Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет комп'ютерних систем і автоматики
(повне найменування інституту, назва факультету (відділення))
Кафедра автоматизації та інтелектуальних інформаційних технологій
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до магістерської дипломної роботи
магістр
(освітньо-кваліфікаційний рівень)

на тему: «Розробка фреймворку для автоматизованого тестування розширень в системі Jira»

Виконав: студент 2 курсу, групи 1АКІТ-21м
Напрям підготовки
151- Автоматизація і комп'ютерно – інтегровані технології
(шифр і назва напряму підготовки, спеціальності)

Паламарчук К. А.
(прізвище та ініціали)
Керівник д.т.н., професор
Квєтний Р.Н.
(прізвище та ініціали)

Рецензент д.т.н. професор кафедри КСУ
Дубовой В.М.
(прізвище та ініціали)

Вінниця 202

Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 15 «Автоматизація та приладобудування»
Спеціальність-151 «Автоматизація та комп'ютерно-інтегровані технології»
Освітньо-професійна програма – Інтелектуальні комп'ютерні системи

ЗАТВЕРДЖУЮ
Завідувач кафедри АІІТ
д.т.н., професор
О.В. Бісікало
"____"_____ 2022 р.

ЗАВДАННЯ

на магістерську кваліфікаційну роботу
_____Паламарчук Катерини Арменівни_____

1.**Тема роботи**_ Розробка фреймворку для автоматизованого тестування розширень в системі Jira
керівник роботи Квєтний Роман Наумович д.т.н.,професор ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ВНТУ № від
_

2. **Строк подання студентом роботи** 12 грудня 2022 р.

3. **Вихідні дані до роботи** Розробка фреймворку для автоматизованого тестування розширень для Jira
4. **Зміст пояснювальної записки** (перелік питань, які потрібно дослідити)
1) Аналіз проблемної області та постановка задачі 2) Аналіз існуючих засобів в області автоматизованого тестування 3) Розробка фреймворку для автоматизованого тестування на основі codeceptjs. 4) Тестування розробленого програмного забезпечення. 5) Економічний розділ
_
5. **Перелік графічного матеріалу** (з точним зазначенням обов'язкових плакатів)
1) Архітектура CodeceptJS. 2) Приклад дій у CodeceptJS 3) Перехід на веб сторінку у CodeceptJS 4) Приклад вказання стогих локаторів у CodeceptJS 5) Приклад команд для запуску тестів у CodeceptJS 6). Користувацький інтерфейс розширення
6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Спеціальна частина	Квєтний Р.Н., професор кафедри АІІТ		
Економічний розділ	Козловський В.О , к.е.н., професор кафедри ЕПОВ		

7. **Дата видачі завдання** _____

КАЛЕНДАРНИЙ ПЛАН

Назва етапів дослідження	Строк виконання етапів	Примітка
1. Науково-технічне та техніко-економічне обґрунтування роботи.	26.09.22 р.	
2. Розробка технічного завдання на науково-дослідну роботу	26.09.22 р.	
3. Аналіз існуючих засобів в області автоматизованого тестування	21.10.22 р.	
4. Розробка фреймворку для автоматизованого тестування на основі codeceptjs	11.11.22 р.	
5. Тестування розробленого програмного забезпечення.	02.12.22 р.	
6. Економічний розділ	02.12.22 р.	
7. Оформлення пояснювальної записки	12.12.22 р.	
8. Захист роботи	з 19.12.22 р. по 30.12.22 р.	

Студентка _____ Паламарчук К.А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Квєтний Р.Н.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У даній дипломній роботі розглянуто питання, пов’язані з розробкою фреймворку автоматизованого тестування розширень для Jira.

Запропонований підхід дозволив спроектувати та розробити програмні засоби для оцінки та аналізу якості отриманих результатів автоматизованого тестування, які за рахунок ефективної організації процесу тестування та обраних інструментів та засобів розробки забезпечують зменшення часу затраченого на тестування, гнучкості функціональних можливостей фреймворку та корегування створених тестів.

Також робота містить загальний огляд засобів та інструментів створення автоматизованих тестів, аналіз їх переваг та недоліків та економічне обгрунтування доцільності даної розробки.

ABSTRACT

In this research, questions about automation framework development for Jira extentions were investigated.

Proposed approach allowes build and develop programing solutions for evaluation and quality analysis of automation testing results , that provides decrease of time spent in testing , flexibility of functional capabilities of framework and correction of created tests by effective organization of testing process and choosen tools of develompent.

Althrough research contains brief overview of tools and instruments of automation tests development, analysis of their pros and cons and economical base of relevance of provided program.

ЗМІСТ

ВСТУП

Актуальність даної роботи полягає в тому, що автоматизоване тестування дає можливість використання таких видів тестування, які неможливо виконати вручну. Також автоматизоване тестування значно прискорює процес тестування без втрати якості, не вимагаючи значних трудовитрат. Крім того, автоматизоване тестування є більш точним, що є важливим фактором при створенні якісного програмного продукту.

Автоматизоване тестування програмного забезпечення - це техніка тестування програмного забезпечення, яка використовується для перевірки та порівняння фактичного результату з очікуваним результатом. Автоматизоване тестування повністю спирається на попередньо підготовлений сценарій, який запускається автоматично. Це допомагає тестувальнику визначити, чи працює додаток так, як очікується. Автоматизація тестів використовується для виконання повторюваних завдань та завдань тестування, які важко виконати вручну.

Основна перевага автоматизації - це висока якість тестування на відміну від ручного. «Людський фактор» не впливає на процес тестування при використанні засобів автоматизації.

Тестування відбувається на протязі всього життєвого циклу ПЗ, починаючи від проектування і закінчуючи етапом експлуатації. Ці роботи безпосередньо пов'язані із завданнями управління вимогами та змінами, адже метою тестування є якраз можливість переконатися у відповідності програм заявленим вимогам. Тому забезпечення якості є одним з головних етапів при створенні програмного продукту. В процесі відбувається перевірка готовності продукту до використання, відповідність до стандартів, наявність технічної документації.

Автоматизація процесу тестування є корисним інструментом для оптимізації витрат компанії та зменшення часу, що витрачається на тестування.

Метою даної роботи є розробка фреймворку для автоматизованого тестування розширень для Jira шляхом дослідження методів автоматизації тестування програмного забезпечення і вибору найбільш ефективних інструментів та засобів проектування.

Об'єктом дослідження є автоматизоване тестування розширень для Jira з використанням різних інструментів та фреймворків.

Предметом дослідження даної роботи є інструменти та фреймворки для автоматизованого тестування розширень для Jira.

Для досягнення поставленої мети необхідно розв'язати наступні задачі:

– сформулювати та описати вимоги до програмної системи, яка буде розроблятися;

– обрати програмне забезпечення та мову для реалізації фреймворку для автоматизації;

– визначити технології та підхід до реалізації;

– підключити всі необхідні бібліотеки, залежності та плагіни, які будуть вхідною точкою для початку розробки фреймворку;

– реалізувати Page Object архітектуру для представлення класів, методів та локаторів;

– створити тестові сценарії для тестування продукту;

– виконати запуск тестів та представити результати запуску тестів;

– описати архітектуру розробленого фреймворку з висновками;

– описати переваги використання розробленого фреймворку в ході тестування веб-додатків на прикладі розширень для Jira.

Методи дослідження. В процесі дослідження застосовуються загальні аналітичні методи комп'ютерних наук, методи розробки автоматизованих тестів, методи теорії алгоритмів, методи оптимізації.

Основний іноваційний результат роботи- розробка, оптимізація та застосування сучасних підходів, методів та алгоритмів для ефективної організації автоматизованого тестування розширень для Jira.

Практичним значенням даної роботи є розроблене програмне забезпечення та алгоритм, що дозволить пришвидшити процес тестування розширень для Jira.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Поняття автоматизованого тестування і його значення для розробки програмного забезпечення

Одним з найбільш ефективних засобів контролю якості програмних продуктів є тестування. Тестування являється вагомою частиною розробки програмного забезпечення. Якістю у сфері розробки програмного забезпечення це не тільки зручність користування програмою або її надійність. Якість програмного засобу – це сукупність його властивостей, які є сукупність його придатності та задовольняють усі потреби за його призначенням. Отже, якісним є продукт, який відповідає чітким критеріям якості, що пред'явлені до нього зацікавленими особами, користувачами або замовниками даного продукту. Щоб програмний продукт вважався якісним він повинен відповідати певним очікуванням ii стандартам. За визначенням Гленфорд Майерс, тестування – це процес дослідження програм з метою знаходження в них помилок [1]. У цьому визначенні під помилками, або дефектами програмного забезпечення, розуміють недоліки в розробленого програмного продукту, наслідком яких є їх невідповідність очікуваним результатам виконання програмного продукту і фактично отриманим результатам. Тестування здійснюється шляхом проведення певних дій у тестованому додатку, результати виконання цих дій у подальшому звіряють з еталонними даними. Також необхідно вказати, що існує поняття «принципів тестування».

Основних принципів тестування 7, вони представлені нижче. Принципи тестування розроблялися останні 50 років і є загальним збірником правил для тестування у цілому.

Тестування може виявити наявність дефектів в програмі, але не може довести їх відсутність. Важливо складати тест-кейси, які будуть знаходити максимальну кількість дефектів. Отже, при правильному тестовому покритті, тестування допоможе знизити ймовірність не виявлення дефектів в програмному забезпеченні. Якщо ж дефекти не були виявлені під час тестування, не можна з упевненістю стверджувати, що їх немає [2].

Неможливо провести повне тестування, яке б враховувало усі комбінації вводу призначеного для користувача та станів системи, за винятком дуже примітивних випадків. Замість спроб виявлення всіх недоліків слід провести аналіз ризиків та розстановити пріоритети, що допоможе якомога більш ефективно розподілити зусилля по забезпеченню якості програмного забезпечення.

Тестування варто проводити напочатку життєвого циклу розробки програмного забезпечення. Зусилля тестування необхідно концентрувати на певних цілях.

Різні елементи системи можуть містити в собі різну кількість дефектів - тобто, частота виявлення дефектів в різних модулях програми може різнитися. Зусилля по тестуванню необхідно розподіляти пропорційно щільності дефектів. Як правило, найбільша кількість критичних дефектів розподілена в обмеженій кількості елементів системи. Що є проявом принципу Парето: 80% дефектів містяться в 20% модулів [3].

Виконуючи одні і ті самі тести знову і знову, Ви виявите те, що вони знаходять все меншу кількість нових помилок. Через те, що система еволюціонує, велика кількість раніше виявлених дефектів виправляються та старі тест-кейси більше не працюють. Щоб запобігти виникненню цього парадоксу, необхідно час від часу вносити поправки в набори тестів, що

використовуються, рецензувати і виправляти їх з метою досягнення їх відповідності новому стану системи і допомагали знаходженню якомога більшої кількості дефектів.

Вибір способу, техніки і виду тестування напряму залежить від самого програмного забезпечення. Наприклад, програмне забезпечення для військових потреб потребує більш суворої та ретельної перевірки, аніж, скажімо, комп'ютерна гра. З тих же міркувань, відповідальний сайт вимагає серйозного тестування на продуктивність, щоб перевірити можливість його роботи при високому навантаженні.

Факт того, що тестування не виявило помилок, ще не означає, що програма готова до користування. Пошук і виправлення дефектів будуть не важливими, якщо виявиться, що система незручна у використанні, і не задовольняє потреб користувача

Тестування може проводитися як вручну, спеціалістами з тестування, так і автоматично, з використанням спеціальних програмних засобів. Процес верифікації програмного продукту, в ході якого основні етапи та функції тесту, такі як запуск, ініціалізація, виконання, аналіз і видача результату, проводяться автоматично з допомогою інструментів автоматизованого тестування, і називається автоматизованим тестуванням програмного забезпечення.

У автоматизованого тестування є як свої переваги, так і недоліки. До сильних сторін автоматизованого тестування відносять: – висока швидкість виконання, набагато перевершує можливості людини; – відсутність впливу «людського фактора» (неуважність, втома); – можливість багаторазового виконання тестів і зниження витрат ресурсів через це; – виконання тест-кейсів особливої складності, недоступних людині; – здатність зберігати,

аналізувати в зручній формі великі обсяги даних; – здатність виконувати низько-рівневі дії з додатком, з операційною системою тощо. До недоліків автоматизованого тестування відносяться: – необхідність залучення висококваліфікованого персоналу для автоматизації замість можливості використовувати працю ручних тестувальників; – витрати на засоби автоматизації, на розробку і підтримку тестів; – фінансові витрати та ризики, пов'язані з наявністю великої кількості наявних інструментів автоматизації і складністю вибору; – старіння тестів у випадках змін вимог, переробки інтерфейсів продуктів, що тестуються. Автоматизація тестування не дозволяє зовсім відмовитися від використання ручного тестування при розробці програмного забезпечення, але дозволяє суттєво знизити його частку, допомагає уникнути рутинних дій у процесі тестування, зменшити вартість і суттєво покращити якість програмного забезпечення, що розроблюється. Тестування є однією з найбільш трудомістких фаз розробки програмного забезпечення, що займає багато часу. При цьому вартість виправлення помилки, не тільки матеріальна, а й фізична, як правило, безпосередньо залежить від часу, який проходить з моменту її виникнення до моменту виявлення. Вважається, що ідея автоматизації процесу тестування дозволить вирішити питання з нестачею часу для здійснення якісного тестування, ставала все більш актуальною та важливою по мірі збільшення складності розроблюваних програмних продуктів і зростання вимог до їх якості. Зараз складно уявити собі розробку масштабного програмного проекту без використання автоматизованого тестування в процесі розробки. Етап розвитку тестування характеризується широкою інтеграцією тестування з процесом розробки в цілому, а також використанням автоматизації, великим набором технологій і інструментальних засобів та бібліотек для автоматизації.

