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

ФІС
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

(назва освітнього ступеня)

на тему: *"Розробка системи для написання та менеджменту автоматизованих BDD тестів на базі Java та Cucumber"*

Виконав(ла): студент(ка) *VI* курсу, групи СПм-61
спеціальності *121 Інженерія програмного*

(шифр і назва спеціальності)

		Левицький Р.В.
	(підпис)	(прізвище та ініціали)
Керівник		
	(підпис)	(прізвище та ініціали)
Нормоконтроль		
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		
	(підпис)	(прізвище та ініціали)
Рецензент		
	(підпис)	(прізвище та ініціали)

Тернопіль
2020

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

Факультет _____
(повна назва факультету)

Кафедра _____
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (прізвище та ініціали)
« » 20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня _____
(назва освітнього ступеня)

за спеціальністю _____
(шифр і назва спеціальності)

студенту _____
(прізвище, ім'я, по батькові)

1. Тема роботи _____

Керівник роботи _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи _____

4. Зміст роботи (перелік питань, які потрібно розробити)

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Магістерська робота на тему «Розробка системи для написання та менеджменту автоматизованих BDD тестів на базі Java та Cucumber». Левицького Руслана Васильовича. Тернопільський національний технічний університет імені Івана Пулюя, Факультет комп'ютерно-інформаційних систем і програмної інженерії, Кафедра програмної інженерії, група СПм–61, Тернопіль, 2020. С. – 84.

Актуальність теми роботи полягає в тому, що BDD методологія розробки програмного забезпечення, при всіх її перевагах є доволі складною в запровадженні на живому проекті, а також має декілька істотних недоліків, обійти які, при роботі з даним підходом важко, а інколи навіть неможливо.

Метою дипломної роботи є розробка інформаційної системи, для написання та менеджменту автоматизованих BDD тестів, що допомагає членам команди, яка працює над проектом розробки програмного продукту швидше та ефективніше інтегрувати BDD методологію в процес розробки, та дозволить повною мірою використовувати всі її переваги

Методи та програмні засоби, використані при виконанні розробки системи: мова програмування Java, тестовий фреймворк TestNG, Selenium Web Driver, BDD фреймворк Cucumber.

Результатом роботи є розроблений програмний продукт для автоматизованого тестування базових елементів web-сторінки з використанням поведінкового методу тестування, реалізованого за допомогою фреймворку Cucumber.

Ключові слова: ТЕСТУВАННЯ, АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ, SELENIUM WEB DRIVER, BDD, CUCUMBER

ABSTRACT

Master's thesis on "Development of systems for writing and managing automated BDD tests based on Java and cucumber." Levitsky Ruslan Vasilyevich. Ivan Pulyuy Ternopil National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, SPm Group - 61, Ternopil, 2020. P. - 84.

The current relevance of the topic is that the BDD software development methodology, with all its advantages leads to the compilation of an invitation to a live project, there are also several significant shortcomings that are visited which, working with this approach is important and even impossible.

The aim of the thesis is to develop an information system for writing and managing automated BDD tests, which helps team members working on a software development project to quickly and efficiently integrate BDD methodology into the box development process, and allows you to take full advantage of it.

Methods and software used in performing system development: Java programming language, TestNG test framework, Selenium Web Driver, BDD Cucumber framework.

The result is a developed software product for automated testing of basic elements of web pages using a behavioral testing method implemented using the Cucumber framework.

Keywords: TESTING, TESTING AUTOMATION, SELENIUM WEB DRIVER, BDD, CUCUMBER

ЗМІСТ

ВСТУП

Тестування програмного забезпечення визначається як процес забезпечення якомога вищої якості програмного забезпечення для користувачів та тестування продукту, щоб запобігти тому, що будь-які проблеми з програмним забезпеченням стануть вузьким місцем у розробленій системі.

Тестування можна дуже умовно розділити на ручне та автоматизоване тестування.

Ручне тестування – це коли тестери виконують тестові кейси вручну без використання будь-яких засобів автоматизації. Вони виконують роль кінцевого користувача і намагаються якомога швидше знайти якомога більше помилок у додатку. Помилки об'єднуються у звіт про помилку, який передається розробникам для перегляду та виправлення.

Додаток не можна протестувати виключно за допомогою автоматизації, тому ручне тестування відіграє життєво важливу роль у тестуванні програмного забезпечення. Це вимагає певного мислення; терпіння, креативність та відкритість серед них.

Тим не менш слідування документації, підтримка її в актуальному та живому стані, орієнтування розробки саме на документацію є дуже корисними практиками. Забезпеченню цих умов на проекті призвана служити методологія розробки, керованої тестуванням, або інакше BDD (Behaviour Driven Development).

Дана методологія є відносно новою і дуже прогресивною, але через деякі складності у впровадженні її на проекті, досі не завоювала сильної довіри та розповсюдженості в IT-проектах.

Тож метою даної роботи є розробка програмного забезпечення для ефективного впровадження BDD методології на проекті, в тому числі – в тестуванні програмного продукту.

Предметом дослідження є методологія BDD, її основи, принципи, процес впровадження та реалізації.

Наукова новизна отриманих результатів: оптимізовано використання фреймворку Cucumber, що забезпечує е та ефективніше впровадження BDD методології в проекті, а також спрощує написання тестових сценаріїв з синтаксисом Gherkin Notation. А завдяки інтеграції з Selenium Web Driver добре підійде для автоматизації Web UI. Кожна з цих технологій може бути використана самостійно для різних програмних продуктів. В даній роботі розглянуто спільне використання Selenium та Cucumber, що дозволяє підвищити загальну швидкодію процесу тестування web-сервісів та розширити функціональні властивості цього процесу.

Практичною цінністю даної роботи є розроблене програмне та алгоритмічне забезпечення, що ефективно реалізує методологію поведінкового тестування та забезпечить проведення тестів у максимальній наближеності до тестових сценаріїв.

1. АНАЛІТИЧНИЙ ОГЛЯД ОБЛАСТІ ДОСЛІДЖЕННЯ

1.1. Аналіз предметної області

1.1.1. Забезпечення якості програмних продуктів

Забезпечення якості (QA) - це акт або процес підтвердження того, що вимоги фірми щодо якості виконуються. Управління якістю продукції передбачає планування, виконання та моніторинг діяльності. Забезпечення якості є аспектом моніторингу цієї дисципліни [2].

Забезпечення якості охоплює процеси та процедури, які систематично контролюють різні аспекти послуги чи об'єкта. Завдяки аудиту та іншим формам оцінки зусилля із забезпечення якості виявляють та виправляють проблеми або відхилення, що виходять за рамки встановлених стандартів або вимог.

Іншими словами, забезпечення якості забезпечує високий рівень якості під час розробки продуктів чи послуг.

Термін "забезпечення якості" іноді використовується взаємозамінно з "контролем якості", іншою аспектом процесу управління. Однак контроль якості стосується фактичного виконання будь-яких вимог до якості, які були встановлені. Забезпечення якості - це перевірка методів контролю якості, щоб переконатися, що вони працюють, як заплановано.

Основна різниця між забезпеченням якості та контролем якості полягає в тому, що заходи із забезпечення якості проводяться під час розробки програмного забезпечення. Діяльність з контролю якості виконується після розробки програмного забезпечення.

Більшість підприємств використовують якісь форми забезпечення якості виробництва, починаючи від виробників споживчих упакованих товарів і закінчуючи компаніями, що розробляють програмне забезпечення. Деякі компанії можуть навіть створити відділ забезпечення якості зі співробітниками

Методи забезпечення якості зосереджені на створенні хороших процесів виробництва продукції з уже вбудованою якістю, а не на проходженні

неспостережуваного виробничого процесу та спробі "перевірити якість" вже готової продукції.

Концепції контролю якості можна простежити, принаймні, до Середньовіччя та зростання гільдій. Майстер міг отримати доступ до мережі зв’язків з іншими майстрами та постачальниками, вступивши до цехової організації. Тоді він міг би отримати вигоду з репутації гільдії, заснованої на стандартах якості продукції, виробленої її членами.

Сучасні підходи до забезпечення якості можуть відрізнятися залежно від галузі. Наприклад, лікарня може застосувати методи контролю якості для покращення якості медичного обслуговування. Це може включати визначення якості шляхом розуміння основних видів діяльності з ефективного догляду за пацієнтами, встановлення контрольних показників якості та вимірювання якості за допомогою опитувань, аудитів та нагляду.

Інший бізнес, такий як виробник харчових добавок, мав би зовсім інші методи забезпечення якості, оскільки параметри його продукту абсолютно відрізняються від лікарні. Наприклад, компанія, що виробляє добавки, може керувати своїми процесами на основі рекомендацій щодо активних фармацевтичних інгредієнтів.

Міжнародна організація зі стандартизації (ISO) була заснована в 1947 році з метою забезпечення якості через національні кордони.1 ISO складається зі стандартних організацій, які представляють понад 160 країн. Він підтримує ефективну систему забезпечення якості для виробничої та сфери послуг [5].

Одним із продуктів ISO є набір стандартів, які стали відомими як сімейство ISO 9000. Критерії, деталізовані в цих системах управління, покликані допомогти організаціям задовольнити законодавчі та нормативні вимоги щодо якості продукції та потреб споживачів.

Міжнародна організація зі стандартизації (ISO) опублікувала великий огляд свого стандарту ISO 9001. Цей огляд включає ряд важливих змін як для організацій, які вже мають цю сертифікацію, так і для тих, хто хоче розробити та впровадити систему управління якістю (СМК).

1.1.2. Методологія Behaviour Driven Development

Керована поведінкою розробка (або BDD) - це гнучка техніка розробки програмного забезпечення, яка заохочує співпрацю між розробниками, службою контролю та нетехнічними або діловими учасниками програмного проекту. Спочатку він був названий у 2003 році Ден Нортом як відповідь на тестовий розвиток (TDD), включаючи прийнятний тест або практику розроблених клієнтом тестових методів, як це виявляється в екстремальному програмуванні. Він розвивався протягом останніх кількох років.

