення тестування продуктивності ПЗ:

– ґрунтується на робочому навантаженні, при якому ПЗ піддається тестуванню в умовах, що відповідають різним сценаріям використання;

– базується на бета-тестуванні, при якому система тестується реальними кінцевими користувачами в реальних умовах експлуатації.

Розглянемо види тестування продуктивності:

Навантажувальне тестування. Спрямоване на аналіз поведінки компонентів або системи під навантаженням, що збільшується. Під навантаженням розуміється очікувана кількість одночасних користувачів, які звертаються до застосунку, і виконують певну кількість операцій у певний період. Таке тестування спрямовано визначення часу відгуку важливих операцій бізнес-процесів програми. У ході випробувань також контролюються бази даних, сервери застосунків та інше обладнання, які впливають на роботу ПЗ.

Стрес-тестування. Спрямоване на аналіз працездатності та стійкості системи в умовах екстремальних навантажень. Цей вид тестування розглядається як ефективний метод оцінки здатності системи впоратися зі збільшенням навантаження до критичних значень, забезпечуючи при цьому збереження доступності та коректну обробку поставлених завдань за умов, що перевищують типові експлуатаційні параметри. Завдяки стрес-тестуванню відбувається виявлення вразливостей, що забезпечують додатковий захист критично важливих компонентів ПЗ.

Тестування стабільності. Застосовується з метою оцінки стійкості ПЗ перебувати під передбачуваним навантаженням протягом тривалого інтервалу часу. Крім того, для даного виду тестування здійснюється моніторинг використання оперативної пам'яті з метою виявлення деградації продуктивності, що проявляється у зниженні швидкості обробки інформації, збільшенні часу реакції та відгук програми на дані, що вводяться.

Конфігураційне тестування. Постає як типовий спосіб оцінки продуктивності, проте у ньому є значна відмінність із іншими способами. Основна увага приділяється впливу різноманітних конфігурацій випробуваного ПЗ, різних пристроїв та ОС. У цьому контексті важливо не тільки визначити, як застосунок буде працювати в ідеальних умовах, але і як він поводитиметься при різних налаштуваннях його оточення.

Юзабіліті – тестування. Метод перевірки ПЗ, яке використовується для того, щоб зрозуміти чи ергономічно випробуване ПЗ чи ні. Тобто тестування дозволяє відповісти на запитання: чи "зручно" користуватися даним додатком кінцевому користувачеві?

Тестування безпеки. Спеціалізується на забезпеченні безпеки ПЗ. У контексті загальної загрози системам ПЗ від боку зловмисників, таких як хакери, конкуренти та інші, дане тестування покликане проводити перевірку стійкості ПЗ, що тестується, перед впливом шкідливих атак і програм. Цей вид тестування необхідний для детектування вразливостей і забезпечення захисту ПЗ від небажаних впливів.

Тестування локалізації. Випробування локалізації відображає процес тестування ПЗ на відповідність різним мовам, культурним особливостям, а також регіональним та місцевим контекстам. Таке тестування відіграє ключову роль у забезпеченні адаптації ПЗ до різних культурних та локальних контекстів, що зрештою сприяє підвищенню зручності використання та рівня задоволеності користувачів.

Тестування сумісності. Пов'язане з інтеграцією випробуваного ПЗ з іншими системами або окремими застосунками, є процесом перевірки спільної роботи цих систем. Воно спрямоване на виявлення потенційних конфліктів та збоїв, які можуть виникнути при взаємодії між компонентами ПЗ та зовнішніми системами. Цей вид тестування необхідний для забезпечення ефективної роботи всієї системи загалом і запобігання можливих негативних наслідків від неправильної інтеграції компонентів.

Література

1. Types of Software Testing. URL: .https://www.geeksforgeeks.org/software-testing/types-software-testing/ (дата звертання: 28.11.2025).

Додаток Б

Вихідний код авто-тесту

```
import { test, expect } from '@playwright/test';

test.describe('allTest', () => {
test.beforeEach(async ({page}) => {
        // Перейти на сторінку калькулятора
await page.goto(' https://calculator.apps.chrome/' );
await page.setViewportSize({ width: 1400, height: 900 });
await page.waitForSelector('.button');
await page.waitForTimeout(200);
})

test('availability', async ({page}) => {
        // Перевірити наявність усіх кнопок
let allBtn = await page.locator('.button').all();
await expect(allBtn).toHaveLength(45);
})

test('addition', async ({page}) => {
// Перевірити роботу додавання
// 7 + 8 = 15
await page.locator('//*[@aria-label="7"]' ).click(); // 7
await page.locator('//*[@id="view45"]').click(); // +
await page.locator('//*[@aria-label="8"]').click(); // 8
await page.locator('//*[@id="view35"]').click(); // =
await page.waitForTimeout(500);
        let result = await page.textContent('.calculator-display >
span > div > span');
await expect(result).toBe("15")
})

test(' subtraction', async ({ page }) => {
// Перевірити роботу віднімання // 7 - 8 = -1
         awai page.locat :'/ [@aria-       ]').click( //  7
         awai page.locat :'/ [@id="view43".click();   //  -
         awai page.locat :'/ [@aria-       ]').click( //  8
         awai page.locat :'/ [@id="view35".click();   //  =

await page.waitForTimeout(500);
let result = await page.textContent('.calculator-display
> span > div > span');
```

```
await expect(result).toBe("-1");
})

test('multiplication', async ({page}) => {
// Перевірити роботу множення // 7 * 8 = 56
        awai page.locat :'/ [@aria-      ]').click( //  7
        awai page.locat :'/ [@id="view41".click();  //  *
        awai page.locat :'/ [@aria-      ]').click( //  8
        awai page.locat :'/ [@id="view35".click();  //  =

await page.waitForTimeout(500);
let result = await page.textContent('.calculator-display
> span > div > span');
await expect(result).toBe("56");
})

test('division', async ({ page }) => {
// Перевірити работу ділення // 6 / 2 = 3
await page.locator(`//*[@aria-label="6"]`).click(); // 6
await page.locator(`//*[@id="view39"]`).click(); // /
await page.locator(`//*[@aria-label="2"]`).click(); // 2
await page.locator(`//*[@id="view35"]`).click(); // =
await page.waitForTimeout(500);

let result = await page.textContent(`.calculator-display
> span > div > span`);
await expect(result).toBe("3");
})
test('log', async ({ page }) => {
// Перевірити обчислення логарифму // log 100 = 2
await page.locator(`//*[@aria-label="1"]`).click();
await page.locator(`//*[@aria-label="0"]`).click();
await page.locator(`//*[@aria-label="0"]`).click();
await page.locator(`//*[@id="view61"]`).click();
await page.waitForTimeout(500);
let result = await page.textContent(`.calculator-display
> span > div > span`);
await expect(result).toBe("2");
    })

})
```