1.2 Аналіз предметної області автоматизації тестування та актуалізація рішень

Автоматизація тестування – це застосування інструментів та технологій для тестування програмного забезпечення з метою зменшення зусиль на тестування, надання можливостей швидше та доступніше. Це допомагає створювати більш якісне програмне забезпечення з меншими зусиллями. Це не тільки виконання автоматичних тестів, це написання і використання різноманітних скриптів для налізу результатів, підготовки тестових даних, парсинга документів тощо, тобто автоматизація всіх рутинних і повторюваних завдань для полегшення процесу тестування. Загалом тестування можна розділити на 3 види: ручне, автоматизоване і частково автоматизоване:

– ручне тестування. Даний вид означає, що все тестування проводиться руками, без використання скриптів і написання автотестів. Тобто всі тестові дані складаються і забиваються руками, прогін тестів також здійснюється руками, ну і звичайно відбувається ручний аналіз виконання тестів і ручна фіксація багів;

– автоматизоване тестування. Тут вже присутня максимальна автоматизація процесу тестування. Тобто генеруються автоматично тестові дані, автоматично виконується прогін тестів, і відбувається автоматичний аналіз прогону тестів і фіксація багів;

– частково автоматизоване тестування. В основному використовується в тих випадках, коли або немає необхідності все автоматизувати, або нема можливості, грошей або часу на автоматизацію. В даному виді тестування використовується напівавтоматична генерація тестових даних, тести

виконуються в автоматичному чи напівавтоматичному режимі, відбувається ручний аналіз прогону тестів і ручна фіксація багів.

Як вже було сказано, автоматизація служить для полегшення роботи тестувальника і поліпшення якості тестування, а значить і продукту. Тобто автоматизація покриває весь спектр робіт з тестування програмного продукту.

Відповідно до інформації порталу «Про Тестінг», автоматизоване тестування ПЗ (Software Automation Testing) – це дії, спрямовані на перевірку тестової системи, в ході якого основні функції та кроки тесту виконуються автоматично за допомогою інструментів для автоматизованого тестування. Software Automation Test Engineer – це технічний фахівець (тестувальник або розробник програмного забезпечення), що забезпечує створення, налагодження та підтримку працездатного стану тестових скриптів, тестових наборів та інструментів для автоматизованого тестування [5].

Інструмент для автоматизованого тестування (Automation Test Tool) – це програмне забезпечення, за допомогою якого фахівець з автоматизованого тестування здійснює створення, налагодження, виконання і аналіз результатів прогону тестових скриптів.

Тестовий Скрипт (Test Script) – це набір певних інструкцій, який автоматично виконується в тестовій системі, щоб перевірити, чи працює система належним чином.

Тестовий набір (Test Suite) – це набір тестових сценаріїв або тестових процедур, що виконуються під час конкретного тестового запуску, для перевірки певної частини ПЗ, об'єднаної загальною функціональністю або цілями.

Тести для запуску (Test Run) – це комбінація тестових скриптів або тестових наборів для подальшого спільного запуску (послідовного або паралельного, в залежності від переслідуваних цілей і можливостей інструменту для автоматизованого тестування)».

Для написання і роботи даних артефактів найзручніше застосовувати загальний фреймворк автоматизованого тестування. Основною проблемою при автоматизації є підготовка та контроль перерахованих артефактів, а також налаштування необхідного середовища для запуску і виконання автоматичних тестів. При цьому є проблема залучення в процес підтримки автоматичних тестів якомога більшої кількості фахівців, що займаються тестуванням, пов'язана з можливим недоліком кваліфікації або спеціалізованих знань мов і середовищ розробки. Для написання автоматичних тестів, більшість інструментів автоматизації вимагають від тестувальника розуміння скриптової мови (Java, Java Script, C# тощо). Зазвичай інструменти дозволяють створювати тести за допомогою запису і відтворення, але, як правило, такі скрипти не надто ефективні, не можуть бути перевикористані і важкі в підтримці.

Фреймворк автоматичного тестування – це набір умов, концепцій і практик, спрямований на перевикористання, зменшення витрат на підтримку і підвищення надійності використання тестів. Використання фреймворку вирішує більшість описаних проблем, що передбачає використання інструменту широким колом фахівців, включаючи розробників і фахівців з ручного тестування [6].

1.3 Моделі створення автоматизованих тестів

Виділяють переважно дві моделі створення автоматизованих тестів: Record and play, Page Object. Розглянемо кожну з них детальніше:

– модель Record and play. Ця модель передбачає запис дій тестувальника здійснених у вікні браузера. Записуються положення курсора, кліки мишки і клавіш на клавіатурі. Це дозволяє швидко створити робочий автоматизований тест, але такі тести є вкрай крихкими і підтримка таких тестів практично неможлива або дуже трудомістка. При автоматизації тестування розроблюваного проекту такі тести принесуть більше збитків ніж прибутку, адже скрипти записані по моделі Record and play не дозволяють відхилень [7];

– модель Page Object. У свою чергу модель Page Object передбачає кодування сценарію тесту вручну на одній з мов програмування. На початку відбувається опис об'єктів сторінки, а потім дій, які потрібно зробити з цими об'єктами [8]. Це дозволяє створювати гнучкі і стійкі до мінорних змін тести, наприклад якщо на веб-сторінці відбувся зсув поля реєстрації, але ключові атрибути полів і кнопок залишилися незмінними, то автотест створений за моделлю record and play доведеться перезаписувати буквально з нуля, а тест створений за моделлю Page Object продовжить нормально функціонувати. Підбиваючи підсумки, в першу чергу, варто зауважити, що ручне і автоматизоване тестування – це взаємодоповнюючі технології. Тому, як у сучасному світі складно випустити якісний продукт без автоматизації, також неможливо обійтися і без ручного тестування. По-друге, хотілося б відзначити, що автоматизовані тести бувають різні і слід вибирати такі тести, які найбільше підійдуть конкретно до даного проекту. Таким чином:

– автоматизація бере вагому участь у проектних процесах і застосовується на всіх рівнях тестування;

– автоматизоване та ручне тестування є взаємодоповнюючими технологіями;

– процес автоматизації все одно потребує людської участі;

– автоматизація потребує певні фінансові додаткові інвестиції, але, в ту ж чергу, дозволяє покращити якість продукту, який розробляється;

– автоматизоване тестування гарантує детерміновану перевірку функціональності; – автоматизація є розробкою (програмуванням);

– підтримка автотестів потребує додаткових витрат в ході активної зміни системи, що тестується.

1.4 Постановка задачі

Проведений аналіз стану проблема дозволяє сформулювати мету досліджень, визначити напрямок і завдання магістерської роботи.

Метою даної роботи є розробка фреймворку для автоматизованого тестування розширень для Jira шляхом дослідження методів автоматизації тестування програмного забезпечення і вибору найбільш ефективних інструментів та засобів проектування.

Фреймворк повинен бути спроектований таким чином, щоб у майбутньому інженери з автоматизації мали можливість легко підтримувати, перевикористовувати та розширювати тестові набори та дані, якщо в ході тестування продукту будуть раптово змінюватися вимоги або функціонал, і потрібно буде швидко змінювати тестові сценарії і програмний код.

Таким чином, необхідно виконати наступні задачі:

– сформулювати та описати вимоги до програмної системи, яка буде розроблятися;

– обрати програмне забезпечення та мову для реалізації фреймворку для автоматизації;

– визначити технології та підхід до реалізації;

– підключити всі необхідні бібліотеки, залежності та плагіни, які будуть вхідною точкою для початку розробки фреймворку;

– реалізувати Page Object архітектуру для представлення класів, методів та локаторів;

– створити тестові сценарії для тестування продукту;

– виконати запуск тестів та представити результати запуску тестів;

– описати архітектуру розробленого фреймворку з висновками;

– описати переваги використання розробленого фреймворку в ході тестування веб-додатків на прикладі розширень для Jira.

2 АНАЛІЗ ІСНУЮЧИХ ЗАСОБІВ В ОБЛАСТІ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ

2.1 Формування вимог до розроблюваної системи

Ціллю розробки є введення автоматизації процесу тестування розширень для Jira на основі створення відповідного фреймворку та спрощення процесу отримання вихідних даних у вигляді виконаного звіту, обґрунтування доцільності використання для автоматизації тестування комерційних засобів розробки, виявлення сильних і слабких сторін такого рішення.

Для виконання поданих вимог необхідно вирішити такі задачі:

- проаналізувати існуючі засоби автоматизації тестування програмного забезпечення;
- вибрати інструмент для автоматизації тестування;
- створити гнучкий фреймворк для автоматизації тестування із подальшою можливістю корегування, розширення та удосконалення проекту;
- створити набір тестових шаблонів та розробити параметричні вхідні дані для тестів;
- реалізувати автоматичний запуск тестів на емуляторах і реальних пристроях;
- налаготити звітування спеціаліста по кожному з проходжень тестів.

2.2 Визначення критеріїв аналізу

В якості критеріїв для аналізу ринку існуючого ПЗ для автоматизації тестування будуть використовуватися:

- поширеність інструменту – мається на увазі наявність бази користувачів, за рахунок яких збільшується ймовірність знаходження

відповідей на ці запитання з використання інструменту, а також ймовірність актуальності знань для розробників. Важливість цього критерію в тому, щоб розробники автотестів мали можливість скористатися накопиченим досвідом інших розробників, а не вирішували всі проблеми «з нуля»;

- підтримка – чи підтримується фреймворк, тобто як часто виходять оновлення/виправлення помилок. Підтримка важлива для можливості виправлення виявлених помилок, що впливають на стабільність роботи системи;
- ціна інструмента – варіантами будуть платне ПЗ, безкоштовне, умовно безкоштовне. Ціна інструменту є економічним критерієм, визначаючим розрахунок прямої ефективності від впровадження програмного засобу для автоматизації;
- відкритість вихідного коду – відкритий вихідний код, або пропрієтарний формат програмного засобу. Цей критерій дозволяє оцінити можливість доопрацювання інструменту для власних потреб;
- підтримка браузерів – так як розглядається фреймворк саме для тестування веб-додатків, важливу роль відіграє підтримка нових версій популярних браузерів;
- підтримка операційних систем. Підтримка якомога більшої кількості операційних систем впливає на вибір того чи іншого інструменту, в тому випадку, якщо тестоване ПЗ передбачає мультиплатформеність;
- мова програмування, на якому написаний фреймворк впливає для бізнесу на простоту/складність пошуку фахівців для розробки і підтримки тестів і самого фреймворку;
- можливості взаємодії зі сторонніми бібліотеками в рамках Continuous Integration / Continuous Delivery. Від цього критерію залежить, можливо

вбудувати тести в процес безперервної розробки і постачання ПЗ, що є важливим критерієм для оцінки застосовності інструменту;

- можливості роботи з моделлю BDD (Behaviour Driven Development). Цей критерій відповідає за рішення необхідності наявності спеціалізованих знань з розробки ПЗ у користувачів фреймворку;
- простота встановлення, налаштування і зручність користування. Аналогічно попереднім пунктам наведеним вище, простота використання впливатиме на розширення списку користувачів фреймворку.

2.3 Порівняльна характеристика існуючих розробок

В даному підрозділі представлена порівняльна характеристика існуючих фреймворків для автоматизації тестування, які є поширеними рішеннями в даній області за описаними в попередньому пункті критеріями.

2.3.1 Selenium

На сьогодні, Selenium інструмент є одним з найбільш популярних фреймворків з відкритим вихідним кодом, призначеним для використання в автоматизації тестування [9]. Розроблений десь в далеких двохтисячних і розвивається протягом наступного десятиліття, за цей час він встиг завоювати серця багатьох тестувальників, особливо тих, у кого в розпорядженні є просунуті навички програмування і досвід написання скриптів. Selenium можна розглядати в якості родоначальника деяких сучасних інструментів автоматизації тестування, наприклад: Katalon Studio, Watir, Protractor і Robot Framework.

Операційних систем, в яких можна працювати з Selenium, є декілька, це Linux, Windows та OS X. Інструмент взаємодіє з багатьма браузерами – Headless браузери, Firefox, Chrome та IE. Вибір мови програмування для

написання скриптів також є широким, їх можна реалізувати на C#, Java, Python, Groovy, Ruby, PHP і Perl.

Однак варто зазначити, що даний фреймворк має як плюси, так і мінуси. До переваг можна віднести гнучкість, а також можливість написання складних і ефективних скриптів для тестування розробляються. З іншого боку, для того щоб почати працювати з Selenium, тестувальник повинен володіти неабиякими знаннями в програмуванні і бути готовим приділяти деяку кількість свого часу і енергії для написання спеціальних фреймів і бібліотек, які забезпечують виконання певних функцій в процесі тестування, ліцензія безкоштовна.

2.3.2 Katalon Studio

Це всеосяжний набір інструментів для тестування веб, мобільних додатків, API та десктопу, який допомагає подолати загальні проблеми автоматизації тестування веб-інтерфейсу, наприклад, спливаючі вікна, iFrame та час очікування [10]. Робота з даним інструментом не залежить від попередніх навичок тестувальника, можна починати роботу навіть з базовими знаннями в сфері тестування. Адже спеціалісти навіть без технічної підготовки та досвіду програмування здатні самостійно реалізувати проект з автоматизації, а для розробників та тестувальників з більшим досвідом Katalon Studio значно зекономить час для написання нових та підтримки існуючих скриптів. Він може легко інтегрувати автоматизовані тести у конвеєр CI / CD, чудово працює в зв'язці з такими відомими інструментами під час тестування ПЗ: JIRA, Jenkins і Git. Бонусом інструменту є функція Katalon Analytics, яка надає повноцінну інформацію про стан та процес тестування за допомогою звітів, які можуть бути у вигляді графічних схем, діаграм, метрик та статистичних збірок. Ліцензія безкоштовна.

2.3.3 Unified Functional Testing (UFT)

Означає уніфіковане функціональне тестування. Раніше він був відомий як QTP (QuickTest Professional) [11]. Один із провідних інструментів автоматизованого функціонального тестування, є продуктом компанії HP у своїй лінійці. Для розробки автоматизованих тестів UFT використовує мову VBScript. Це єдина мова, яка повністю підтримується IDE UFT One. Браузери, які підтримуються: Internet Explorer, Edge, Firefox, Google Chrome. Підтримується операційна системаWindows різних версій та CI з Jenkins. UFT One в основному використовується для функціонального, регресійного та сервісного тестування. Використовуючи UFT One, можна автоматизувати дії користувачів у веб- або клієнтському комп'ютерному додатку, а також тестувати та виявляти помилки в одних і тих же діях для різних користувачів, різному наборі даних, в різних операційних системах Windows та / або різних браузерах. Автоматизація за допомогою UFT One, якщо планується та виконується належним чином, може заощадити значний час та гроші порівняно з ручним тестуванням. Він відомий своєю простотою використання та підтримкою з боку постачальника та великої спільноти тестувальників автоматизації. З цієї причини кваліфіковані фахівці UFT One завжди користуються попитом, ліцензія платна.