Щодо “Agile специфікацій, BDD та тестування обміну” у листопаді 2009 року в Лондоні, Ден Норт дав таке визначення BDD:

BDD - це методологія другого покоління, заснована на витягуванні, багатостороння зацікавлена сторона, багатомасштабна, високоавтоматизована, гнучка методологія. Він описує цикл взаємодій з чітко визначеними результатами, що призводить до доставки робочого, перевіреного програмного забезпечення, яке має значення.

BDD фокусується на отриманні чіткого розуміння бажаної поведінки програмного забезпечення шляхом обговорення із зацікавленими сторонами. Він розширює TDD, пишучи тестові кейси природною мовою, яку можуть читати непрограмісти. Розроблені поведінкою розробники використовують рідну мову в поєднанні з повсюдною мовою дизайну, керованого доменом, щоб описати мету та переваги свого коду. Це дозволяє розробникам зосередитись на тому, чому слід створювати код, а не на технічних деталях, і мінімізує переклад між технічною мовою, якою написаний код, і доменною мовою, якою говорять бізнес, користувачі, зацікавлені сторони, управління проектами тощо.

BDD визначається вартістю бізнесу; тобто вигода для бізнесу, яка накопичується після того, як заявка запущена у виробництво. Єдиний спосіб реалізації цієї переваги - це користувальницький інтерфейс (и) програми, зазвичай (але не завжди) графічний інтерфейс.

Таким же чином кожен фрагмент коду, починаючи з інтерфейсу користувача, можна вважати зацікавленою стороною інших модулів коду, які він використовує. Кожен елемент коду забезпечує певний аспект поведінки, який у співпраці з іншими елементами забезпечує поведінку програми.

Першим виробничим кодом, який розробники BDD впроваджують, є інтерфейс користувача. Тоді розробники можуть скористатися швидкими відгуками про те, чи виглядає та поводиться інтерфейс належним чином. Завдяки коду та використовуючи принципи хорошого дизайну та рефакторингу, розробники виявляють співавторів інтерфейсу користувача та всіх наступних одиниць коду.

Наприклад, вимога до програми може бути наступною: "Користувач має мати змогу залогінитись на сайті". У BDD розробник або інженер з контролю якості може пояснити вимоги, розбивши це на конкретні приклади. Мова наведених нижче прикладів називається Gherkin і використовується для опису поведінки.

```
Scenario: User should be able to log in
  Given a user previously created and account
    and he enter his credentials into the login form
   When he click “Log In” button
   then he should be redirected to the Wellcome page
```

Цей сценарій є прикладом, призначений для ілюстрації конкретного аспекту поведінки програми.

Обговорюючи сценарії, учасники сумніваються, чи завжди описані результати є наслідком тих подій, що відбуваються в даному контексті. Це може допомогти розкрити подальші сценарії, що пояснюють вимоги. Наприклад, експерт домену, який зауважує, що відшкодовані товари не завжди повертаються на склад, може змінити вимоги на „Повернені або замінені предмети повинні бути повернуті на склад, якщо вони не мають несправності.

Це, у свою чергу, допомагає учасникам визначити обсяг вимог, що призводить до кращих оцінок того, як довго триватимуть ці вимоги.

Слова "Дано", "Коли" та "Тоді" часто використовуються, щоб допомогти вигнати сценарії, але не є обов'язковими.

Ці сценарії також можна автоматизувати, якщо існує відповідний інструмент, що дозволяє автоматизувати на рівні інтерфейсу користувача. Якщо такого інструменту не існує, можливо, його можна буде автоматизувати на наступному рівні, тобто: якщо був використаний шаблон дизайну MVC, рівень Контролера.

Ті самі принципи прикладів, що використовують контексти, події та результати, використовуються для керування розробкою на рівні абстракції програміста, на відміну від рівня бізнесу. Наприклад, наступні приклади описують різні аспекти поведінки списку в мові програмування:

```
Scenario: New string variable is empty
  Given create a new variable
   then the variable should be empty.

Scenario: Initiated variable is not empty.
  Given create a new variable
   when I initiate it with some value
   then the variable should not be empty.
```

Обидва ці приклади потрібні для опису логічної природи змінної на Java та отримання вигоди від цієї природи. Ці приклади, як правило, автоматизовані з використанням фреймворків TDD. У BDD ці приклади часто інкапсулюються в один метод, причому назва методу є повним описом поведінки. Обидва приклади потрібні для того, щоб код був цінним, і інкапсуляція їх таким чином полегшує опитування, видалення або зміну поведінки.

1.2. Аналіз стану розв'язання проблеми

Є багато переваг у методології BDD, але є також багато проблем, з якими можна зіткнутися під час впровадження. При детальному вивченні даного підходу.

Behaviour Driven Development (BDD) - це впровадження програми шляхом опису її поведінки з точки зору зацікавлених сторін. Цей підхід заснований на постійній розмові та співпраці між ключовими зацікавленими сторонами та зворотньому зв'язку із клієнтом. Ця методологія має багато переваг, але є також багато проблем, з якими ви можете зіткнутися під час впровадження.

Нижче наведений список типових проблем, з якими стикаються різні пректи при впровадженні BDD:

1. Недостатній бюджет

Перехід на BDD - як і будь-яка інша зміна життєвого циклу розробки програмного забезпечення - вимагає не тільки нових процесів розробки та тестування, але й грошових вкладень. Команді потрібно навчити ділових людей та інженерів, як краще спілкуватися, розробляти доменний словниковий запас та всюдисущу мову, запускати чи налагоджувати автоматизацію тестування та впроваджувати тестовий розвиток, і на всі ці нові ініціативи потрібні гроші.

2. Розмір команди

BDD вимагає повної специфікації перед початком спринту. Зазвичай команда спільно обговорює вимоги та переганяє їх у виконувані сценарії (тести), використовуючи мову домену для кожної конкретної функції, перед початком ітерації розробки спринту. Вся ця попередня робота може бути проблемою для невеликих команд та швидких проектів, і додаткові зусилля можуть уповільнити їх, не окупившись.

3. Поганий зворотній зв’язок

Залишатися на зв'язку з клієнтами та експертами доменів може не становити проблем для деяких організацій, але для багатьох ця вимога постійного контакту із зовнішніми людьми може спричинити затримки. У випадках, коли зворотній зв'язок є обов'язковим для початку нової історії користувача, щоб поліпшити поточну поведінку функціоналу, якщо відповідний експерт домену недоступний на той час або ви чекаєте на аналіз відгуків клієнтів, розробку можна сповільнився, примусово перефокусувався або навіть зупинився.

4. Відсутність контролю якості з навичками кодування

BDD - це методологія з високою автоматизацією, тому навички кодування для інженерів з контролю якості є обов'язковими. Якщо на прокеті вже встановлена автоматизація тестів, це не повинно бути проблемою (особливо якщо практикується розробкп на основі тестів або прийняття на основі тестів), але багато організацій намагаються перейти на BDD, оскільки у них немає ресурсів для автоматизації інженери або тому, що їх інженери з контролю якості знають лише, як проводити ручне тестування. Якщо персоналу контролю якості потрібні навички кодування, перед тим, як ви спробуєте BDD, то можна організувати можливості для навчання та перетворити свої ручні зусилля з контролю якості на процес автоматизації тестування. Інший варіант - найняти декількох старших інженерів з автоматизації тестів, які будуть відповідати за побудову та підтримку системи тестування та написання низькорівневих методів (цеглинок), які інші інженери можуть використовувати для написання автоматизованих тестів на більш високому рівні.

5. Мінімальна ділова логіка

Особливо частими є невдачі на проектах, де було прийнято рішення запустити BDD, але проекти були лише інтерфейсом користувача, не маючи складної бізнес-логіки. Слід мати на увазі, що BDD застосовується до проектів із достатньою організаційною та доменною складністю, що може спричинити проблеми у розумінні або передачі вимог з точки зору бізнес-домену. BDD не дає багато переваг лише для інтерфейсу, суто технічні проблеми, де ключова складність полягає не в розумінні та спілкуванні.

6. Забагато зустрічей

Потрібно призначити додаткові семінари та зустрічі, щоб полегшити співпрацю навколо історій та особливостей. Але не варто запрошувтаи всіх на кожну окрему зустріч - краще запитати лише одного представника або експерта від кожного підрозділу (власника продукту, бізнес-аналітика, QA, розробника тощо). Щоб мінімізувати час, який витрачається на зустрічі, слід підготувати керівництво, надіславши команді список матеріалів, які слід переглянути заздалегідь, написавши кілька основних прикладів або сценаріїв для початку відкриття та

переконавшись, що кожна функція має достатню логіку (і зміна тексту, якщо ні). Інженери повинні підготуватися, переглянувши список історій для наради та придумавши список уточнюючих питань.

7. Невдале планування та нереальні очікування

Виходячи досвіду, більшість менеджерів розглядають BDD як срібну кулю для всіх питань і розраховують негайно отримати результати. Але як і будь-який новий процес, пристосування до нього вимагає часу. BDD також вимагає змін на всіх стадіях життєвого циклу розробки програмного забезпечення: створення вимог, аналіз вимог, розробка та тестування. Щоб скористатися реальними перевагами BDD, спочатку потрібно створити достатню основу з повсюдною лексикою, сценаріями та тестовим покриттям. Якщо не поділитися цим баченням із зацікавленими сторонами заздалегідь, можуть виникнути проблеми.