2.3.4 Watir

Це група бібліотек Ruby для автоматизування веб-браузерів [12]. Це дозволяє писати тести, які легко читати та підтримувати. Іншими словами, це простий та гнучкий інструмент. Він являється безкоштовним інструментом з відкритим кодом, який не вимагає витрат на використання цього

інструменту. Використовує Ruby, повнофункціональну сучасну мову сценаріїв, а не власний сценарій постачальника. Watir підтримує будь-які додатки – не важливо, на якій технології він розроблений. Він підтримує лише Internet Explorer у Windows. Watir-WebDriver підтримує різні браузери, такі як Chrome, Firefox, Internet Explorer та браузер Opera. Він також працює в безголовому режимі (HTMLUnit). Ruby дає можливість підключатися до баз даних, читати файли даних та електронні таблиці, ліцензія безкоштовна.

2.3.5 Test Complete

Це потужний автоматизований інструмент тестування графічного інтерфейсу для мобільних, веб- та десктопних додатків [13]. Дає можливість створювати точні та повторювані автоматизовані тести на багатьох пристроях, платформах та середовищах. Інструмент дозволяє командам писати сценарії різними мовами, включаючи JavaScript, VBScript та Python, або створювати складні тести за допомогою можливостей запису та відтворення. Він постачається із готовою інтеграцією з інструментами CI/CD, такими як Jenkins та Jira, популярними фреймворками з відкритим кодом та інструментами, такими як Selenium та SoapUI, а також власною підтримкою BDD, ліцензія платна.

2.3.6 Robot Framework

Представляє собою фреймворк для автоматичного тестування з відкритим вихідним кодом, в якому реалізується підхід тестування на основі ключових слів для приймального тестування і розробки через приймальне тестування (ATDD) [14]. Robot Framework надає можливість вирішення різних завдань автоматизації тестування. Однак його можливості можуть бути розширені завдяки впровадженню додаткових бібліотек за допомогою Python і Java. Наприклад, Selenium WebDriver – популярна зовнішня

бібліотека, яка використовується в Robot Framework. Тестувальники можуть використовувати Robot Framework як фреймворк для автоматичного тестування не тільки веб-додатків, але для додатків під Android та iOS. Він виявиться простим у вивченні для тих тестувальників, хто вже знайомий з тестуванням на основі ключових слів, ліцензія безкоштовна.

2.3.7 CodeceptJS

CodeceptJS – це потужний end-to-end фреймворк з відкритим вихідним кодом [15]. Він включає в себе такі переваги:

- орієнтований на сценарії – дозволяє писати прийомочні тести з перспективи кінцеового користувача. Робить тести читабельними та простими для виконання;
- може використовувати більшість доступних драйверів – Playwright, WebDriver, Puppeteer, Protractor, Appium – при цьому не змінюючи код тестів;
- дозволяє запускати тести в декілька потоків;
- дозволяє створювати декілька сесій у межах одного тесту.

Codeceptjs може автоматизувати всі веб-сайти і не залежить від технології, в якій розроблено програму.

Отже, проаналізувавши існуючі аналоги інструментів, які можна було б узяти за основу створення цілісного фреймворку для автоматизації розширень для Jira, можна сказати, що фаворитом є CodeceptJS, так як основними перевагами і критеріями, в порівнянні з наявними на ринку продуктами, у нього будуть:

- прозорість: забезпечує високу швидкість та гнучкість серед міжфункціональних команд процесу життєвого циклу розробки програмного

забезпечення (для розробників, відділу забезпечення якості, клієнтів та керівництва);

- незалежність від платформи: скорочує час тестувальника програмного забезпечення для написання тестових сценаріїв для кожної платформи, що перевіряється. Оскільки це відповідає принципу написання одного тестового сценарію і працює на будь-якій платформі;
- сприяє постійним зусиллям щодо інтеграції: безперервна інтеграція та безперервна доставка є сучасними гнучкими механізмами процесу розробки ПЗ.

2.4 Висновки до розділу

В даному розділі сформовано вимоги до розроблюваної системи, визначено критерії аналізу існуючих застосунків в області автоматизованого тестування, проаналізовано сучасні засоби створення автоматизованих тестів та, на основі визначених критеріїв, обрано CodeceptJS як базу для створення фреймворку автоматизованого тестування розширень для Jira.

3 РОЗРОБКА ФРЕЙМВОРКУ ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ НА ОСНОВІ CODECEPTJS

3.1 Складові частини фреймворку CodeceptJS

CodeceptJS – це потужний end-to-end фреймворк з відкритим вихідним кодом. Він включає в себе такі переваги:

- Орієнтований на сценарії – дозволяє писати прийомочні тести з перспективи кінцеового користувача. Робить тести читабельними та простими для виконання;
- може використовувати більшість доступних драйверів – Playwright, WebDriver, Puppeteer, Protractor, Appium – при цьому не змінюючи код тестів;
- дозволяє запускати тести в декілька потоків;
- дозволяє створювати декілька сесій у межах одного тесту.

Codeceptjs може автоматизувати всі веб-сайти і не залежить від технології, в якій розроблено програму.

У подальших підрозділах детально проаналізовано складові фреймворку.

3.1.1 Архітектура CodeceptJS

CodeceptJS передає виконання усіх команд хелперам – спеціальним функціям, які можна викликати у будь якому місці проекту. Зважаючи на те, які хелпери використовуються, тести будуть виконуватись по-різному. Якщо потрібна кросс-браузерна підтримка – слід обрати WebDriver або TestCafe, які базуються на Selenium. Якщо в пріорітеті швидкість виконання тестів – доцільно обрати Puppeteer на основі Chromium. [15] Діаграма архітектури CodeceptJS зображена на рисунку 3.1.

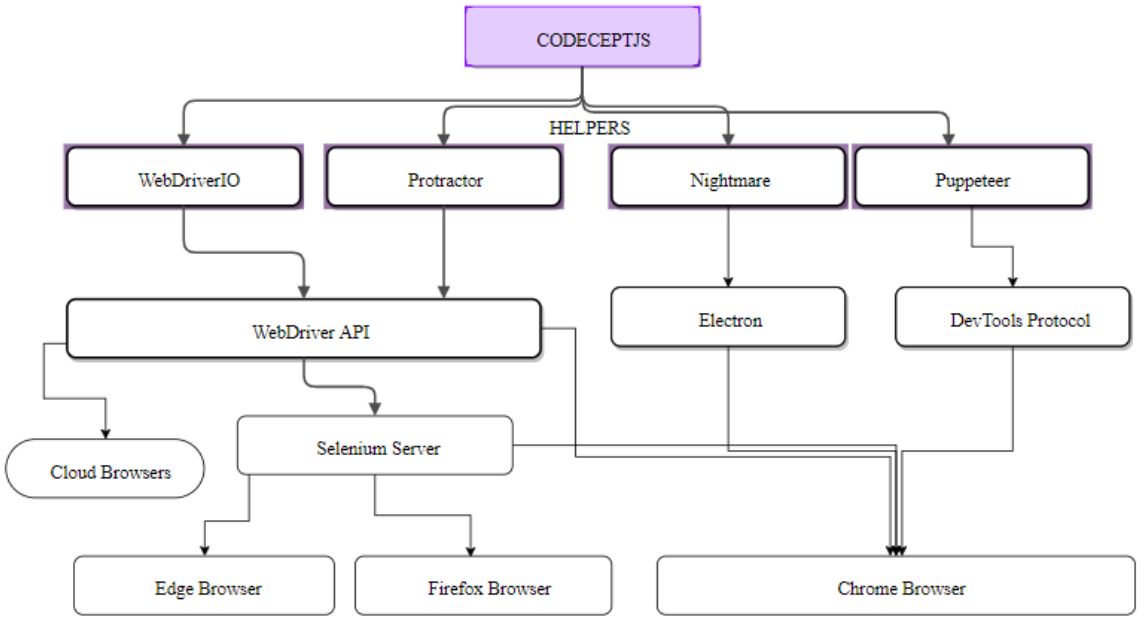

Рисунок 3.1 – архітектура CodeceptJS

Всі хелпери мають однаковий API, що дозволяє з легкістю мігрувати тести з одного бекенду на інший. Проте, через різниці у бекендах та їх обмеження, не гарантовано що тести на різних хелперах будуть сумісними один з одним. Наприклад, не можна встановити заголовки запитів у WebDriver чи Protractor, але це можна зробити у Puppeteer та Nightmare. [16]

3.1.2 Сценарії CodeceptJS

Написання тестових сценаріїв здійснюється з перспективи користувача. Існує актор (представлений як I), який містить дії, описані у хелперах.

```
I.amOnPage('/');
I.click('Login');
I.see('Please Login', 'h1');
// ...
```

Рисунок 3.2 – Приклад дій у CodeceptJS

Зазвичай тест починається з відкриття веб сторінки у браузері. Слід використати функцію «`I.amOnPage()`» яка доступна в усіх хелперах.

```
// When "http://site.com" is url in config
I.amOnPage('/'); // -> opens http://site.com/
I.amOnPage('/about'); // -> opens http://site.com/about
I.amOnPage('https://google.com'); // -> https://google.com
```

Рисунок 3.3 – Перехід на веб сторінку у CodeceptJS

Слід зазначити, що якщо URL не починається з протоколу (http:// чи https://), він вважається відносним і буде доданим до URL який було встановлено у файлі конфігурації.

Пошук елементів на сторінці відбувається за допомогою CSS чи Xpath локаторів. По замовчуванню Codeceptjs намагається вгадати тип локатору. Щоб вказати тип локатора необхідно передати об’єкт який називається строгий локатор.

```
I.seeElement({css: 'div.user'});
I.seeElement({xpath: '//div[@class=user]'});
```

Рисунок 3.4 – Приклад вказання стогих локаторів у CodeceptJS

3.1.3 Запуск тестів у CodeceptJS

Для запуску тестів використовується просто команда ‘run’, а для виконання тестів в декількох браузерах чи декількох потоках використовується команда ‘run-multiple’.

Щоб переглянути покрокові результати запущених тестів потрібно додати прапорець ‘--steps’. Для перегляду більш детальних результатів

використовується прапорець ‘--debug’. Щоб отримати дуже детальні результати використовується прапорець ‘--verbose’.

```
npx codeceptjs run --steps

npx codeceptjs run --debug

npx codeceptjs run --verbose
```

Рисунок 2.5 – Приклад команд для запуску тестів у CodeceptJS

Конфігурація встановлюється в файлі codecept.conf.js який створюється автоматично в процесі ініціалізації. В середині файлу конфігурації можна вмикати і конфігурувати хелпери та плагіни. [17] В проекті можна використовувати декілька файлів конфігурації. В такому випадку необхідно вказати файл за допомогою прапорця ‘-c’ під час запуску. Наприклад можна вказати розмір вікна чи ввімкнути ‘headless’ режим (коли інтерфейс браузера не відображається).

```
const { setHeadlessWhen, setWindowSize } = require('@codeceptjs/configure');

// run headless when CI environment variable set
setHeadlessWhen(process.env.CI);
// set window size for any helper: Puppeteer, WebDriver, TestCafe
setWindowSize(1600, 1200);

exports.config = {
  // ...
}
```

Рисунок 2.6 – Приклад встановлення конфігурації у CodeceptJS

CodeceptJS також дозволяє виконувати декілька сесій в середині одного тесту. Це може бути корисно при тестуванні комунікації між користувачами чату чи інших систем. Для відкриття ще одного браузеру потрібно

використовувати функцію session(). Дана функція очікує першим параметром назву сесії. До сесії можна повернутись використавши вказане ім’я. Також можна перевантажити конфігурацію передавши нові налаштування як другий параметр. Функції передані в функцію сесії можуть використовувати об’єкт ‘I’, об’єкти сторінок та інші об’єкти оголошені в середині сценарію. Ці функції також можуть бути оголошені як асинхронні. [18]

```
Scenario('test app', ({ I }) => {
  I.amOnPage('/chat');
  I.fillField('name', 'davert');
  I.click('Sign In');
  I.see('Hello, davert');
  session('john', () => {
    // another session started
    I.amOnPage('/chat');
    I.fillField('name', 'john');
    I.click('Sign In');
    I.see('Hello, john');
  });
  // switching back to default session
  I.fillField('message', 'Hi, john');
  // there is a message from current user
  I.see('me: Hi, john', '.messages');
  session('john', () => {
    // let's check if john received it
    I.see('davert: Hi, john', '.messages');
  });
});
```

Рисунок 3.7 – Приклад використання сесій у CodeceptJS

3.1.4 CodeceptUI

CodeceptJS має інтерактивний, графічний інтерфейс для запуску тестів, що називається CodeceptUI. Він працює у браузері та допомагає організувати ваші тести.

CodeceptUI може бути використаний для:

- запуску тестів за групами або поодинці
- отримати звіти тестування
- перевірки та огляду тестів
- редагування тестів та об'єктів сторінок
- написання нових тестів
- повторне використання одного сеансу браузера протягом декількох тестових запусків
- легкого входу та виходу з безголового режиму

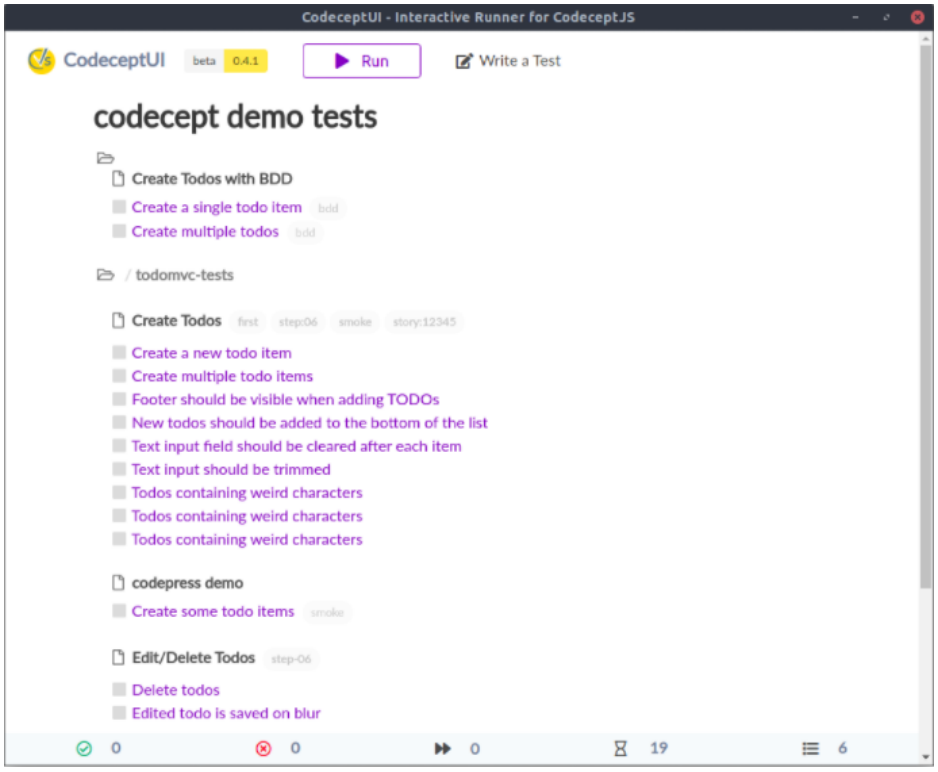

Рисунок 3.8 – Графічний інтерфейс CodeceptUI

Інтерфейс встановлюється автоматично але його можна встановити за допомогою команди ‘npm i @codeceptjs/ui --save’. Щоб запустити CodeceptUI необхідно мати проект з декількома реалізованими тестами та виконати команду ‘npm run codeceptjs:ui’.

3.1.4 Найкращі практики використання CodeceptJS

Доцільно використовувати семантичні елементи сторінки під час написання тестів. Замість CSS/Xpath локаторів краще використати видимі ключові слова на сторінці. Коли важко знайти необхідний текст в середині конкретного елементу рекомендується використати білдер локаторів. [19] Він дозволяє створювати складні локатори через чистй API. Наприклад, якщо необхідно натиснути на елемент, що не є кнопкою чи посиланням і використати його текст, можна використати ‘locate()’ щоб створити читабельний локатор.

```
// clicks element <span class="button">Click me</span>
I.click(locate('.button').withText('Click me'));
```

Рисунок 3.9 – Приклад використання білдера локаторів у CodeceptJS

Щоб писати простіші та ефективніші тести доцільно використовувати скорочення. Вони дозволяють писати тести орієнтовані на одну особливість та та спростити все що не відноситься на пряму до тесту.

Ключові моменти використання скорочень:

- якщо данні необхідні для тесту, слід використовувати api для їх створення;
- якщо необхідно ввійти в систему, слід використати плагін ‘autologin’ замість написання кроків входу в тілі тесту;
- розбиття довгих тестів на декілька коротких, так як довгі тести легко зламати та важко оновлювати;

- повторне використання кроків які часто використовуються;

```
// inside config/components.js
module.exports = {
    DatePicker: "./components/datePicker",
    Dropdown: "./components/dropdown",
}


  include: {
      I: './steps_file',
      ...require('./config/pages'), // require POs and DOs for module
      ...require('./config/components'), // require all components
  },

// inside codecept conf file
bootstrap: () => {
  codeceptjs.container.append({
    testUser: {
      email: 'test@test.com',
      password: '123456'
    }
  });
}
```

Рисунок 3.10 – Приклад використання описаних практик для файлу конфігурацій.

Також доцільно використовувати декілька файлів конфігурацій в проекті. Разом з ними варто використовувати ‘.env’ файл щоб завантажувати чутливі дані, такі як паролі адміністаторів, ключі доступу до баз даних та хмарних сервісів.

3.2 Вибір мови програмування для розробки програмного забезпечення з використанням CodeceptJS.

JavaScript - це мова програмування, яка дає можливість реалізовувати складну поведінку веб-сторінки.Програми на цій мові називаються скриптами. Вони можуть вбудовуватися в HTML і виконуватися автоматично при завантаженні веб-сторінки. Скрипти поширюються і виконуються, як простий текст. Їм не потрібна спеціальна підготовка або компіляція для запуску. Це відрізняє JavaScript від іншої мови - Java. [20] Сьогодні JavaScript може виконуватися не тільки в браузері, а й на сервері або на будь-якому іншому пристрої, який має спеціальну програму, що називається «движком» JavaScript. У браузера є власний движок, який іноді називають «віртуальна машина JavaScript». [21]

JavaScript - це «безпечна» мова програмування. Вона не надає низькорівневий доступ до пам'яті або процесору, тому що спочатку мова була створена для браузерів, які не потребують цього. Можливості JavaScript сильно залежать від оточення, в якому він працює. Наприклад, Node.JS підтримує функції читання / запису довільних файлів, виконання мережевих запитів і т.д. У браузері для JavaScript є доступ до всього, що пов'язано з маніпулюванням веб-сторінками, взаємодією з користувачем і веб-сервером.

Наприклад, в браузері JavaScript може:

– додавати новий HTML-код на сторінку, змінювати існуючий вміст, модифікувати стилі;

– реагувати на дії користувача, клацання миші, перемістити вказівник, натискання клавіш;

– відправляти мережеві запити на віддалені сервера, завантажувати і завантажувати файли (технології AJAX і COMET);

– отримувати і встановлювати куки, задавати питання відвідувачеві, показувати повідомлення;

– запам'ятовувати дані на стороні клієнта («local storage»).

JavaScript - це єдина браузерна технологія, що поєднує в собі всі три речі: повна інтеграція з HTML / CSS, «прості речі робляться просто», підтримується всіма основними браузерами і включений за замовчуванням. [16]

JavaScript сам по собі досить компактний, але дуже гнучкий. Розробниками написано велику кількість інструментів поверх основної мови JavaScript, які дають змогу використовувати величезну кількість додаткових функцій . До них відносяться:

– програмні інтерфейси додатка (API), вбудовані в браузери, що забезпечують різні функціональні можливості, такі як динамічне створення HTML і установку CSS стилів, захоплення і маніпуляція відеопотоком, робота з веб-камерою користувача або генерація 3D графіки і аудіо семплів;

– сторонні API дозволяють впроваджувати функціональність в сайти від інших розробників, таких як Twitter або Facebook; [22]

– також буде можливість застосувати до HTML сторонні фреймворки і бібліотеки, що дозволить прискорити створення сайтів і додатків.

3.3 Опис загальної роботи програмного забезпечення

Дану розробку планується застосовувати відносно розширень для системи управління проектами Jira. Система містить багато модулів та розширень, потребує виконання великої кількості тестів для перевірки правильності роботи кожного з них. Розроблений фреймворк буде містити автоматизовані тести для перевірки графічного інтерфейсу та функціоналу розширень.

Зовнішній вигляд розширення, що буде протестовано, наведено на рисунку 3.1.

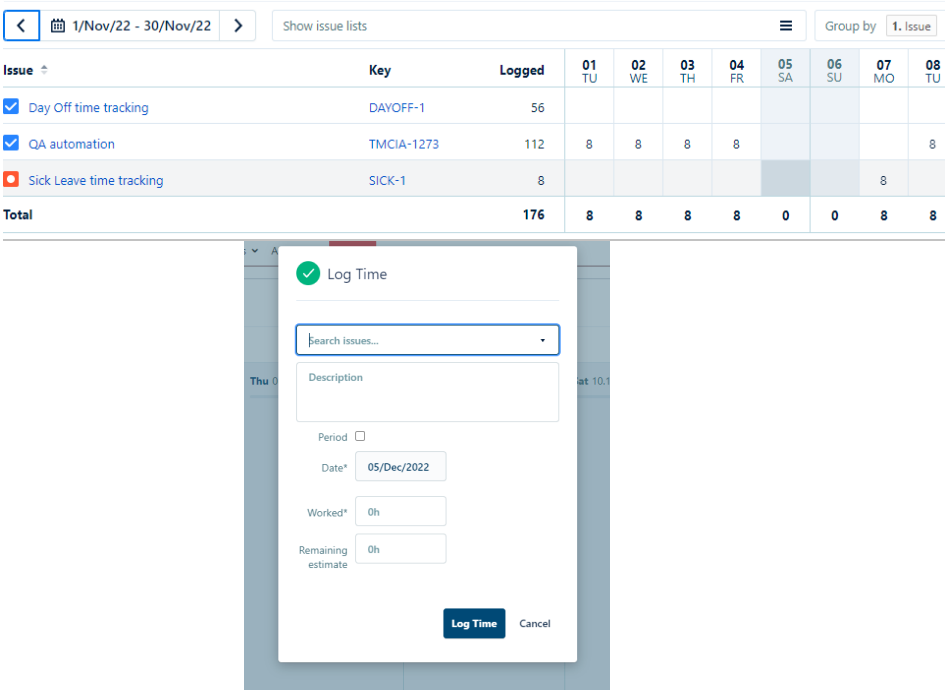

Рисунок 3.11 – Користувацький інтерфейс розширення

Розширення, для якого будуть розроблені автоматизовані тести, має багато web-елементів управління (кнопок, форм, перемикачів тощо).

Спроектована система буде складатись з наборів тестів, кожен з яких відповідатиме за перевірки певної частини функціоналу проекту.

За допомогою вбудованих функцій Puppeteer можна емулювати дії користувача на сторінці та виявляти наявність невидимих елементів. Використання даного фреймворку для автоматизації дозволить отримати достовірну інформацію про роботу всіх модулів.

За допомогою фреймворку можна перевірити роботу посилань, кнопок та випадних списків, текстових полів, перемикачів на сайті, заповнення форм. Це базові елементи будь-якої web-сторінки, саме тому їх було обрано як об’єкт дослідження.

Загальний життєвий цикл автоматизації тестування наведено на рисунку А в додатку А, алгоритм автоматизованого тестування сторінки для створення транзакції наведено на рисунку Б в додатку Б.

Розглянемо приклад автоматизації тест-кейсу. В таблиці 3.1 наведено набір кроків, які необхідно виконати для відтворення тестової ситуації, а саме логування часу затраченого на виконання задачі.

Таблиця 3.1 – Приклад тест-блоку

№	Use-Case	Description	Expected result
1.1	Логування часу затраченого на задачу	Вибір часовго проміжку	Дата обрана коректно
1.2		Перейти до вибору задачі	Вікно з вибором задачі повинно бути успішно відкрито
1.3		Натиснути кнопку «Log time»	Форма для введення даних «Log time» повинна бути успішно відкрита
1.4		Ввести дані у всі обов’язкові до заповнення поля	Дані повинні бути введені та відображені в полях для вводу
1.5		Натиснути на кнопку збереження «Log time»	Створений звіт повинен бути успішно збережений та відображена в таблиці звітів користувача

Даний тест-кейс було автоматизовано програмно, за кожен крок відповідає фрагмент коду. Розглянемо їх детальніше.

```
I.amOnPage('/login/');
I.waitForElement(loginPage.login);
I.waitForElement(loginPage.password);
I.fillField(loginPage.login, 'username');
I.fillField(loginPage.password, 'password');
I.click(loginPage.loginButton);
```

Представлений фрагмент коду відповідає за авторизацію користувача. Авторизація є обов’язковим кроком, тому що без користувацької авторизації усі наступні дії будуть неможливими.

Оскільки Codeceptjs - це інструмент для автоматизації веб додатків, то велика частина роботи з ним це робота з веб елементами (WebElements). WebElements - ні що інше, як DOM об'єкти, що знаходяться на веб сторінці. Для зручного використання всередині проекту, всі необхідні селектори були винесені в окремі файли, які називають Page Object. Вони містять css та xpath визначення усіх елементів сторінки, а також функції які дозволяють виділити елементи з динамічними параметрами. А для того, щоб здійснювати якісь дії над DOM об'єктами / веб елементами необхідно їх точним чином визначити (знайти) [23].

```
I.waitForElement(accountsPage.accountsTab);
I.waitForElement(accountsPage.trasactions);
I.click(accountsPage.trasactions);
I.waitForElement(trasactionsPage.createTransaction);
I.click(trasactionsPage.createTransaction);
I.waitUrlEquals('/createtransaction',10);
I.waitForElement(trasactionsPage.createTransaction.name);
```

Даний фрагмент лістингу реалізує відкриття форми для вибору часовго проміжку, для цього було використаний метод click в який переданий елемент визначений у файлі `tempoPage.js`.

```
I.fillField(tempoPage.logTime.AccounBalance,'Accoun Balance');
I.fillField(tempoPage.logTime.TransferAmount,'Transfer Amount');
I.fillField(tempoPage.logTime.Reference,'Reference');
I.fillField(tempoPage.logTime.PostDate,'Post Date');
```

```
I.fillField(trasactionsPage.createTransaction.PaymentCurrency,'Payment Currency')
;
I.fillField(trasactionsPage.createTransaction.SendInBeneficiaryCurrency,'Send In
Beneficiary Currency');
I.checkOption(trasactionsPage.createTransaction.isUrgent,'Is Urgent');
I.click(accountsPage.trasactions.save);
```

Даний фрагмент лістингу реалізує введення інформації в поля форми для логування часу, для цього було використаний метод fillField в який переданий елемент визначений у файлі `tempoPage.js` та необхідні дані.

Схема роботи програми наведена на рисунку В в додатку В.

Повний лістинг скрипта для автоматизації тестування наведено в додатку Д.

3.4 UML-діаграма використання програми

Візуальне моделювання в UML можна уявити як певний процес рівневого спуску від найбільш загальної і абстрактної концептуальної моделі вихідної системи до логічної, а потім і до фізичної моделі відповідної програмної системи [24]. Для досягнення цих цілей спочатку будується модель у формі так званої діаграми варіантів використання (use case diagram), яка описує функціональне призначення системи або, іншими словами, те, що система буде робити в процесі свого функціонування [25]. Діаграма варіантів використання є вихідним концептуальним поданням або концептуальною моделлю системи в процесі її проектування і розробки.

Мета розробки діаграми варіантів використання:

– визначити і межі і контекст модельованої предметної області на початкових етапах проектування;

– сформулювати вимоги до функціональної поведінки проектованої системи;

– розробити вихідну модель системи для її подальшої деталізації у формі логічних і фізичних моделей.

– підготувати документацію для взаємодії розробників системи з її замовниками і користувачами.

Базові елементи діаграми використання:

– варіант використання;

– актор;

– інтерфейс;

– відносини [26].

Варіант використання застосовується для специфікації загальних особливостей поведінки системи або будь-якої іншої сутності предметної області без розгляду внутрішньої структури цієї сутності. Кожен варіант використання визначає послідовність дій, які повинні бути виконані проектованої системою при взаємодії її з відповідним актором. Діаграма варіантів може доповнюватися текстом, який розкриває зміст складових її компонентів. Такий текст отримав назву примітки або сценарію[27].

Кожен варіант використання відповідає окремому сервісу, який надає модельовану сутність або систему по запиту користувача (актора), тобто визначає спосіб застосування цієї сутності [28]. Сервіс, який ініціалізується за запитом користувача, є закінченою послідовність дій. Це означає, що після того як система закінчить обробку запиту користувача, вона повинна повернутися в початковий стан, в якому готова до виконання наступних запитів.