8. Сильно розподілена команда

Швидкі методології працюють найкраще, коли існує швидке, часте спілкування між усіма зацікавленими сторонами на стендах та інших зустрічах. Деякі агілісти вважають, що BDD працює лише тоді, коли команди розміщені на 100 відсотків, але, виходячи з досвіду, можна досягти міцної співпраці з розподіленими командами - якщо ми говоримо про один або два часові пояси. Це починає ставати складнішим, коли членів команди розділяють більше ніж на пару годин, особливо коли QA та розробники знаходяться в різних часових поясах. Важко вибрати час, коли кожен може бути ефективним на зустрічі, навіть не беручи до уваги також можливі мовні чи культурні бар'єри. BDD - це все про спілкування, співпрацю та зворотній зв'язок, тому, якщо один із цих аспектів зазнає невдачі, зазнає невдачі вся концепція.

9. Неадекватне спілкування

Не можна не наголосити на цьому: комунікація перш за все і часто життєво необхідна до та під час впровадження BDD. Якщо у культурі команди ділові люди та інженери взагалі або недостатньо часто спілкуються між собою, перехід на BDD буде важким. Важко виправити ці звички, оскільки спротив змінам є частиною людської природи. Крім того, інженери, як правило, дивляться на світ інакше, ніж

неінженери. Зазвичай вони надзвичайно логічні, прагматичні та прямі. Їм доведеться попрацювати над тим, щоб часто спілкуватися з діловою стороною, виступати на зустрічах та виходити назустріч, щоб повідомити про прогрес та перешкоди на шляху.

10. Розглядати якість як виключно відповідальність тестувальників

Деякі організації вважають, що BDD пише лише автотести синтаксису Gherkin, і все - просто написав деякі автоматизовані тести без майстер-класів чи сценаріїв, заздалегідь перевірених у бізнесі. Є також випадки, коли керівництво вважає, що розробники взагалі не повинні писати тести, оскільки це має робити виключно QA, що в цілому є неправильним, але особливо стосовно BDD.

BDD корисний, коли тестування не відповідає лише інженерам з контролю якості. Мова для конкретного домену корисна, коли співробітники бізнесу та інженери з контролю якості та розробники співпрацюють та сприяють визначенню, створенню та підтримці функціональних тестів. Якщо ніхто, крім тестувальників, не читає ці сценарії, з точки зору тестування, BDD просто виглядає як додаткові накладні витрати.

11. Невідповідне форматування сценаріїв BDD

Частиною хорошого спілкування є забезпечення того, щоб кожен міг зрозуміти сценарії BDD, тому те, як ви їх пишете, має значення. Варто на увазі ці три заборони.

– Не використовуйте елементи керування інтерфейсом у своїх кроках. Дотримуйтесь бізнес-логіки вашого додатка, замість того, щоб концентруватися лише на користувальницькому інтерфейсі.

– Не змішуйте більше двох доменів в одному сценарії. Якщо ваш сценарій описує функціональну поведінку програми, він не включає перевірку продуктивності та візуальної перевірки. Деякі люди намагаються охопити все, перевіряючи все на кожному екрані програми (включаючи безпеку, продуктивність, текст та стиль елементів) в одному функціональному автоматизованому сценарії тестування. Хороший сценарій BDD повинен бути конкретним.

– Не використовуйте занадто багато кроків в одному сценарії. Хороший сценарій BDD повинен бути коротким, не більше п'ятнадцяти кроків і в ідеалі не більше восьми. В іншому випадку буде важко зрозуміти, для чого призначений сценарій, його буде важко підтримувати, і тест може провалитися, перш ніж досягти функціональності, яку він мав намір перевірити, через деякі проблеми на попередніх кроках.

BDD - це потужний підхід, і в правильних руках він може заощадити проектові багато грошей і зробити клієнтів щасливими. Однак BDD може бути важко прийняти. Це вимагає змін у всіх процесах, тому зовнішня тренерка з досвідом роботи з BDD може бути гарною ідеєю за підтримки керівництва

Головне, про що слід пам’ятати, - це реалістичні очікування щодо прийняття BDD. Не варто чекати, що це буде легко, і що він негайно вирішить усі проблеми.

1.3. Виявлення протиріч відомих теоретичних або експериментальних результатів

Нещодавно було опубліковано кілька статей Миколи Адволодкіна (SDTimes та SauceLabs)[6], які використовують підміну аргументу (так званий «аргумент-опудало») для критики поведінкового розвитку (BDD). BDD не є автоматизацією тестів - це спільний аналіз вимог у поєднанні з тестовою розробкою (TDD), яка, незважаючи на назву, теж не є тестуванням.

То що ж таке BDD?

TDD та BDD - це методи проектування та розробки програмного забезпечення. Вони проводять тести як побічний продукт, але вони не проводять тестування. Це поняття незрозуміле для переважної більшості людей, які заявляють, що роблять BDD. Якщо ви пишете свої тести після того, як написали код, ви не робите BDD, незалежно від того, який інструмент ви використовуєте.

Виходячи з цього, кращою назвою для цих статей буде "BDD - це не тестування", або "Робити BDD - шкідливо" чи щось подібне. Коли BDD робиться добре, це не заважає автоматизації тестів - це покращує її.

Себ Роуз та Гаспар Надь популяризували BDD як 3 практики [7] - дослідження, формулювання та автоматизацію, які наведені на рисунку 1.1.

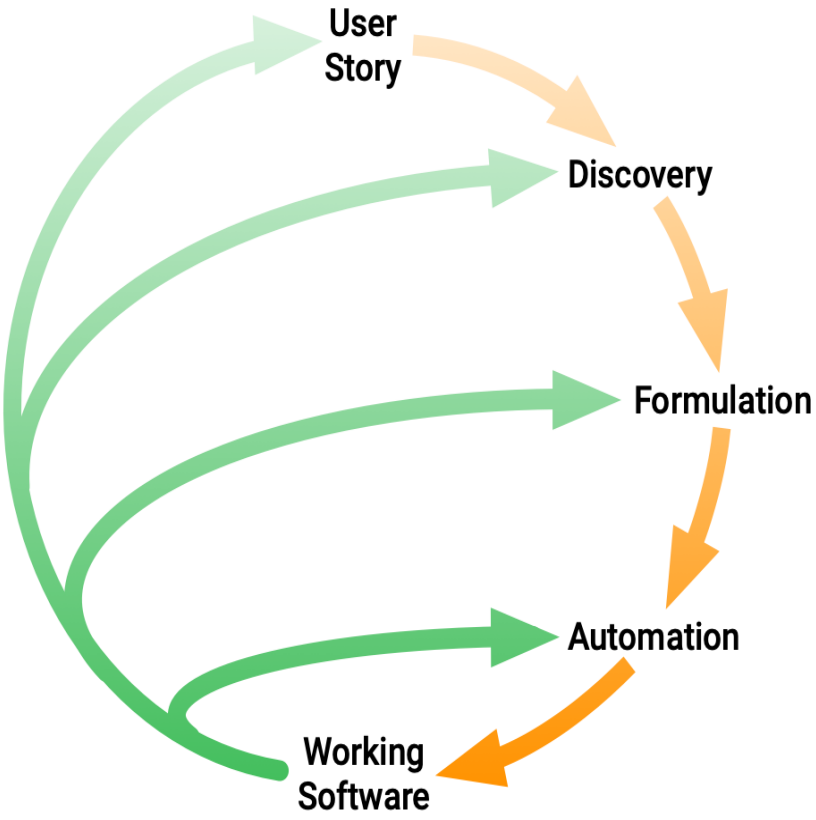

Рисунок 1.1 – Життєвий цикл BDD методології

Ось перелік цих практик:

– Дослідження - Створіть спільне розуміння вимог за допомогою співпраці, яка зазвичай досягається за допомогою структурованої бесіди, зосередженої на правилах та прикладах

– Формулювання - приклади поведінки системи задокументовані за допомогою бізнес-термінології

– Автоматизація - Приклади поведінки автоматизовані, створюючи діючу документацію, яка перевіряє поведінку системи

Проблеми виникають, коли за кожну практику відповідають неправильні ролі (або практики повністю пропускаються). Що часто спостерігається на проектах:

– Дослідження: Зовсім не практикується

– Формулювання: PO / BA пише Gherkin сценарії перед кодуванням та / або тестер пише Gherkin сценарій після того, як розробники імплементували функціонал

– Автоматизація: Тестери автоматизують Gherkin сценарії після того, як розробники імплементували певний функціонал.

Це не BDD. Це просто погана комунікація, повільний зворотний зв’язок та традиційна автоматизація тестів.

Ось що, натомість, є BDD:

– Дослідження: Робиться спільно різноманітною групою (3 особи). PO / BA, розробник, тестер, UX, Ops.

– Формулювання: Зроблено розробниками, за бажанням тестери / UX. PO / BA схвалює це.

– Автоматизація: Зроблено розробниками до впровадження виробничого коду. Вони єдині, хто може це зробити, тому що використовують його для керівництва розробкою.

Нижче розглянуті чотири твердження, які Микола робить у статті SDTimes:

– Не та людина бере BDD під контроль - це часто буває. Однак це не критика BDD, а критика людей, що неправильно застосовують її.

– Інструменти BDD створюють додаткові залежності - зроблений добре BDD призводить до сильно розв’язаного коду, що полегшує розроблення менших частин ізольовано. Знову ж таки, це критика нерозуміння того, як застосовувати BDD, а не самого BDD.

– Інструменти BDD тяжко розпаралелюються - це правда, але при сильно розв’язаній системі потреба в розпаралелізації мінімальна. За допомогою сильно

розв’язаної кодової бази можна усунути більшість операцій вводу-виводу, усуваючи необхідність паралелізації для прискорення набору тестів.

– Тести стають не більш а менш читабельними - Знову ж таки, не критика BDD, а критика погано написаних тестів. BDD не говорить вам писати нечитабельні тести, але це те, з що виходить, коли вони пишуться тестерами, які починають працювати з BDD з мисленням як при написанні звичайних UI сценаріїв.