Актор визначає роль, яку відіграє користувач або будь-яка інша система, що взаємодіє з суб'єктом. При цьому актори служать для позначення узгодженої безлічі ролей, які можуть грати користувачі в процесі взаємодії з проектованої системою. Кожен актор може розглядатися як окрема роль щодо конкретного варіанту використання [29].

Інтерфейс служить для специфікації параметрів моделі, які видимі ззовні без вказівки їх внутрішньої структури. Формально інтерфейс еквівалентний абстрактному класу без атрибутів і методів з наявністю тільки абстрактних операцій. З системно-аналітичної точки зору інтерфейс не тільки відокремлює специфікацію операцій системи від їх реалізації, а й визначає загальні межі проектованої системи. В подальшому інтерфейс може бути уточнений явною вказівкою тих операцій, які специфікують окремий аспект поведінки системи. Важливість інтерфейсів полягає в тому, що вони визначають стикувальні вузли в проектованій системі, що необхідно для організації колективної роботи над проектом. Крім того, специфікація інтерфейсів сприяє "безболісній" модифікації вже існуючої системи при переході на нові технологічні рішення. У цьому випадку змінам піддається тільки реалізація операцій, але ніяк не функціональність самої системи. А це забезпечує сумісність наступних версій програм з початковими при спіральній технології розробки програмних систем.

Графічний символ окремого інтерфейсу може з'єднуватися на діаграмі суцільною лінією з тим варіантом використання, що його підтримує. Суцільна лінія в цьому випадку вказує на той факт, що пов'язаний з інтерфейсом варіант використання повинен реалізовувати всі операції, необхідні для даного інтерфейсу, а можливо й більше [30].

Між компонентами діаграми варіантів використання можуть існувати різні відносини, які описують взаємодію екземплярів одних акторів і варіантів використання з екземплярами інших акторів і варіантів. Один актор може взаємодіяти з декількома варіантами використання. В цьому випадку цей актор звертається до кількох сервісів даної системи. У свою чергу один варіант використання може взаємодіяти з декількома акторами, надаючи для всіх них свій сервіс. Слід зауважити, що два варіанти використання,

визначені для однієї і тієї ж сутності, не можуть взаємодіяти один з одним, оскільки кожен з них самостійно описує закінчений варіант використання цієї сутності. Більш того, варіанти використання завжди передбачають деякі сигнали або повідомлення, коли взаємодіють з акторами за межами системи. У той же час можуть бути визначені інші способи для взаємодії з елементами всередині системи.

В UML є кілька стандартних видів відносин між акторами і варіантами використання:

– ставлення асоціації (association relationship);

– ставлення розширення (extend relationship);

– ставлення узагальнення (generalization relationship);

– ставлення включення (include relationship). [31]

В UML діаграмі використання , що наведена на рисунку Д1, як актори зазначені такі компоненти програми:

– користувач;

– інтерфейс системи;

– система .

3.5 Висновки

В даному розділі проаналізовано інструменти фреймворка CodeceptJS, та визначено їх основні можливості. CodeceptJS має ряд переваг над іншими інструментами для тестування web-додатків, тому що може використовувати всі доступні драйвери браузерів, він працює швидше ніж інші інструменти та має уніфікований набір команд.

Саме тому для подальшої розробки програмного забезпечення було вибрано інструмент CodeceptJS.

4 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Визначення небхідних складових для перевірки розробленого програмного забезпечення

Після завершення розробки програмного забезпечення необхідно впевнитись в корректності його роботи. Отримані результати мають бути вірними і валідними щоб свідчити про правильність роботи системи.

Для цього потрібно створити певний набір тест-кейсів, що буде слугувати контрольним пакетом для перевірки розробленого програмного продукту. Необхідно виконати одні й ті ж кроки мануально, а потім повторити за допомого автоматизованих тестів. Отримані результати потрібно зрівняти і якщо вони будуть однаковими, то це свідчить про правильність роботи програмного забезпечення.

Для перевірки функціоналу з використанням автоматизованих тестів необхідні наступні складові:

доступ до мережі Інтернет, швидкість з’єднання близько 10 мБіт/с;

наявність персонального комп’ютера з ресурсами вище середнього;

встановлений NodeJS 14.16.0;

встановлена програмна бібліотека Puppeteer для NodeJS;

встановлений Інтернет-браузер на базі движка Chromium (в даному випадку Chrome).

4.2 Тестування програмного забезпечення з використанням контрольного набору вхідних даних

Таблиця 4.1 – Результати перевірки тестового набору даних

Тест-кейс	Очікуваний результат	Результат ручного тестування	Результат автоматизованого тестування
1. Авторизуйтесь в системі. Введіть E-mail та Password. Нажміть кнопку Log In.	Користувач повинен бути авторизований у системі.	Успішно	Успішно
2. Змініть пароль.	Пароль має бути змінений.	Успішно	Успішно
3. Внесість затрачений час	Звіт має бути створений.	Успішно	Успішно
4. Відредактуйте внесений звіт.	Звіт має бути відредагований	Успішно	Успішно
5. Оберіть проміжок часу для звітів	Проміжок часу має бути обраний	Успішно	Успішно

Тест-кейс	Очікуваний результат	Результат ручного тестування	Результат автоматизованого тестування
6. Заплануйте час на задачу	Запис про планування має бути створений.	Успішно	Успішно
7. Відредагуйте запланований час на задачу.	Запис має бути відредагований.	Успішно	Успішно
8. Видаліть звіт про внесений час	Звіт має бути видалений.	Успішно	Успішно
9. Перегляньте інформацію про внесений час.	Інформація має бути доступною.	Успішно	Успішно
10. Відкрийте секцію “Timesheet”	Секція ”Timesheet” має відкритися.	Успішно	Успішно

4.3 Висновки

В даному розділі було протестовано правильність роботи розробленого фреймоврку автоматизованого тестування застосунків для Jira.

Якщо проаналізувати отримані результати тестування, можна підтвердити, що програмне забезпечення працює правильно та корректно і може бути застосовано для автоматизованого тестування застосунків для Jira.

5. ЕКОНОМІЧНИЙ РОЗДІЛ

5.1 Технологічний аудит розробленого фреймворку для автоматизованого тестування розширень для Jira

Як було зазначено раніше, фреймворк являє собою інфраструктуру програмних рішень, що полегшує розробку інших, більш складних систем, які потребують свого тестування. Таке тестування неможливо виконати вручну. Тому автоматизоване тестування значно прискорює процес тестування без втрати якості, не вимагаючи значних трудовитрат. Крім того, автоматизоване тестування є більш точним, що є важливим фактором при створенні якісного програмного продукту.

Тому метою нашої роботи було розроблення фреймворку для автоматизованого тестування розширень для Jira шляхом дослідження методів автоматизації тестування програмного забезпечення і вибору найбільш ефективних інструментів та засобів проектування.

Для досягнення поставленої мети були розв'язані такі задачі: сформульовано і описано вимоги до програмної системи, яка буде розроблятися; обрано програмне забезпечення та мову для реалізації фреймворку для автоматизації; визначити технології та підхід до реалізації; реалізовано Page Object архітектуру для представлення класів, методів та локаторів; створити тестові сценарії для тестування продукту; зроблено запуск тестів та представлено результати запуску тестів; описано архітектуру розробленого фреймворку з висновками; визначено переваги використання розробленого фреймворку в ході тестування веб-додатків на прикладі розширень для Jira.

Результатом виконаної магістерської кваліфікаційної роботи є розроблене програмне забезпечення та алгоритм, використання яких дозволить пришвидшити процес тестування розширень для Jira.

Для встановлення комерційного потенціалу розробленого нами програмного забезпечення та алгоритмів, використання яких дозволить пришвидшити процес тестування розширень для Jira, було запрошено 3-х відомих експертів – кандидатів технічних наук, доцентів Гармаша В.В., Маслія Р.В. та Кривогубченка С.Г.

Встановлення комерційного потенціалу розробленого нами програмного забезпечення було здійснено за критеріями, наведеними в таблиці 5.1.

Таблиця 5.1 – Рекомендовані критерії оцінювання технічного рівня та комерційного потенціалу будь-якої розробки і їх бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
Ринкові перспективи					
5	Експлуатаційні витрати значно вищі, ніж в	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатацій	Експлуатаційні витрати трохи нижчі, ніж в	Експлуатаційні витрати значно нижчі, ніж в аналогів

	аналогів		них витрат аналогів	аналогів	
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження таблиці 5.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої	Необхідно отримання великої кількості дозвільних документів на	Процедура отримання дозвільних документів для виробництва	Необхідно тільки повідомлення відповідним органам про	Відсутні будь-які регламентні обмеження на виробництво та реалізацію

	кількості дозвільних документів на виробництво та реалізацію продукту	виробництво та реалізацію продукту, що вимагає значних коштів та часу	та реалізації продукту вимагає незначних коштів та часу	виробництво та реалізацію продукту	продукту

Запрошені експерти оцінили розроблене нами програмне забезпечення для пришвидшення процесу тестування розширень для Jira таким чином (табл. 5.2):

Таблиця 5.2 – Результати технологічного аудиту розробленого мобільного додатка (за шкалою оцінювання 0-1-2-3-4)

Критерії	Прізвище, ініціали експертів		
	Гармаш В.В.	Маслій Р.В.	Кривогубченко С.Г.
	Бали, що їх виставили експерти:		
1	4	3	3
2	3	4	4
3	3	3	4
4	4	4	4
5	4	3	4
6	3	3	4
7	4	4	3
8	4	4	3
9	4	4	3
10	4	4	4
11	4	4	4
12	4	3	4
Сума балів	$СБ_1 = 45$	$СБ_2 = 43$	$СБ_3 = 44$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{3} = \frac{45+43+44}{3} = \frac{132}{3} = 44,00$		

Встановлення комерційного потенціалу розробленого нами програмного забезпечення для пришвидшення процесу тестування розширень для Jira будемо здійснювати на основі рекомендацій, наведених в таблиці 5.3 [32].

Таблиця 5.3 – Рівні комерційного потенціалу будь-якої наукової розробки

Середньоарифметична сума балів $\overline{СБ}$, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 48	Високий

Оскільки середньоарифметична сума балів, що їх виставили експерти, складає 44,00 бали, то це свідчить, що розроблене нами програмного забезпечення для пришвидшення процесу тестування розширень для Jira має рівень комерційного потенціалу, який вважається «високим».

Це пояснюється тим, що при розробці програмного забезпечення нами були розроблені, оптимізовані та застосовані сучасні підходи, методи та алгоритми для ефективної організації автоматизованого тестування розширень для Jira, що дозволило пришвидшити процес тестування розширень для Jira.

5.2 Розрахунок витрат на розроблення фреймворку для автоматизованого тестування розширень для Jira

При розробленні нашого програмного забезпечення були зроблені певні витрати. Зокрема:

А). Основна заробітна плата $З_о$ розробників, яка визначається за формулою:

$$З_о = \frac{М}{Т_р} \cdot t \quad \text{грн,} \tag{5.1}$$

де М – місячний посадовий оклад розробника (дослідника), грн; приймемо, що

М = (6700…25000) грн/місяць;

$T_р$ – число робочих днів в місяці; приймемо $T_р = 20$ день;

t – число днів роботи розробників.

Зроблені розрахунки зведемо до таблиці 5.4:

Таблиця 5.4 – Основна заробітна плата розробників

Найменування посади виконавця	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на оплату праці, грн
1. Науковий керівник магістерської роботи	25000	1250	20 годин	≈ 4167 (при 6-годинному робочому дні)
2. Магістрант-студент-виконавець	2000 (беремо 6700)	335	92	≈ 30820
3. Консультант з економічної частини	16000	800	1,5 години	≈ 200
Загалом				$З_о$ = 35187 грн

Б). Додаткова заробітна плата $З_д$ розробників розраховується як (10…12)% від величини їх основної заробітної плати, тобто:

$$З_д = \alpha \cdot З_о = (0{,}1...0{,}12) \cdot З_о .$$

(5.2)

Приймемо, що $\alpha = 0{,}12$. Тоді для нашого випадку отримаємо:

$$З_д = 0{,}12 \times 35187 = 4222{,}44 \approx 4223 \text{ грн.}$$

В). Нарахування на заробітну плату $НЗП_{зп}$ розробників (дослідників) розраховуються за формулою:

$$НЗП_{зп} = (З_о + З_д) \cdot \frac{\beta}{100},$$

(5.3)

де β – ставка обов'язкового єдиного внеску на державне соціальне страхування, %. β = 22%. Тоді:

$$НЗН_{зп} = (35187 + 4223) \times 0{,}22 = 8670{,}2 \approx 8671 \text{ грн.}$$

Г). Амортизація основних засобів А, які використовувались під час виконання цієї роботи:

$$А = \frac{Ц \cdot Н_а}{100} \cdot \frac{Т}{12} \quad \text{грн,} \tag{5.4}$$

де Ц – загальна балансова вартість основних засобів, грн;

$Н_а$ – річна норма амортизаційних відрахувань. Для нашого випадку можна прийняти, що $Н_а = (2,5...25)\%$;

Т – термін використання основних засобів, місяці.

Зроблені розрахунки зведено в таблицю 5.5.

Таблиця 5.5 – Розрахунок амортизаційних відрахувань

Найменування обладнання, приміщень тощо	Балансова вартість, грн.	Норма амортизації, %	Термін використання, міс.	Величина амортизаційних відрахувань, грн
1. Комп'ютерна техніка, обладнання тощо	52000	25	3,0 (при 90% використанні	2925
2. Приміщення університету, кафедри	20000	3,5	3,0 при 70% використанні	122,5 × 123
Всього				А = 3048 грн

Д). Витрати на матеріали М розраховуються за формулою:

$$М = \sum_{1}^{n} Н_і \cdot Ц_і \cdot К_і - \sum_{1}^{n} В_і \cdot Ц_в \quad \text{грн.,} \tag{5.5}$$

де $Н_і$ – витрати матеріалу *і*-го найменування, кг; $Ц_і$ – вартість матеріалу *і*-го най-менування; $К_і$ – коефіцієнт транспортних витрат, $К_і = (1,1…1,15)$; $В_і$ –

маса відходів матеріалу *і*-го найменування; $Ц_в$ – ціна відходів матеріалу *і*-го найменування; n – кількість видів матеріалів.