BDD – складний підхід. Він вимагає співпраці. Для цього потрібні розробники, які розуміються на технологіях TDD, що вимагає навчання та досвіду. Для цього потрібні тестери, які розуміють, що тестування не має нічого спільного з BDD. Найкращий спосіб, яким розробники та тестувальники можуть внести свій внесок у процес BDD – це долучитися до специфікації вимог (Discovery) і протистояти бажанням писати орієнтовані на інтерфейс, повільні, нечитабельні тести на Gherkin.

Який висновок можна зробити з дослідження даних матеріалів?

Микола має рацію, що BDD - це не автоматизація тестів. Однак я не згоден з ним, коли він каже, що "BDD, як виявилося, робить автоматизацію тестів ще більш складною".

Будь-яка команда, яка співпрацює над відкриттями та формулюванням, отримає переваги спільного розуміння та зменшення об’єму роботи. Якщо вони потім використовують автоматизацію для керування розробкою, вони отримають перевагу читабельної і живої документації, яка забезпечує цінність протягом усього терміну служби продукту.

І, що найважливіше, розв’язана структура, яка виникає внаслідок роботи зовні (все це важливо як для TDD, так і для BDD), насправді може зробити автоматизацію тестів набагато менш складною, надаючи програмне забезпечення, яке вже має вбудовану можливість перевірки.

2. РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

2.1.Пошук актантів та варіантів використання

Визначення суб’єктів (акторів, актантів) є одним із перших кроків у аналізі варіантів використання. Кожен тип зовнішніх сутностей, з якими система повинна взаємодіяти, представлений актором. Наприклад, операційне середовище програмної системи складається з користувачів, пристроїв та програм, з якими система взаємодіє. Вони називаються акторами, що має такі характеристики:

– Актор у моделюванні варіантів використання визначає роль, яку відіграє користувач або будь-яка інша система, яка взаємодіє з предметом.

– Актор моделює тип ролі, яку відіграє сутність, яка взаємодіє з суб'єктом (наприклад, обмінюючись сигналами та даними), але яка є зовнішньою для суб'єкта.

– Актори можуть представляти ролі, які виконують люди-користувачі, зовнішнє обладнання або інші суб'єкти.

– Актори не обов'язково представляють конкретні фізичні сутності, а лише конкретні грані (тобто "ролі") деяких сутностей, які мають значення для специфікації пов'язаних з ними випадків використання.

– Один фізичний екземпляр може виконувати роль кількох різних акторів, а даний актор може грати кілька різних екземплярів.

Типи акторів включають (Рисунок 2.1):

– користувачів

– системи баз даних

– клієнтів та серверів

– хмарні платформи

– пристрої

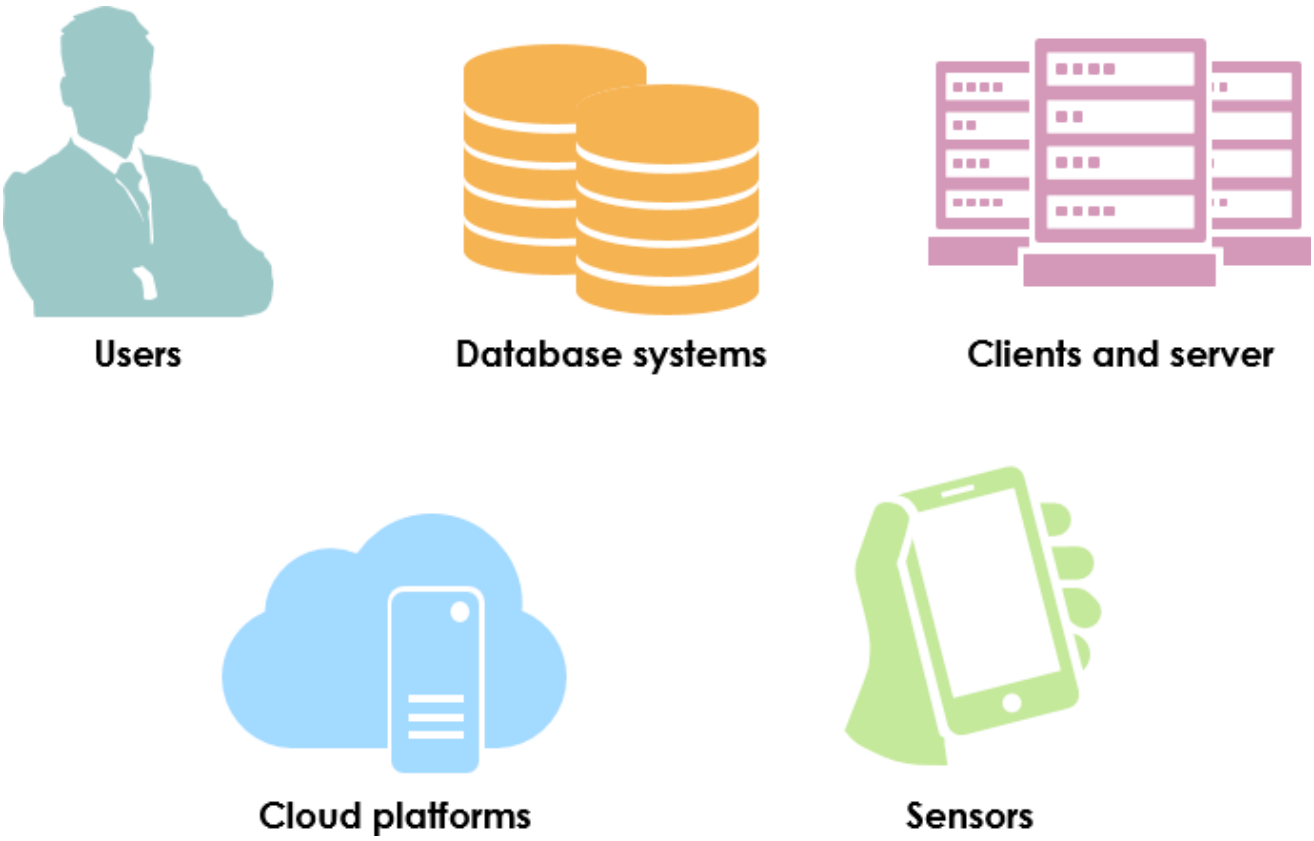

Рисунок 2.1 – Можливі типи акторів в UML

Актор (кінцевий користувач) – це термін, який використовується для людей, які зазвичай використовують продукт часто або регулярно у своїй роботі. Кінцеві користувачі - це «клієнти», які безпосередньо використовують різні продукти та інструменти для досягнення бізнес-цілей своєї організації. Оскільки вони тісно співпрацюють з товаром, кінцеві користувачі розглядаються як одне з найкращих джерел інформації про фактичну експлуатацію товару та вартість, яку він надає.

Персони тісно пов'язані із варіантами використання використання, і вони можуть охоплювати декілька випадків використання. Важливо не мати занадто багато людей, для яких ви намагаєтесь щось побудувати. Тільки тому, що ви вирішили побудувати щось для однієї персони, це не означає, що це також не буде працювати для іншої персони.

Прекрасним прикладом цього є рухома дорожня валіза. Спочатку вона була розроблена для стюардів на коротких рейсах з 1-2 ночами. Вона була

створена для легкого зберігання, прокату через аеропорт та утримання достатньо для зберігання одягу на день або двох. Роль персоналу управителя та їх використання використовували важливу роль у визначенні кінцевого продукту, але чи багато з нас також використовують ці сумки для короткочасних подорожей?

В ході аналізу вимог до програмного забезпечення, було виявлено наступних акторів:

- Ручний тестувальник (Manual QA)
- Автоматизатор тестування (Automation QA)
- Менеджер (Project Manager)

Для цих ролей було визначено наступні варіанти використання, діаграма яких наведена на рисунку

а) Налаштувати проект (Configure Project)
   1) Set the number of parallel threads
   2) Set the number of fail retries
   3) Set browser
   4) Set execution tags

б) Створити визначення тестових кроків (Create Step Definition)
   1) Створити метод на мові Java (Create Method in Java)
   2) Створити шаблон кроку (Create Step Pattern)
   3) Додати визначення кроку до класу ReusableRunner.class, для того щоб даний крок міг використовуватися в написання повторно використовуваних сценаріїв (Add Stepdef to Ruusable Runner class)

в) Написати локатор (Create Locator)
   1) Написати параметризований локатор (Create Parametric Locator)

г) Написати тестовий сценарій на мові Gherkin (Create Gherkin Scenario)

д) Написати повторно використовуваний сценарій (Create Reusable Scenario)

е) Запустити виконання тестів (Run Tests)

ж) Отримати звіт про виконання тестів (View Report)