Е). Витрати на комплектуючі К розраховуються за формулою:

$$К = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i \quad \text{грн,} \tag{5.6}$$

де $Н_i$ – кількість комплектуючих *і*-го виду, шт.; $Ц_i$ – ціна комплектуючих *і*-го виду; $К_i$ – коефіцієнт транспортних витрат, $К_i = (1{,}1 \ldots 1{,}15)$; n – кількість видів комплектуючих.

Під час виконання роботи загальні витрати на матеріали та комплектуючі склали приблизно 1000 грн.

Ж). Витрати на силову електроенергію $В_е$ розраховуються за формулою:

$$В_е = \frac{В \cdot П \cdot Ф \cdot К_п}{К_д}, \tag{5.7}$$

де В – вартість 1 кВт-год. електроенергії, в 2022 р. В ≈ 3,0 грн/кВт;

П – установлена потужність обладнання, кВт; П = 1,0 кВт;

Ф – фактична кількість годин роботи обладнання, годин.

Приймемо, що Ф = 230 годин;

$К_п$ – коефіцієнт використання потужності; $К_п < 1 = 0{,}74$.

$К_д$ – коефіцієнт корисної дії, $К_д = 0{,}63$.

Тоді витрати на силову електроенергію будуть дорівнювати:

$$В_е = \frac{В \cdot П \cdot Ф \cdot К_п}{К_д} = \frac{3 \cdot 1{,}0 \cdot 230 \cdot 0{,}74}{0{,}63} = 810{,}48 \approx 811 \text{ грн.}$$

И). Інші витрати $В_{інш}$ можна прийняти як (50…300)% від основної заробітної плати розробників, тобто:

$$В_{інш} = (0,5....3) \times З_о.$$

(5.8)

Для нашого випадку отримаємо:

$$В_{інш} = 1,8 \times 35187 = 63336,60 \approx 63337 \text{ грн.}$$

К). Сума всіх попередніх статей витрат складає витрати на виконання нашої роботи (безпосередньо розробником-магістрантом) – В.

$$В = 35187 + 4223 + 8671 + 3048 + 1000 + 811 + 63337 = 116277 \text{ грн.}$$

Л). Загальні витрати на розроблення та можливе впровадження розробленого нами програмного забезпечення $В_{заг}$ розраховуються за формулою:

$$В_{заг} = \frac{В}{\beta},$$

(5.9)

де β – коефіцієнт, який характеризує етап (стадію) виконання цієї роботи. Можна прийняти, що, $\beta \approx 0,9$ [32], оскільки робота потребує незначного доопрацювання.

Тоді: $В_{заг} = \frac{116277}{0,9} = 129196,66$ грн або приблизно 130 тисяч грн.

Тобто прогнозовані загальні витрати на розробку та можливе впровадження (комерціалізацію) розробленого нами програмного забезпечення для пришвидшення процесу тестування розширень для Jira становлять приблизно 130 тисяч грн.

5.3 Розрахунок економічного ефекту від можливої комерціалізації нашої розробки

Економічний ефект від впровадження та можливої комерціалізації розробленого нами програмного забезпечення для пришвидшення процесу

тестування розширень для Jira пояснюється його значно кращими функціональними можливостями. Тому нашу розробку можна реалізовувати на ринку дещо дорожче, ніж аналогічні за функціями розробки. Тобто, якщо одна підписка на рік для компаній (які займаються розробкою додатків до JIRA або подібних систем управління проектами) становить приблизно $99 (або 3960 грн при курсі НБУ в грудні 2022 року 1$ ≈40 грн), а кількість таких підписок становить 1000 шт., то ми можемо реалізовувати нашу розробку за ціною приблизно 4500 грн за одну підписку, або на 4500 – 3960 = 540 грн дорожче.

Аналіз ринку також показав, що потенційна кількість замовників (підписників) розробленого нами програмного забезпечення для пришвидшення процесу тестування розширень для Jira також буде зростати, тобто можна очікувати зростання попиту на нашу розробку принаймні протягом 3-х років після її впровадження.

Тобто, якщо наша розробка буде впроваджена з 1 січня 2024 року (оскільки потребує що доопрацювання), то її результати будуть виявлятися протягом 2024-го, 2025-го та 2026-го років.

Прогноз зростання попиту на нашу розробку складає по роках:

а) 2024 р. – приблизно +200 шт. до базового року;

б) 2025 р. – +350 шт. до базового року;

в) 2026 р. – +500 шт. до базового року.

Можливе збільшення чистого прибутку $\Delta П_i$, що його може отримати потенційний інвестор від комерціалізації, тобто виведення нашої розробки на ринок, становитиме:

$$\Delta П_i = \sum_{1}^{n} (\Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\upsilon}{100}),$$

(5.10)

де $\Delta Ц_о$ – покращення основного якісного показника від впровадження результатів нашої розробки у цьому році. Для нашого випадку це є збільшення ціни реалізації нашої розробки $\Delta Ц_о$ = 4500 - 3960 = + 540 грн (0,54 тисячі грн);

N – основний кількісний показник, який визначає обсяг діяльності у році до впровадження результатів розробки; N = 1000 шт.;

ΔN – покращення основного кількісного показника від впровадження результатів розробки. Таке покращення становитиме по роках: у 2024 році – + 200 шт., у 2025 році +350 шт. та у 2026 році + 500 шт. (відносно базового 2022 року);

$Ц_о$ – основний якісний показник (тобто ціна), який визначає обсяг діяльності у році після впровадження результатів розробки, грн; $Ц_о$= 4500 грн (4,5 тисяч грн);

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки; для нашого випадку n = 3;

λ – коефіцієнт, який враховує сплату податку на додану вартість; $\lambda = 0{,}8333$;

ρ – коефіцієнт, який враховує рентабельність продукту. Рекомендується приймати ρ = (0,2...0,5); візьмемо ρ = 0,5;

υ – ставка податку на прибуток. У 2022-23 роках υ = 18%. У 2024 році також очікуємо на 18%.

Тоді можливе зростання чистого прибутку $\Delta П_1$ для потенційного інвестора протягом першого року від можливого впровадження нашої розробки (2024 р.) складе:

$$\Delta П_1 = [0,54 \cdot 1000 + 4,5 \cdot 200] \cdot 0,8333 \cdot 0,5 \cdot (1 - \frac{18}{100}) \approx 492 \text{ тис. грн.}$$

Можливе зростання чистого прибутку $\Delta П_2$ для потенційного інвестора від можливого впровадження нашої розробки протягом другого (2025 р.) року складе:

$$\Delta П_2 = [0,54 \cdot 1000 + 4,5 \cdot 350] \cdot 0,8333 \cdot 0,5 \cdot (1 - \frac{18}{100}) \approx 723 \text{ тис. грн.}$$

Можливе зростання чистого прибутку $\Delta П_3$ для потенційного інвестора від можливого впровадження нашої розробки протягом третього (2026 р.) року складе:

$$\Delta П_3 = [0,54 \cdot 1000 + 4,5 \cdot 500] \cdot 0,8333 \cdot 0,5 \cdot (1 - \frac{18}{100}) \approx 953 \text{ тис. грн.}$$

Приведена вартість зростання всіх чистих прибутків від можливого впровадження нашої розробки становитиме:

$$ПП = \sum_{1}^{т} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої роботи, грн;

т – період часу, протягом якого виявляються результати впровадженої роботи, роки. Для нашого випадку т = 3 роки;

τ – ставка дисконтування. Приймемо τ = 0,10 (10%);

t – період часу від моменту початку розроблення програмного забезпечення для пришвидшення процесу тестування розширень для Jira до моменту отримання можливих чистих прибутків потенційним інвестором.

Тоді приведена вартість зростання всіх можливих чистих прибутків ПП, що їх може отримати потенційний інвестор від комерціалізації нашої розробки, складе:

$$ПП = \frac{492}{(1+0,1)^2} + \frac{723}{(1+0,1)^3} + \frac{953}{(1+0,1)^4} \approx 407 + 543 + 651 = 1601 \text{ тисяч грн.}$$

Теперішня вартість інвестицій PV, що повинні бути вкладені інвестором для реалізації і комерціалізації нашої розробки: PV = (1,0…5) × $В_{заг}$.

Для нашого випадку PV = (1,0…5) × 130 = 3 × 130 = 390 тисяч грн.

Розраховуємо абсолютний ефект від можливих вкладених інвестицій $Е_{абс}$.

$$Е_{абс} = ПП - PV, \quad (5.12)$$

де ПП – приведена вартість збільшення всіх чистих прибутків для інвестора від можливого впровадження нашої розробки, грн;

PV – теперішня вартість інвестицій PV = 390 тисяч грн.

Абсолютний ефект від можливого впровадження нашої розробки (при прогнозованому ринку збуту) за три роки складе:

$$Е_{абс} = 1601 - 390 = 1211 \text{ тисяч грн.}$$

Оскільки $Е_{абс} > 0$, то комерціалізація нашої розробки може бути доцільною.

Далі розрахуємо внутрішню дохідність $Е_в$ вкладених інвестицій:

$$Е_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1, \quad (5.13)$$

де $Е_{абс}$ – абсолютний ефект вкладених інвестицій; $Е_{абс}$ = 1211 тис. грн;

PV –теперішня вартість початкових інвестицій PV = 390 тис. грн;

$Т_ж$ – життєвий цикл розробки, роки.

$Т_ж$ = 4 років (2023-й, 2024-й, 2025-й, 2026-й роки)

Для нашого випадку отримаємо:

$$Е_в = \sqrt[4]{1 + \frac{1211}{390}} - 1 = \sqrt[4]{1 + 3,1051} - 1 = \sqrt[4]{4,1051} - 1 = 1,423 - 1 = 0,423 = 42,3\%.$$

Далі визначимо ту мінімальну дохідність, нижче за яку потенційному інвестору не вигідно буде займатися комерціалізацією нашої розробки.

Мінімальна дохідність або мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau_{мін} = d + f, \quad (5.14)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d = (0,10...0,12);

f – показник, що характеризує ризикованість вкладень; f = (0,1...0,30).

Приймемо f = 0,30.

Для нашого випадку отримаємо:

$$\tau_{мін} = 0,12 + 0,30 = 0,42 \text{ або } \tau_{мін} = 42\%.$$

Оскільки величина $E_в$ = 42,3% > $\tau_{мін}$ = 42%, то потенційний інвестор у принципі може бути зацікавлений у фінансуванні та комерціалізації нашої розробки.

Далі розраховуємо термін окупності коштів, вкладених у можливу комерціалізацію розробленого нами програмного забезпечення для пришвидшення процесу тестування розширень для Jira.

Термін окупності $T_{ок}$ розраховується за формулою:

$$T_{ок} = \frac{1}{E_в}. \quad (5.15)$$

Для нашого випадку термін окупності $T_{ок}$ коштів становитиме:

$$T_{ок} = \frac{1}{0,423} = 2,36 \text{ років} < 3 \text{ років},$$

що свідчить про потенційну доцільність комерціалізації розробленого нами програмного забезпечення для пришвидшення процесу тестування розширень для Jira.

Далі проведено моделювання залежності величини внутрішньої дохідності вкладених потенційних інвестицій від рівня інфляції в країні. Як

відомо, на наступні роки, на жаль, прогнозується високий рівень інфляції (в межах 30% і вище), що обумовлюється військовою агресією росії проти України.

Прийнявши прогнозований рівень інфляції на наступні роки у 30% отримаємо:

$$ПП = \frac{492}{(1+0,3)^2} + \frac{723}{(1+0,3)^3} + \frac{953}{(1+0,3)^4} \approx 291 + 329 + 334 = 954 \text{ тисяч грн.}$$

Тоді абсолютний ефект від можливого впровадження нашої розробки за три роки складе:

$$E_{абс} = 954 - 390 = 564 \text{ тисяч грн.}$$

Внутрішня дохідність $E_в$ вкладених інвестицій становитиме:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1,$$

де $E_{абс}$ – абсолютний ефект вкладених інвестицій; $E_{абс}$ = 564 тисячі грн;

PV –теперішня вартість початкових інвестицій PV = 390 тисяч грн.

Для нашого випадку отримаємо:

$$E_в = \sqrt[4]{1 + \frac{564}{390}} - 1 = \sqrt[4]{1 + 1,4461} - 1 = \sqrt[4]{2,4461} - 1 = 1,25 - 1 = 0,25 = 25,0\%.$$

Зроблені розрахунки у вигляді графіків наведено на рис. 5.1.

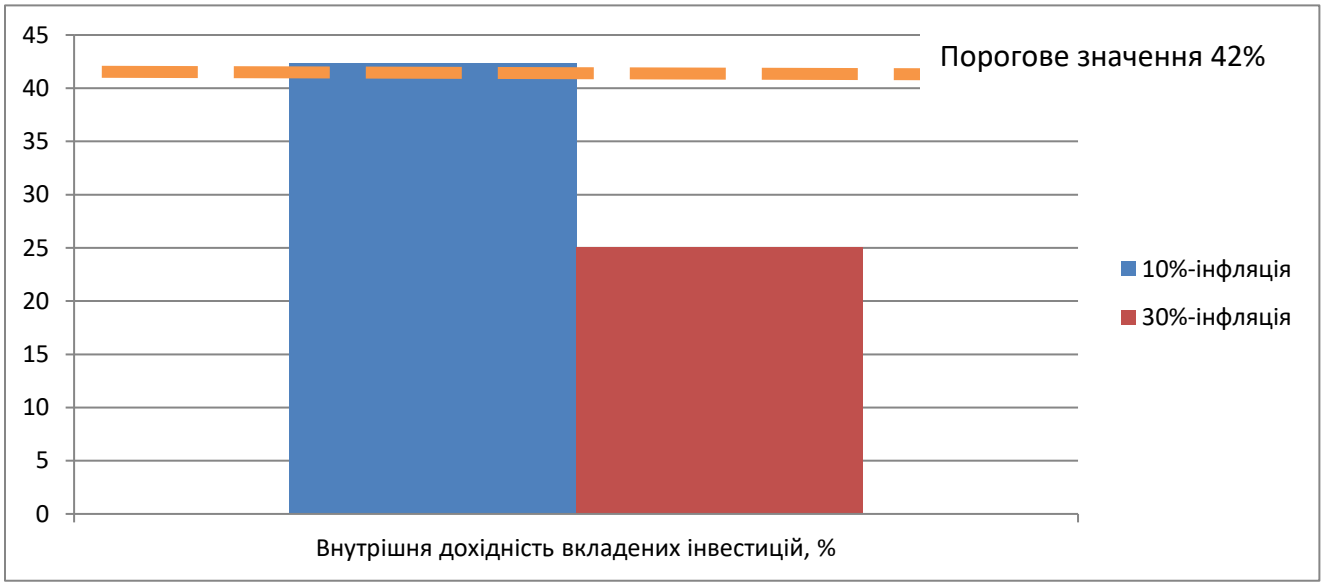

Рисунок 5.1 – Моделювання залежності величини внутрішньої дохідності потенційних інвестицій від рівня інфляції в країні

Аналіз графіка на рис 5.1 показує, що при рівні інфляції в 10% величина внутрішньої дохідності інвестицій становить $E_в$ = 42,3%, що дещо більше порогового значення $\tau_{мін}$ = 42%. і тому комерціалізація нашої розробки може бути доцільною. При рівні інфляції в 30% величина внутрішньої дохідності інвестицій, вкладених в комерціалізацію нашої розробки, становить всього $E_в$ = 25,0%, що менше порогового значення $\tau_{мін}$ = 42%, і тому комерціалізація нашої розробки потенційним інвестором може бути проблематичною. Для прийняття остаточного рішення потрібні додаткові розрахунки.

Результати виконаної економічної частини магістерської кваліфікаційної роботи зведено у таблицю:

Показники	Задані у ТЗ	Досягнуті у магістерській кваліфікаційній роботі	Висновок
1. Витрати на розробку	Не більше 150 тис. грн	130 тис. грн.	Досягнуто
2. Абсолютний ефект від	Не менше 1000 тисяч	1211 тисяч грн	Виконано

впровадження розробки (в майбутній вартості грошей), тисяч грн	грн (за три роки)		
3. Внутрішня дохідність інвестицій, %	не менше 42%	42,3%	Досягнуто
4. Термін окупності інвестицій, роки	до 3-ти років	2,36 років	Виконано

Таким чином, основні техніко-економічні показники розробленого нами програмного забезпечення для пришвидшення процесу тестування розширень для Jira, визначені у технічному завданні, виконані.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Паламарчук К.А. Аналіз сучасних інструментів для автоматизованого тестування web-додатків. / К. А. Паламарчук // Матеріали доповідей науково-технічної конференції підрозділів Вінницького національного технічного університету, 10 березня. – Вінниця : ВНТУ, 2021.

2. 10 Преимуществ автоматизации тестирования [Електронний ресурс]. Режим доступу: http://getbug.ru/10-preimushhestv-avtomatizatsii-testirovaniya/ – Назва з екрану.

3. What Is Automation Testing (Ultimate Guide To Start Test Automation) [Електронний ресурс]. Режим доступу: https://www.softwaretestinghelp.com/automation-testing-tutorial-1/ – Назва з екрану.

4. Савин Р. Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах / Р. Савин. – М. : Дело, 2007. – 312 с.

5. What Is Software Testing | Everything You Should Know[Електронний ресурс]. Режим доступу: https://www.softwaretestingmaterial.com/software-testing/ – Назва з екрану.

6. Turevska O., Shubin I. Improving the automated testing of Web-based services by reflecting the social habits of target audiences. Information Technologies in Innovation Business Conference (ITIB), 2015, с. 93-96.

5. Портал Про Тестинг. Тестирование Программного Обеспечения / [Електронний ресурс]: – Режим доступу http://www.protesting.ru (дата звернення: 27.02.2021)

6. Карбасов Д., Пасевич К. Тестирование ПО с использованием инструментов HP Mercury [Електронний ресурс]: Москва, 2008-2018. / URL: http://softwaretesting.ru/library/vendors/162-hp-mercury#6 (дата звернення 23.02.2021)

7. Савкин В. Принципы управления качеством программ [Електронний ресурс]: Москва. Открытые системы, 2008-2018. URL: http://www.osp.ru/os/2008/06/5344965 (дата звернення: 27.02.2021)

8. Vechur O., Shevchenko D. Evaluation and optimization of software product infrastructure. East European Scientific Journal. Printed in the «Jerozolimskie 85/21, 02- 001 Warsaw, Poland», 2020

9. Selenium IDE. Documentation / [Електронний ресурс]: – Режим доступу https://www.selenium.dev/documentation/en/webdriver/ (дата звернення: 28.02.2021)

10. Katalon Automation Recorder Quickstart / [Електронний ресурс]: – Режим доступу https://www.katalon.com/resources-center/blog/katalon-automation-recorder/ (дата звернення: 10.03.2021)

11. Micro Focus or one of its affiliates. UFT One / [Електронний ресурс]: – Режим доступу https://www.microfocus.com/en-us/products/uft-one/overview (дата звернення: 10.03.2021)

12. Watir / [Електронний ресурс]: – Режим доступу http://watir.com/ (дата звернення: 12.03.2021)

13. SmartBear / [Електронний ресурс]: https://smartbear.com/ (дата звернення: 12.03.2021)

14. Robot Framework / [Електронний ресурс]: – Режим доступу https://robotframework.org/#learning

15.Codecept JS [Електронний ресурс]: – Режим доступу https://codecept.io/ – Назва з екрану.

16. Codecept JS [Електронний ресурс]: – Режим доступу: https://codeception.com/docs/02-GettingStarted

17. CodeceptJS — современные end2end тесты для NodeJS [Електронний ресурс]: – Режим доступу: https://habr.com/ru/post/319656/

18. Codecept JS [Електронний ресурс]: – Режим доступу: https://github.com/codeceptjs/examples

19. Codecept and testing [Електронний ресурс]: – Режим доступу: https://www.npmjs.com/package/codeceptjs

20. Коли тестування повинно бути автоматизованим? [Електронний ресурс]. Режим доступу: https://www.stickyminds.com/article/when-should-test-be-automated– Назва з екрану.

21. Колмогоров К. А. Роль автоматического тестирования в процессе разработки программного обеспечения / К. А. Колмогоров // Компьютерные инструменты в образовании. – 2006. – № 3. – С. 29-32.

22. Гребенюк В. М. Оценка целесообразности внедрения автоматизированного тестирования / В. М. Гребенюк // Науковедение. – 2013. – № 1. – С. 1–8.

23. Современная автоматизация web-приложений [Електронний ресурс]. Режим доступу: http://seleniumforall.blogspot.com/2013/02/web.html. – Назва з екрану.

24. Диаграма вариантов использования [Електронний ресурс]. Режим доступу: http://khpi-iip.mipk.kharkiv.edu/library/case/leon/gl4/gl4.html

25. UML прецедент – кто такой актер? [Електронний ресурс]. Режим доступу: https://coderoad.ru/21062245/UML-%D0%BF%D1%80%D0%B5%D1%86%D0%B5%D0%B4%D0%B5%D0%BD%

D1%82%D0%BA%D1%82%D0%BE%D1%82%D0%B0%D0%BA%D0%BE%D0%B9%D0%B0%D0%BA%D1%82%D0%B5%D1%80

26. Діаграма варіантів використання мови UML [Електронний ресурс]. Режим доступу: https://vunivere.ru/work23825. – Назва з екрану.

27. UML-диаграммы классов [Електронний ресурс]. Режим доступу: https://prog-cpp.ru/uml-classes/

28. Канонические диаграммы языка UML и особенности их графического представления. [Електронний ресурс]. Режим доступу: https://helpiks.org/9-21800.html

29. Automated UI testing that covers you from device cloud to packaged apps [Електронний ресурс]. Режим доступу: https://smartbear.com/product/testcomplete/overview/

30. WebDriver. Обзор и принцип работы [Електронний ресурс]. Режим доступу: https://kreisfahrer.gitbooks.io/selenium-webdriver/content/webdriver_intro/webdriver_obzor_i_printsip_raboti.html. – Назва з екрану.

31. Web-элементы управления [Електронний ресурс]: – Режим доступу: http://www.web.starport.ru/usability/pg0_16.php – Назва з екрану.

32. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт. / Укладачі В.О. Козловський, О.Й. Лесько, В.В.Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

ДОДАТКИ

Додаток А
(обов’язковий)
Технічне завдання

ЗАТВЕРДЖУЮ
Завідувач кафедри АІІТ
д.т.н., професор
О.В. Бісікало

«____»_____ 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на магістерську кваліфікаційну роботу
«Розробка фреймворку для автоматизованого тестування розширень в системі Jira»
08-31.МКР.010.02.000 ТЗ

Керівник роботи:

д.т.н.,проф. каф. АІІТ

Квєтний Р.Н.

«____»_____ 2022 р.

Виконавець:

ст.гр. 1АКІТ-21м

Паламарчук К.А.

«____»_____ 2022 р.

Вінниця ВНТУ – 2022

1.Назва та галузь застосування

Програмне забезпечення для автоматизованого тестування розширень в системі Jira. Розроблений засіб дозволяє підвищити контроль якості автоматизованого тестування та зменшити час, що витрачається на регресивне тестування.

2. Підстава для розробки

Інструменти для автоматизованого тестування додатків знаходяться в стадії активного розвитку. З кожним роком зростає популярність автоматизованого тестування. В зв’язку з цим збільшується попит на спеціалістів в даній галузі. Саме тому виникає необхідність розробки програмного забезпечення для автоматизованого тестування , що дозволяє підвищити контроль якості та зменшити час, що витрачається на регресивне тестування. Підставою для розробки є наказ ВНТУ №213 від 14. 09.2022.

3. Мета та призначення розробки

Головною метою даної роботи є розробка фреймворку для автоматизованого тестування розширень для Jira шляхом дослідження методів автоматизації тестування програмного забезпечення і вибору найбільш ефективних інструментів та засобів проектування.

4. Джерела розробки

1. Паламарчук К.А. Аналіз сучасних інструментів для автоматизованого тестування WEB-додатків / К. А. Паламарчук // Матеріали конференції “XLX Науково-технічна конференція факультету комп'ютерних систем і автоматики (2021)” Вінницького національного технічного університету у 2021 році.

2. Billinghurst M. Emerging Technologies of Augmented Reality: Interfaces and Design./ Billinghurst Mark, Thomas Bruce, 2007. - p. 209 - ISBN 978-1-5990-4066-0.

3. Роман Савин «Тестирование dot com» [Електронний ресурс]. Режим доступу: https://itexts.net/avtor-roman-savin/157563-testirovanie-dot-com-roman-savin/read/page-1.html. – Назва з екрану.

4. Дастин Э. Автоматизированное тестирование программного обеспечения / Элфрид Дастин, Джефф Рэшка, Джон Пол. – М. : «Вильямс», 2010. — 464 с

5.CodeceptJS[Електронний ресурс]. Режим доступуhttps: codecept.io/ – Назва з екрану

5. Показники призначення

Необхідні вхідні дані для коректної роботи програми:

- доступ до мережі Інтернет;
- наявність облікового запису в системі Jira;
- встановлений додаток Tempo.

Результати роботи:

- перевірка коректності залогованого часу;
- звіт про коректність залогованого часу.

6. Економічні показники

- прогнозовані витрати на розробку – не більше 75 тис. грн;
- абсолютна ефективність розробки – не менше 2197,5 тис. грн щороку;
- термін окупності витрат для виробника – не більше 1,416 року.

7. Стадії розробки

1. Розділ 1 «Аналіз проблемної області та постановка задачі» має бути виконаний до 26.09.2022.

2. Розділ 2 «Аналіз існуючих засобів в області автоматизованого тестування» має бути виконаний до 21.10.2022.

3. Розділ 3 «Розробка фреймворку для автоматизованого тестування на основі codeceptjs» має бути виконаний до 11.11.2022.

4. Розділ 4 «Тестування розробленого програмного забезпечення.» має бути виконаний до 02.12.2022.

5. Економічний розділ має бути виконаний до 02.12.2022.

8. Порядок контролю та приймання

1. Рубіжний контроль. Провести до 15.11.2022.

2. Попередній захист магістерської кваліфікаційної роботи. Провести до 16.12.2022.

3. Захист магістерської кваліфікаційної роботи. Провести в період з 19.12.2022 до 23.12.2022.

Додаток Б(обов'язковий).

Графічна частина

Зав. кафедри АІІТ _____ професор О.В. Бісікало

(підпис) (прізвище та ініціали)

Науковий керівник _____ професор Р.Н. Квєтний

(підпис) (прізвище та ініціали)

Тех.контроль _____ професор Р.Н. Квєтний

(підпис) (прізвище та ініціали)

Норм.контроль _____ професор Р.Н. Квєтний

(підпис) (прізвище та ініціали)

Рецензент _____ професор В.М. Дубовой

(підпис) (прізвище та ініціали)

Ст. групи 1АКІТ-21м _____ К. А. Паламарчук

(підпис) (прізвище та ініціали)