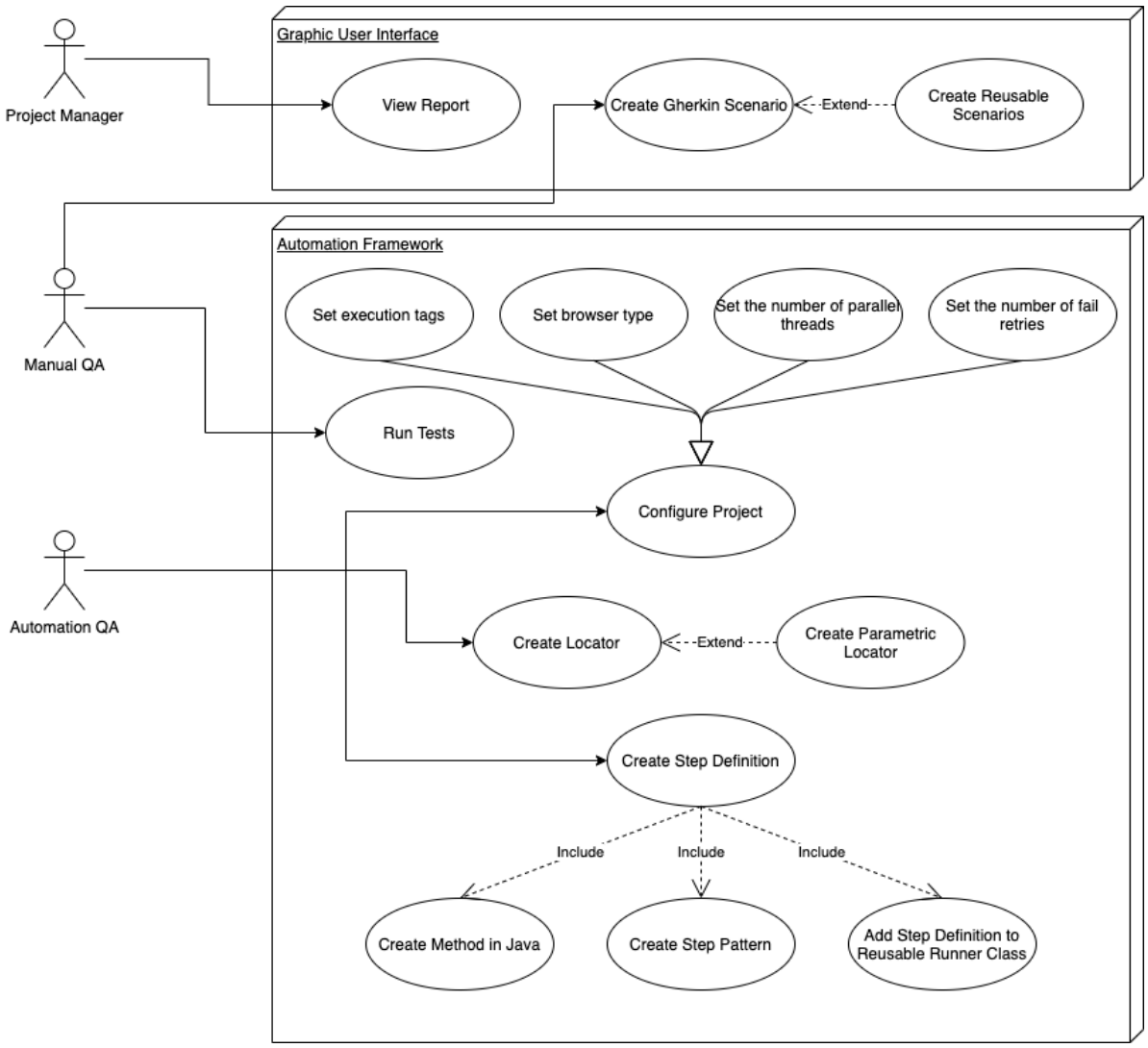

Рисунок 2.2 – Діаграма варіантів використання

Тут діаграма розбита на дві функціональні підсистеми: власне фреймворк та графічний інтерфейс. Як бачимо, в основному з фреймворком взаємодіє Automation QA. А з графічним інтерфейсом – Project Manager.

2.2. Аналіз вимог. Визначення вимог

Метою цього етапу є більш детальне визначення системних входів, процесів, виходів та інтерфейсів. Наприкінці цієї фази процеси системи будуть визначені на

функціональному рівні, тобто функції, які будуть виконуватися, будуть відомі, але не обов’язково як вони будуть виконуватися. Якщо спеціально не обмежено Статутом проекту, Аналіз вимог не повинен враховувати комп'ютерні програми, файли та потоки даних.

Аналіз вимог дозволить виявити та врахувати ризики, пов’язані з тим, як технологія буде інтегрована у стандартні операційні процедури. Аналіз вимог збиратиме функціональні та системні вимоги бізнес-процесу, вимоги користувача та експлуатаційні вимоги (наприклад, при експлуатації, що необхідно для підтримання роботи системи).

Для того, щоб розпочати аналіз вимог, має бути затверджений Статут проекту. Обсяг проекту буде зрозумілий та зазначений у Статуті проекту. Будуть відомі ролі та відповідальність за різні заходи в життєвому циклі розвитку.

В кожного члена команди розробки прогамного продукту є своя роль в процесі аналізу та визначення вимог.

Менеджер проекту: Менеджер проекту відповідає за успіх цього етапу. Основна відповідальність керівника проекту під час аналізу вимог полягає у забезпеченні доступу бізнес-аналітиків до належних: експертів з предметних питань, документації про бізнес-процеси, існуючих технологій та потенційних технологічних рішень, а також поточних та бажаних артефактів. Основною перешкодою для успішного аналізу вимог є відсутність впливу будь-якого з раніше перелічених пунктів.

Бізнес-аналітик - бакалавр повинен спочатку розробити план того, як буде здійснено діяльність з аналізу вимог. Потім BA повинен задокументувати описи бізнес-процесів та зібрати вимоги системи до експертів з предметних питань (МСП) таким чином, щоб забезпечити простежуваність до документів, створених під час попередньої діяльності, та створити основу для подальшої діяльності. Усі визначені вимоги повинні входити в обмеження обсягу проекту та узгоджуватися із заявою про потреби замовника. BA отримає матрицю відстеження вимог, яка стане основою для проектування.

Керівник команди тестування - бере участь у цій діяльності, щоб гарантувати, що вимоги, визначені бізнес аналітиком та прийняті замовником, є вимірюваними та що IT має ресурси для проведення адекватного тестування. Залучення на цьому етапі тестуючого свинця забезпечує належне планування та підготовку до різних етапів тестування, які відбуваються під час SDLC.

Функціональні вимоги: Цей документ детальніше визначає системні входи, процеси, виходи та інтерфейси. Такі деталі можуть бути надані у різних форматах. Зазвичай жодної індивідуальної техніки чи моделі недостатньо, щоб виразити вимоги нетривіальної системи. Різні типи проектів та різні різновиди клієнтів досягнуть успіху, використовуючи різні методи збору та представлення функціональних вимог. Функціональні вимоги повинні містити Матрицю простежуваності вимог, яка буде рівномірно прийнята як основний компонент усіх вимог.

Завершення Аналізу вимог означає презентація задокументованих вимог Замовнику та Проектувальникам.

2.3.Верифікація вимог

Специфікація вимог до програмного забезпечення (SRS) - це детальний опис програмної системи, яка повинна бути розроблена на основі функціональних та нефункціональних вимог. SRS була розроблена на основі угоди між замовником та підрядником. Це може включати ситуації, коли користувач має намір взаємодіяти з програмною системою. Документи специфікації програмного забезпечення, що відповідають всім необхідним вимогам для розробки проекту. Щоб розробити програмну систему, ми повинні чітко розуміти програмну систему. Для цього нам потрібно постійно спілкуватися з клієнтами, щоб зібрати всі вимоги.

Хороший SRS визначає, як програмна система взаємодіє з усіма внутрішніми модулями, апаратним забезпеченням, взаємодіє з іншими програмами, а

користувач взаємодіє з різними реаліями. Менеджери використовують документи специфікації програмного забезпечення вимог до якості (SRS) для створення планів випробувань. Важливо, щоб тестери були очищені з усіма деталями, зазначеними в цьому документі, щоб уникнути помилок у тестових випадках та очікуваних результатів.

Настійно рекомендується перевірити або протестувати документ SRS перед написанням тестових сценаріїв та складанням будь-якого плану тестування. Нижче показано, як перевірити SRS та важливі моменти, про які слід пам’ятати при тестуванні SRS.

Специфікація програмного забезпечення на даному рівні розробки:

а) Вступ

Пропозиція розробити інструмент управління завданнями. Назва інструменту: «Система для написання та менеджменту автоматизованих BDD тестів». Мета продукту - поставити інструмент для створення сценаріїв та виконання тестів на читабельній, предметно-орієнтованій мові.

б) Загальний огляд

Інструмент дозволяє ручним тестувальникам писати читабельні сценарії тестів, виконувати їх та генерувати звіти про виконання.

Для досягнення основних цілей проекту слід виконати наступні кроки:

1) Сформувати команду розробників;
2) Вказати основне середовище, де буде використовуватися програмне забезпечення;
3) Інтернет-зв’язок повинен підтримуватися;
4) Розробити алгоритм генерації звітів про виконання завдань;
5) Дозволити виконувати завдання та тести;
6) Звести зусилля до створення звітів до мінімуму;
7) Придумати стратегію тестування та план тестування.

в) Вимоги

1) Функціональні:
   - Як manual QA Engineer, я хочу мати можливість писати тестові сценарії, використовуючи зручну для читання мову;
   - Як manual QA Engineer, я хочу мати можливість виконувати письмові тести;
   - Як Project Manager я хочу мати можливість бачити автоматичні тестові сценарії в квитках Jira;
   - Як Project Manager, я хочу мати можливість бачити звіт про виконання тесту;
   - Як manual QA Engineer, я хочу мати можливість керувати тегами, що використовуються в системі;
2) Нефункціональні вимоги:
   - Вимоги до продукту:
     - Тест повинен виконуватися принаймні в 5 паралельних потоках потоко-безпечним способом;
     - Члени команди повинні мати можливість писати тестові сценарії читабельною мовою;
     - Для кожного запущеного тестового сценарію слід створити папку Repots;
     - Час очікування FindElement на сторінці повинен бути не менше 3 секунд;
   - Організаційні вимоги:
     - Проектування та розробка системи здійснюються відповідно до розроблених специфікацій;
     - Процес розробки здійснюється з урахуванням усіх частин життєвого циклу розробки;
     - Зазначені фреймворки та мова програмування використовуються від початку до кінця;
   - Зовнішні вимоги:

- Виклики API до інструменту повинні бути надійно зашифровані;
- Для використання системи клієнт повинен прийняти ліцензійну угоду Apache з відкритим кодом;

3) Системні вимоги:
   - В кінці тижня має бути сформовано підсумковий звіт про виконання завдання;
   - Наприкінці спринту слід сформувати підсумковий звіт про виконання завдання;
   - Звіт повинен містити резюме завдання та стан виконання, а також дату виконання (якщо виконано);
   - Якщо працює кілька проектів, звіт повинен розділяти відповідні завдання на відповідний звіт про виконання завдань;
   - Кожен тип звіту має унікальний значок на інтерфейсі користувача;
   - Коли користувач вибирає піктограму, слід отримати відповідний звіт;

2.4. Аналіз ризиків. Керування ризиками

Управління ризиками проекту - це процес виявлення, аналізу та реагування на будь-який ризик, що виникає протягом життєвого циклу проекту, щоб допомогти проекту залишатися на шляху і досягти своєї мети. Управління ризиками не є реактивним; це повинно бути частиною процесу планування, щоб з’ясувати ризик, який може трапитися в проекті, і як контролювати цей ризик, якщо він насправді виникає.

Ризик - це все, що потенційно може вплинути на графік, результативність чи бюджет вашого проекту. Ризики - це потенціали, і в контексті управління проектами, якщо вони стають реаліями, вони потім класифікуються як “проблеми”, які необхідно вирішити. Отже, управління ризиками - це процес виявлення, категоризації, пріоритетності та планування ризиків до того, як вони стануть проблемами.

Управління ризиками може означати різні речі для різних типів проектів. Для великомасштабних проектів стратегії управління ризиками можуть включати детальне планування кожного ризику, щоб забезпечити наявність стратегій пом'якшення, якщо виникають проблеми. Для менших проектів управління ризиками може означати простий, пріоритетний перелік ризиків з високим, середнім та низьким пріоритетом.

У бізнесі управління IT-ризиками передбачає процес виявлення, моніторингу та управління потенційними інформаційними безпеками або технологічними ризиками з метою пом'якшення або мінімізації їх негативного впливу.

Приклади потенційних IT-ризиків включають порушення безпеки, втрату або крадіжку даних, кібератаки, збої в системі та стихійні лиха. Все, що може вплинути на конфіденційність, цілісність та доступність ваших систем та активів, може вважатися IT-ризиком.

Для ефективного управління IT-ризиками виконайте наступні сім кроків у процесі управління ризиками:

- Встановлення контексту,
- Ідентифікація,
- Оцінка,
- Потенційні способи боротьби з ризиком,
- Створення плану,
- Впровадження,
- Перегляд та оцінка плану.

Схематично дані кроки зображені на рисунку 2.3

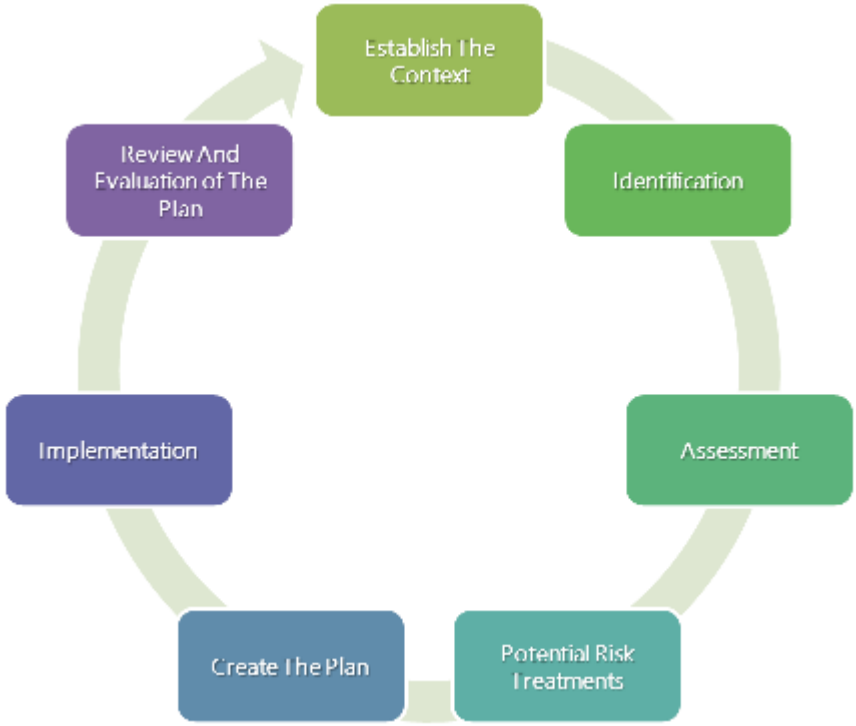

Рисунок 2.3 – Сім кроків процесу управління ризиками

В ході ретельного вивчення можливих ризиків та проведення їх планування, було виявлено наступні ризики

а) Системні ризики:

1) Браузер, який використовується користувачем, може не підтримувати всі функції React;
2) Користувач втрачає підключення до Інтернету;
3) База даних не здатна зберігати всі дані;

б) Командні ризики:

1) Неможливо прийняти кандидата на роботу з необхідними навичками;
2) Член команди хворий;
3) Відсутність можливості проводити тренінги для команди;

в) Організаційні ризики:

1) Відсутність визначеного підходу до розробки;

2) Зміна підходу до розвитку «посередині» життєвого циклу розвитку;
3) Термін дії ліцензії на використовуване програмне забезпечення;
4) Інструментальні ризики:
5) Сервер не може вчасно обробляти всі запити;
6) База даних застрягла в “тупиковій ситуації”, коли користувачі одночасно призначають завдання проекту;

г) Ризики SRS:
1) Зміна вимог вимагає значного вкладення часу та ресурсів;
2) Погано визначені вимоги ведуть до товару, якого споживач не очікує;

д) Ризики планування:
1) Нездорові обмеження в часі;
2) Погано спланований або розподілений бюджет;
3) Погано скомпонована команда розробки;

2.5. Вибір середовища розробки

Вибір правильної IDE Java є критично важливим для успішної розробки додатків. Правильний IDE допомагає розробникам обробляти шлях до класу, створювати файли, створювати аргументи командного рядка тощо. Eclipse, NetBeans та IntelliJ - це добре зарекомендовані, потужні та добре розроблені середовища розробки, які є безкоштовними, з відкритим вихідним кодом або і те і інше. Netbeans має корпоративне видання з додатковими функціями. IntelliJ IDEA пропонує комерційну версію з розширеною функціональністю.

Eclipse існує з 2001 року, коли IBM випустила Eclipse як платформу з відкритим кодом. Керується некомерційною організацією Eclipse Foundation і використовується у відкритих комерційних та комерційних проектах. Починаючи зі скромних коренів, Eclipse став основною платформою, яка також використовується в інших мовах програмування.

NetBeans був розроблений самостійно у другій половині 1990-х. Вона виникла як платформа з відкритим кодом після придбання компанією Sun в 1999 році. Тепер ця IDE може використовуватися для розробки програмного забезпечення для всіх версій Java, починаючи від Java ME і закінчуючи Enterprise Edition. Як і Eclipse, NetBeans має безліч плагінів.

IntelliJ IDEA від JetBrains існує з 2001 року, доступний у комерційному виданні та безкоштовному виданні спільноти з відкритим кодом. JetBrains - заснована компанія, відома своїм плагіном Resharper для Visual Studio, що особливо корисно для розробки C #.

В кожної з представлених IDE є свої переваги.

Найбільшою перевагою Eclipse є те, що він має велику колекцію плагінів, що робить його універсальним та настроюваним. Ця платформа працює у фоновому режимі, складаючи код і відображаючи помилки в міру їх виникнення. IDE організовано в Perspectives, які є візуальними контейнерами, що пропонують набір подань та редакторів.

NetBeans має кілька пакетів: два видання C / C ++ та PHP, випуск Java SE, випуск Java EE та одне видання, що пропонує все необхідне для проекту. Цей IDE також пропонує інструменти та редактори, які можна використовувати для HTML, PHP, XML, JavaScript тощо. Також є підтримка HTML5 та інших веб-технологій.

IntelliJ IDEA підтримує різноманітні мови, включаючи Java, Scala, Groovy, Clojure та інші. Цей IDE має такі функції, як інтелектуальне заповнення коду, аналіз коду та вдосконалений рефакторинг. Комерційна версія Ultimate, орієнтована на сектор підприємств, підтримує SQL, ActionScript, Ruby, Python та PHP. Версія 12 цієї платформи постачається з дизайнером інтерфейсу Android для розробки додатків для Android.

Разом з тим, кожне середовище має свої ключові особливості, які грають велику роль у виборі тієї чи іншої IDE для розробки програмного продукту, враховуючи всі його особливості.

Функції багатозадачності, фільтрації та налагодження в Eclipse є сильними сторонами. Розроблений з урахуванням потреб великих проектів розробки, він

вирішує такі завдання, як аналіз та дизайн, управління продуктами, впровадження, розробка вмісту, тестування та документація.

NetBeans переважає над Eclipse завдяки підтримці баз даних та драйверам для Java DB, MySQL, PostgreSQL та Oracle. Його Провідник баз даних легко створює, модифікує та видаляє таблиці та бази даних у середовищі IDE. У минулому широко розгляданий як тінь Eclipse, NetBeans виступив як грізний конкурент.

IntelliJ IDEA має написані користувачем плагіни. Він пропонує більше 900 плагінів, а також більше 50 плагінів у корпоративній версії. Користувачі можуть надсилати більше плагінів, використовуючи вбудовані в платформу компоненти Swing.