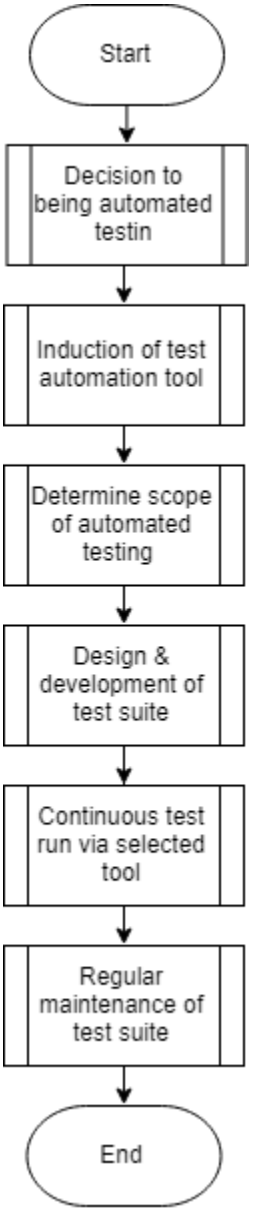

Рисунок А – Життєвий цикл автоматизації тестування

Додаток Б (обов’язковий). Схема програми

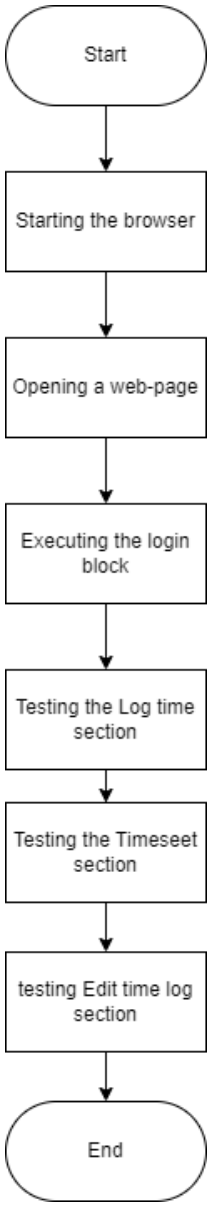

Рисунок Б – Схема програми

Додаток В (обов’язковий). Схема логування часу

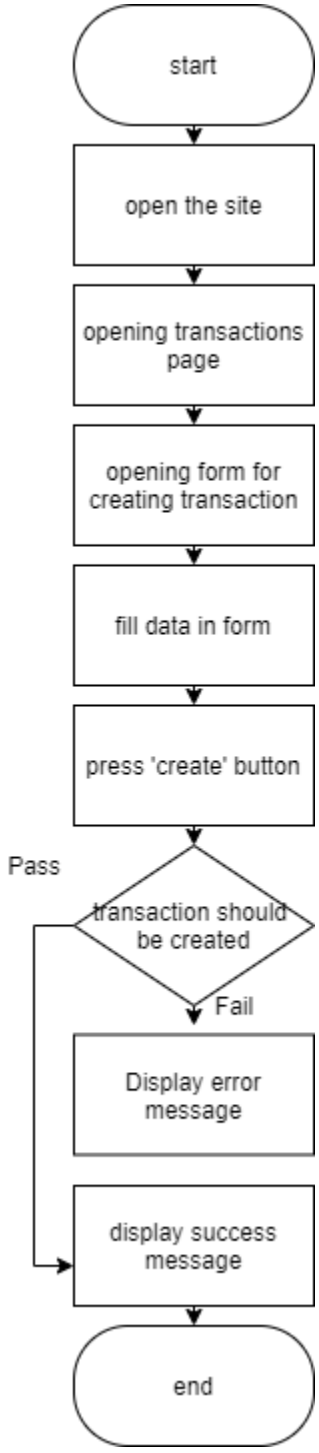

Рисунок В – Схема логування часу

Додаток Г (обов’язковий). Діаграма варіантів використання

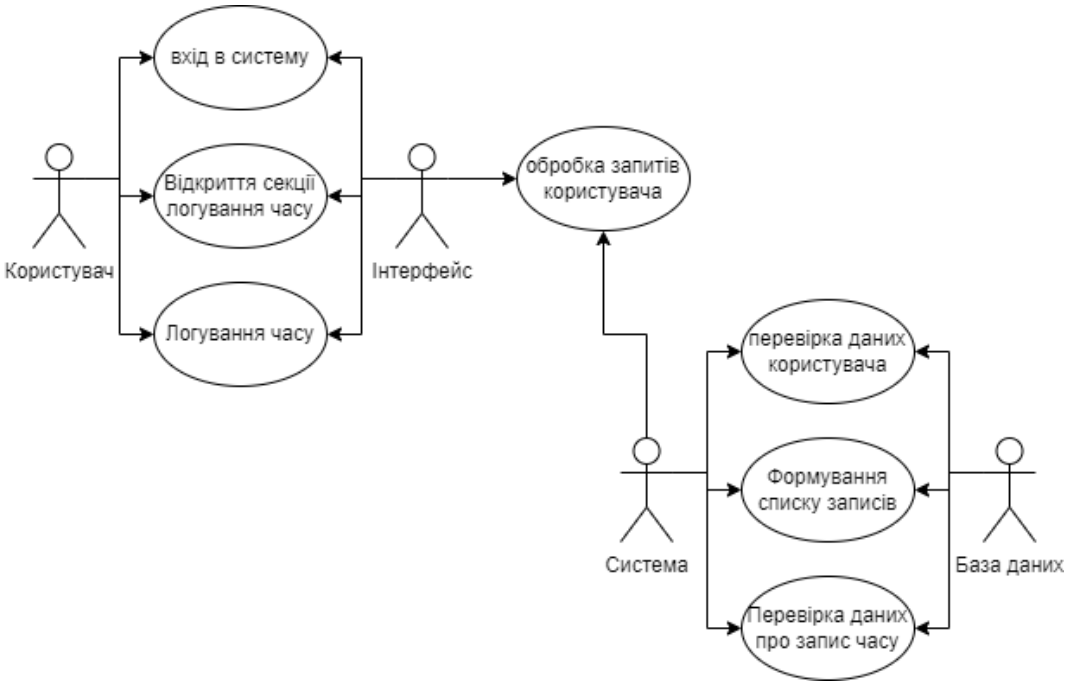

Рисунок Г - Діаграма варіантів використання

Додаток Д (обов'язковий). Лістинг програми

```
Scenario('@login Log in with valid credentials', ({ I }) => {
    I.amOnPage('/login/');
    I.waitForElement(loginPage.login);
    I.waitForElement(loginPage.password);
    I.fillField(loginPage.login, 'username');
    I.fillField(loginPage.password, 'password');
    I.click(loginPage.loginButton);
    I.waitForElement(accountsPage.accountsTab);
});

Scenario('@passwordRecovery Password recovery', ({ I }) => {
    I.amOnPage('/login/');
    I.waitForElement(loginPage.forgotPassword);
    I.click(loginPage.forgotPassword);
    I.waitUrlEquals('/passwordrecovery',10);
    I.waitForElement(loginPage.email);
    I.fillField(loginPage.email, 'email@mail.com');
    I.click(loginPage.reset);
    I.waitForElement(popups.success);
    I.waitForAnEmail('Your password was reseted');
});

Scenario('@createTransaction Creating Transaction(Non-Urgent type)', ({ I }) => {
    I.amOnPage('/login/');
    I.waitForElement(loginPage.login);
    I.waitForElement(loginPage.password);
    I.fillField(loginPage.login, 'username');
    I.fillField(loginPage.password, 'password');
    I.click(loginPage.loginButton);
    I.waitForElement(accountsPage.accountsTab);
    I.waitForElement(accountsPage.trasactions);
    I.click(accountsPage.trasactions);
    I.waitForElement(trasactionsPage.createTransaction);
    I.click(trasactionsPage.createTransaction);
    I.waitUrlEquals('/createtransaction',10);
    I.waitForElement(trasactionsPage.createTransaction.name);
    I.fillField(trasactionsPage.createTransaction.name,'name');
    I.fillField(trasactionsPage.createTransaction.address,'Address');
    I.fillField(trasactionsPage.createTransaction.city,'City');
    I.fillField(trasactionsPage.createTransaction.Country,'Country');
```

```
    I.fillField(trasactionsPage.createTransaction.AccountNumber,'Account Number')
;
    I.fillField(trasactionsPage.createTransaction.BankCodeType,'Bank Code Type');
    I.fillField(trasactionsPage.createTransaction.BankCodeType,'Bank Code Type');
    I.fillField(trasactionsPage.createTransaction.SWIFT,'SWIFT');
    I.fillField(trasactionsPage.createTransaction.BankName,'Bank Name');
    I.fillField(trasactionsPage.createTransaction.File,'File');
    I.fillField(trasactionsPage.createTransaction.AccountCurrency,'Account Curren
cy');
    I.fillField(trasactionsPage.createTransaction.AccounBalance,'Accoun Balance')
;
    I.fillField(trasactionsPage.createTransaction.TransferAmount,'Transfer Amount
');
    I.fillField(trasactionsPage.createTransaction.Reference,'Reference');
    I.fillField(trasactionsPage.createTransaction.PostDate,'Post Date');
    I.fillField(trasactionsPage.createTransaction.PaymentCurrency,'Payment Curren
cy');
    I.fillField(trasactionsPage.createTransaction.SendInBeneficiaryCurrency,'Send
 In Beneficiary Currency');
    I.checkOption(trasactionsPage.createTransaction.isUrgent,'Is Urgent');
    I.click(accountsPage.trasactions.save);
    I.waitForElement(popups.success);

});

Scenario('@declineTransaction Decline the transaction', ({ I }) => {
    I.amOnPage('/login/');
    I.waitForElement(loginPage.login);
    I.waitForElement(loginPage.password);
    I.fillField(loginPage.login, 'username');
    I.fillField(loginPage.password, 'password');
    I.click(loginPage.loginButton);
    I.waitForElement(accountsPage.trasactions);
    I.click(accountsPage.trasactions);
    I.click(trasactionsPage.declineTransactionWithId(trasactionId));
    I.waitForElement(popups.declineTransaction);
    I.fillField(popups.declineComment);
    I.click(popups.confirm);
    I.waitForElement(popups.success);
});

Scenario('@selectDateRabge Selecting beneficiaries date range', ({ I }) => {
    I.amOnPage('/login/');
    I.waitForElement(loginPage.login);
    I.waitForElement(loginPage.password);
    I.fillField(loginPage.login, 'username');
    I.fillField(loginPage.password, 'password');
```

```
    I.click(loginPage.loginButton);
    I.waitForElement(accountsPage.beneficiaries);
    I.click(accountsPage.beneficiaries);
    I.waitForElement(beneficiariesPage.calendar);
    I.selectDate(beneficiariesPage.calendar,Date.now);
    I.click(beneficiariesPage.calendarConfirm);
});

Scenario('@createSubUser Creating new SubUser', ({ I }) => {
    I.amOnPage('/login/');
    I.waitForElement(loginPage.login);
    I.waitForElement(loginPage.password);
    I.fillField(loginPage.login, 'username');
    I.fillField(loginPage.password, 'password');
    I.click(loginPage.loginButton);
    I.waitForElement(accountsPage.accountsTab);
    I.waitForElement(accountsPage.createUser);
    I.click(accountsPage.createUser);
    I.waitForElement(accountsPage.createUserForm.email);
    I.fillField(accountsPage.createUserForm.email, 'email@mail.com');
    I.fillField(accountsPage.createUserForm.phoneNumber, '11111111111');
    I.fillField(accountsPage.createUserForm.firstName,'test');
    I.fillField(accountsPage.createUserForm.lastName, 'test');
    I.checkOption(accountsPage.createUserForm.readOnly);
    I.click(accountsPage.createUserForm.submit);
    I.waitForElement(popups.success);
});

Scenario('@viewTransactionsInfo Viewing transaction info', ({ I }) => {
    I.amOnPage('/login/');
    I.waitForElement(loginPage.login);
    I.waitForElement(loginPage.password);
    I.fillField(loginPage.login, 'username');
    I.fillField(loginPage.password, 'password');
    I.click(loginPage.loginButton);
    I.waitForElement(accountsPage.trasactions);
    I.click(accountsPage.trasactions);
    I.click(trasactionsPage.viewTransactionWithId(trasactionId));
    I.waitForElement(popups.transactionInfo);

});

Scenario('@createBeneficiary Creating new beneficiary(Company type)', ({ I }) =>
{
    I.amOnPage('/login/');
    I.waitForElement(loginPage.login);
    I.waitForElement(loginPage.password);
```

```
    I.fillField(loginPage.login, 'username');
    I.fillField(loginPage.password, 'password');
    I.click(loginPage.loginButton);
    I.waitForElement(accountsPage.accountsTab);
    I.click(beneficiariesPage.createBeneficiary);
    I.waitForElement(beneficiariesPage.selectType);
    I.selectOption(beneficiariesPage.selectType,'company');
    I.fillField(beneficiariesPage.name);
    I.selectOption(beneficiariesPage.currency, 'USD');
    I.fillField(beneficiariesPage.address);
    I.fillField(beneficiariesPage.city);
    I.fillField(beneficiariesPage.state);
    I.fillField(beneficiariesPage.country);
    I.fillField(beneficiariesPage.zipcode);
    I.fillField(beneficiariesPage.AccountNumber);
    I.selectOption(beneficiariesPage.customer, 'Test');
    I.checkOption(beneficiariesPage.validated);
    I.fillField(beneficiariesPage.bankForm.name);
    I.fillField(beneficiariesPage.bankForm.address);
    I.fillField(beneficiariesPage.bankForm.city);
    I.selectOption(beneficiariesPage.bankForm.country, 'United States');
    I.fillField(beneficiariesPage.bankForm.SWIFT);
    I.fillField(beneficiariesPage.bankForm.bankCode);
    I.selectOption(beneficiariesPage.bankForm.type, 'private');
    I.click(beneficiariesPage.validated.save);
    I.waitForElement(popups.success);
});

Scenario('@editBeneficiary Editing non-validated Beneficiary', ({ I }) => {
    I.amOnPage('/login/');
    I.waitForElement(loginPage.login);
    I.waitForElement(loginPage.password);
    I.fillField(loginPage.login, 'username');
    I.fillField(loginPage.password, 'password');
    I.click(loginPage.loginButton);
    I.waitForElement(accountsPage.accountsTab);
    I.click(beneficiariesPage.editBeneficiaryWithId(id));
    I.waitForElement(beneficiariesPage.selectType);
    I.selectOption(beneficiariesPage.selectType,'company');
    I.fillField(beneficiariesPage.name);
    I.selectOption(beneficiariesPage.currency, 'USD');
    I.fillField(beneficiariesPage.address);
    I.fillField(beneficiariesPage.city);
    I.fillField(beneficiariesPage.state);
    I.fillField(beneficiariesPage.country);
    I.fillField(beneficiariesPage.zipcode);
    I.fillField(beneficiariesPage.AccountNumber);
```

```
    I.selectOption(beneficiariesPage.customer, 'Test');
    I.checkOption(beneficiariesPage.validated);
    I.fillField(beneficiariesPage.bankForm.name);
    I.fillField(beneficiariesPage.bankForm.address);
    I.fillField(beneficiariesPage.bankForm.city);
    I.selectOption(beneficiariesPage.bankForm.country, 'United States');
    I.fillField(beneficiariesPage.bankForm.SWIFT);
    I.fillField(beneficiariesPage.bankForm.bankCode);
    I.selectOption(beneficiariesPage.bankForm.type, 'private');
    I.click(beneficiariesPage.validated.save);
    I.waitForElement(popups.success);
});

Scenario('@openJournal "Journal for selected account" section opening', ({ I }) =
> {
    I.amOnPage('/login/');
    I.waitForElement(loginPage.login);
    I.waitForElement(loginPage.password);
    I.fillField(loginPage.login, 'username');
    I.fillField(loginPage.password, 'password');
    I.click(loginPage.loginButton);
    I.waitForElement(accountsPage.trasactions);
    I.click(accountsPage.viewJournalOfUserWithId(id));
    I.waitForElement(accountsPage.journal.adjustments);
    I.waitForElement(accountsPage.journal.calendar);
    I.waitForElement(accountsPage.journal.search);
    I.waitForElement(accountsPage.journal.print);
    I.waitForElement(accountsPage.journal.transactionType);
    I.waitForElement(accountsPage.journal.Reference);
    I.waitForElement(accountsPage.journal.PostDate);
    I.waitForElement(accountsPage.journal.amount);
    I.waitForElement(accountsPage.journal.status);

});
```