Проаналізувавши сильні сторони та особливості всіх трьох IDE, та співставивши з особливостями та вимогами до розроблюваного програмного продукту, було вирішено обрати IntelliJ IDEA в якості основного середовища розробки, яке буде використовуватися в даному проекті.

2.6.Загальний опис системи

Розроблена система стане основою для створення автоматизованих тестів на мові Gherkin notation та забезпечить усі інструменти для написання, виконання та звітування про завершені сценарії тестування. Система буде в основному орієнтована на інженерів з контролю якості та бізнес-аналітиків, архітекторів систем та інший персонал, який відповідає або зацікавлений у забезпеченні випробувань для розроблених програм.

Цей програмний продукт призначений для тестування веб-служб. Програма дозволяє використовувати позитивні та негативні тести для тестування графічного інтерфейсу користувача, а також для тестування певних функцій. Також після виконання звіту (результату) буде сформовано звіт про інформацію про

проходження тесту, очікуваний результат та фактичний результат. Для проведення тесту використовуючи Selenium WebDriver. Використання цього сучасного додатку для автоматизації дозволить вам надійно отримати інформацію про правильну роботу елементів.

Таким чином, можна використовувати вбудовані функції Selenium WebDriver для імітації дій користувача на сторінці. Програма також перевірить невидимі елементи на сторінці та відобразить інформацію про це. Розробники можуть створювати невидимі елементи для інших цілей або планують додати певні функції в майбутньому. За допомогою цієї програми ви можете перевірити роботу посилань, кнопок і випадаючих списків, текстових полів та перемикачів на сайті. Це основні елементи будь-якої веб-сторінки, саме тому вони були обрані об’єктами дослідження.

Пошук посилань на сторінці здійснюється за допомогою пошуку відповідного тегу в розмітці HTML. Отже, наявні блоки для тестування практично всіх базових елементів web-сторінки.

Набір дій було автоматизовано, за кожен крок відповідає фрагмент коду. Розглянемо їх детально:

```
@Demo @AutomationPratice @Checkout
Feature: Checkout Process
 As a Customer
 I want to be able to go through checkout process
 So that I can purchase a product in the online-store

 @Positive @P1 @ShoppingCart @PLP #First test scenario
 Scenario: Add a Product to Cart
   Given I am on "http://automationpractice.com/index.php" URL
   When I set "Dress" text to the "search_query" "element by
name"
   And I click on the "submit_search" "element by name"
   And I hover over the "Printed Summer Dress"
"AutomationPractice PLP Product Tile"
   And I click on the "Printed Summer Dress"
"AutomationPractice PLP Add to Cart Button"
   Then I should see the "Product successfully added to your
shopping cart" "exact element"
```

Даний фрагмент коду відповідає за тестування одного з найважливіших процесів в електронній комерції - шляху товару крізь процес оформлення замовлення, або інакше кажучи - чекаут. Саме тому цей сценарій позначений тегом з найвищим пріоритетом (P1), тому завжди буде виконаний першочергово. Пошук елементів сторінки здійснюється за допомогою ідентифікаторів, які оточено лапками, усі функції сторінки контактної форми описані в методах, згадно патерну Page Object. При написанні тесту використовуються лише виклик цих методів через синтаксис Gherkin Notation.

Розглянемо приклад негативного сценарію:

```
@Negative @ShoppingCart @PLP @Run #And example of negative
scenario
Scenario: Delete Product from Mini Cart
 Given I am on "http://automationpractice.com/index.php" URL
 When I set "Dress" text to the "search_query" "element by
name"
 And I click on the "submit_search" "element by name"
 And I hover over the "Printed Summer Dress"
"AutomationPractice PLP Product Tile"
 And I click on the "Printed Summer Dress" "AutomationPractice
PLP Add to Cart Button"
 Then I should see the "Product successfully added to your
shopping cart" "exact element"
 And I click on the "Close window" "element by title"
 When I hover over the "Cart" button
 And I click on the "Printed Summer Dress" "AutomationPractice
Mini Cart Delete Button"
 When I hover over the "Cart" button
 Then I shouldn't see the "Printed Summer Dress"
"AutomationPractice Mini Cart Delete Button"
```

В процесі виконання даного тесту перевіряється, факт видалення продукту з корзини, шляхом натискання іконки “x” в випадаючому меню міні-корзини поряд з відповідним продуктом, попередньо доданим до корзини.

Результати проходження тестів заносяться в звіт, що створюється програмою в процесі виконання. Звітність про результати формується засобами фреймворку TestNG і зберігається в директорії framework/reports.

Інструменти фреймворку TestNG дозволяють створити так звану "навігацію" між тестовими блоками, де тестери можуть переглядати успішні та невдалі тести, отримувати детальний звіт кожного тесту та час кожного тесту. Атрибут TestNG

також дозволяє запускати тести в чіткому порядку, що є важливою і важливою перевагою. Оскільки в кожному тестовому блоці наступний тест починається з точки завершення попереднього тесту.

Такі особливості дозволяють побудувати тестові блоки в зручній структурі, виконувати дії послідовно, так само, як робив би те користувач. Емуляція дій користувача є однією з основних задач автоматизації ручного процесу тестування.

На рисунку 2.4 наведено приклад відображення процесу проходження тест плану.

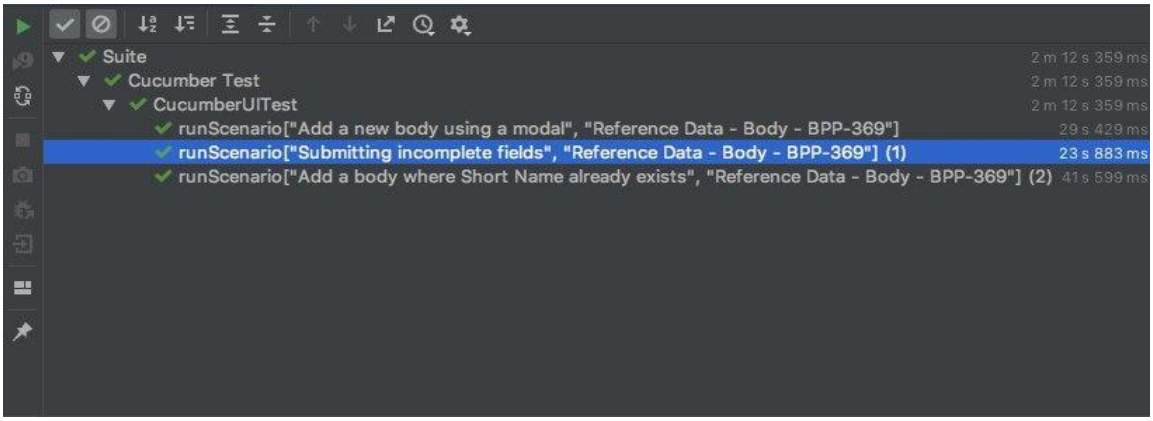

Рисунок 2.4 – Приклад відображення процесу проходження тест плану

На рисунку 2.5 наведено приклад відображення тест репорту.

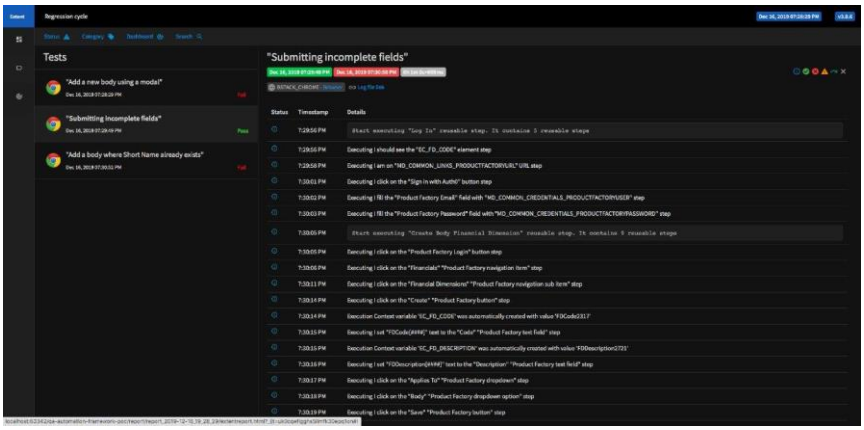

Рисунок 2.5 – Приклад відображення звіту тестування

Засобами Extent Reporting Framework Після виконання тесту буде сформовано детальний звіт про тестування, де можна відстежувати показники тестування та фільтрувати тести за категоріями, авторами, пристроями та ярликами. Невдалий тест супроводжується скріншотом помилки та докладним описом помилки.

2.7. Використані бібліотеки та фреймворки

У цьому розділі наведено список бібліотек та фреймворків, використаних при розробці даного ПЗ, та обґрунтовано та наведено приклади застосування в коді.

TestNG - це загальноприйнята система тестування Java з відкритим кодом, не обмежуючись модульними тестами. Як природний конкурент JUnit, він перевершує свого попередника функціями, які краще підходять для інтеграції та наскрізного тестування, все ще залишаючись такими ж сильними, як JUnit в області модульних тестів.

TestNG підтримує параметризовані тести нестандартно (набагато зручніше, ніж це робить JUnit), полегшує запуск багатопотокових тестів і дозволяє виражати залежності між тестовими методами.

Selenium - це набір засобів автоматизації тестування програмного забезпечення з відкритим кодом, який фактично став продуктом у світі забезпечення якості. Маючи перелік декількох мов програмування, усі основні підтримувані операційні системи та браузери, Selenium в даний час використовується у виробництві таких компаній, як Netflix, Google, HubSpot, Fitbit тощо. Весь набір пропонує цілий ряд рішень для різних проблем та потреб тестування. Далі в статті ми поговоримо про ці інструменти тестування та причини, чому Селен залишався актуальним після десяти років з моменту його створення. Але наразі давайте розберемося, що саме селен зробив так велику справу.

Творіння Джейсона Хаггінса у 2004 році було структурою JavaScript, спрямованою на звільнення його творця від повторюваних ручних тестувань. Продукт, який вперше отримав передбачувану назву JavaScriptTestRunner, міг виконувати тести безпосередньо у браузері, керувати взаємодіями на сторінці та повторно запускати їх без ручного введення. Цей інструмент JavaScript злетів після того, як Хаггінс зрозумів свій потенціал, зробив його відкритим і перейменував у пульт дистанційного керування Selenium. Інноваційна його частина полягала в тому, що жоден інший інструмент до того, як дозволив тестувальникам спілкуватися з браузером на вибраній мовою програмування.

Однак незабаром стало очевидним, що браузери застосовують обмеження безпеки на JavaScript, унеможливлюючи використання всіх можливостей інструменту. У той час Google був завзятим користувачем Selenium, але інженери боролися з обмеженнями. Один з них, Саймон Стюарт, розпочав роботу над продуктом, який буде розмовляти з веб-переглядачами, який він назвав WebDriver. Селен об’єднав зусилля з WebDriver і змінив спосіб тестування програмного забезпечення вже більше десяти років.

Однак проект Selenium займається не лише розробкою драйверів, але й розробкою суміжних продуктів - Selenium Server дозволяє організовувати віддалені запуски браузера.З Selenium Grid ви можете створювати кластери серверів Selenium. Вони порівнянні з переліченими вище інструментами та структурами, оскільки вони також беруть участь у побудові системи тестового запуску. Крім того, існує "реєстратор" під назвою "Selenium IDE", який може записувати операції користувача та генерувати код для виконання записаних операцій за допомогою інтерфейсу WebDriver.

Здебільшого, коли люди говорять про Selenium, вони мають на увазі Selenium WebDriver. Найбільша частина Selenium продукту зосереджена саме на цьому продукті.

Те, як працює автоматизація тестів у WebDriver, часто порівнюють із керуванням таксі. У керуванні таксі та автоматизації випробувань є три учасники: замовник / інженер-випробувач, автомобіль / браузер, водій таксі / WebDriver.

Згідно з цією аналогією, тестер наказує WebDriver взаємодіяти з елементами так само, як клієнт дає вказівки таксисту. Потім WebDriver передає браузеру (машині) команди, які звучать приблизно так: Коли на кнопку можна натиснути, натисніть кнопку. Потім браузер надає WebDriver інформацію про значення та статуси веб-елементів, які згодом надсилаються до сценарію.

Основний код Selenium Framework для розроблюваного ПП сконцентрований у класі BasePage

Cucumber - це програмний інструмент, який підтримує Behavior Driven Testing (BDD). Центральним у підході до Cucumber BDD є його звичайний аналізатор мови під назвою Gherkin. Це дозволяє визначати очікувану поведінку програмного забезпечення логічною мовою, яку клієнти можуть зрозуміти. Таким чином, Cucumber дозволяє виконувати документацію про особливості, написану у тексті, орієнтованому на бізнес. Його часто використовують для тестування іншого програмного забезпечення. Він запускає автоматизовані тести прийняття, написані у стилі BDD [10].

Cucumber спочатку був написаний мовою програмування Ruby, і спочатку використовувався виключно для тестування Ruby як доповнення до рамки BDD RSpec. Зараз Cucumber підтримує різноманітні мови програмування за допомогою різних реалізацій, включаючи Java та JavaScript. Порт Cucumber з відкритим кодом у .Net називається SpecFlow. Наприклад, Cuke4php і Cuke4Lua - це програмні мости, що дозволяють протестувати проекти PHP та Lua відповідно.

Ціллю графічного інтерфейсу даного ПП є надання користувачу можливості швидко та ефективно керувати створенням, редагуванням, сортуванням та пріоритетом запуску автоматизованих BDD тестів.

Перелік основних функцій ПП, які будуть доступні користувачу:

– Можливість створення, редагування та видалення feature файлів з тестовими сценаріями;

– Валідація правильності написання тестових кроків, та автозаповнення при їх введенні;

– Вибір доступних параметрів та локаторів з бази даних;

– Редактор тегів виконання тестів;

Сортування та фільтрація тестових сценаріїв за іменем і тегом.

При розробці користувацького інтерфейсу використовуватимуться JavaFX.

JavaFX - це платформа, заснована на Java, яка використовується для створення додатків з багатими графічними інтерфейсами. З його допомогою можна створювати настільніх додатки, які працюють безпосередньо з операційної системи, Інтернет-додатки (RIA), що працюють у браузерах, та програми на мобільних пристроях. JavaFX призначений для заміни раніше використаної бібліотеки Swing. Платформа JavaFX може конкурувати з Microsoft Silverlight, Adobe Flash та подібними системами.

В даний час технологія JavaFX забезпечує створення потужного графічного інтерфейсу користувача (Graphical User Interface (GUI)), 2D і 3D графіку, забезпечує створення компонентів високоякісної графіки та анімації для різних додатків.

2.8.Концептуальна архітектура системи

2.8.1. Page Object Model

Найбільш популярною і стабільною моделлю тестових фреймворків на сьогодні є Page Object Model (надалі POM).

POM — патерн проектування, який використовують для написання автоматизованих тестів, що дозволяє витягувати вміст з одного елемента HTML і інкапсулювати його у функцію, яку користувачі можуть бачити для доступу до розширених елементів інтерфейсу.

При написанні на веб-сторінці, потрібно звернутися до елементів на цій веб-сторінці, щоб клацнути посилання та визначити, що відображається. Однак, якщо ви пишете тести, які безпосередньо маніпулюють елементами HTML, ваші тести будуть нестабільними до змін в інтерфейсі користувача. Об'єкт сторінки обгортає

HTML-сторінку або фрагмент спеціальним API-додатком, що дозволяє маніпулювати елементами сторінки, не викопуючи HTML-код.

Основне правило для об’єкта сторінки полягає в тому, що він повинен дозволяти клієнту програмного забезпечення робити що-небудь і бачити все, що може людина. Він також повинен забезпечувати інтерфейс, який легко програмувати, і приховує основний віджет у вікні. Отже, для доступу до текстового поля ви повинні мати методи доступу, які приймають і повертають рядок, прапорці повинні використовувати логічні значення, а кнопки повинні бути представлені назвами методів, орієнтованих на дії. Об'єкт сторінки повинен інкапсулювати механіку, необхідну для пошуку та обробки даних у самому елементі керування графічним інтерфейсом. Хорошим емпіричним правилом є уявлення про зміну конкретного елемента управління - в цьому випадку інтерфейс об’єкта сторінки не повинен змінюватися.

Незважаючи на термін "сторінка", ці об'єкти, як правило, не повинні будуватися для кожної сторінки, а для значущих елементів на сторінці. Отже, сторінка, що показує кілька альбомів, матиме об’єкт сторінки списку альбомів, що містить кілька об’єктів сторінки альбому. Напевно, також міститься об’єкт сторінки заголовка та об’єкт сторінки нижнього колонтитула. Тим не менш, частина ієрархії складного інтерфейсу існує лише для того, щоб структурувати інтерфейс - такі складені структури не повинні розкриватися об’єктами сторінки. Основним правилом є моделювання структури сторінки, яка має сенс для користувача програми.

Подібним чином, якщо ви переходите на іншу сторінку, початковий об'єкт сторінки повинен повернути інший об'єкт сторінки для нової сторінки. Загалом операції з об'єктами сторінки повинні повертати основні типи (рядки, дати) або інші об'єкти сторінки.

Існують розбіжності в думках щодо того, чи повинні об’єкти сторінки включати самі твердження, чи просто надавати дані для тестових скриптів, щоб робити ці твердження. Прихильники включення тверджень в об’єкти сторінок кажуть, що це допомагає уникнути дублювання тверджень у тестових скриптах,

полегшує надання кращих повідомлень про помилки та підтримує більше API стилю TellDontAsk. Прихильники об'єктів сторінок без тверджень стверджують, що включення тверджень поєднує обов'язки забезпечення доступу до даних сторінки з логікою твердження та призводить до здутого об'єкта сторінки.

Об'єкти сторінки зазвичай використовуються для тестування, але не повинні робити твердження самі. Їх відповідальність полягає у наданні доступу до стану основної сторінки.

2.8.2. Реалізація Cucumber Framework в архітектурі системи

Центральною ідеєю BDD підходу до розробки програмного забезпечення є написання вимог в файлах з розширенням .feature використовуючи синтаксис Gherkin Notation так, щоб ці вимоги слугували орієнтиром для розробки, а також щоб існувала можливість виконувати сценарії, описані таким чином в цих вимогах, як автоматизовані тести.

Реалізується це на Java шляхом створення двох класів. Класу, в якому зібрані методи, які визначають тестові кроки, та класу, який контролює запуск та виконання BDD сценаріїв та кроків.

Перший, з перерахованих класів, представлений в розроблюваній програмі класом StepDefinitions.class. Все що потрібно зробити для того, щоб Cucumber framework почав розпізнавати методи в цьому класі як визначення тестових кроків - це позначити метод однією з анотацій: @Given, @When, @Then або @And, та передати як параметр в дужках регулярний вираз, по якому Cucumber визначить, який тестовий крок мовою Gherkin в тестових файлах асоціювати з анотованим методом.

Нижче приведений фрагмент коду з класу StepDefinitions, з декількома визначеннями кроків:

```
	@Given("^I am on \"([^\"]*)\" URL$")
	public void i_am_on_url(String url) {
	   Reporter.log("Executing step: I am on '" + url + "' url");
	   String processedUrl =
TestParametersController.checkIfSpecialParameter(url);
	   driver().get(processedUrl);
	   if (!url.equals(processedUrl)) {
	       Reporter.log("<pre>[input test parameter] " + url + "' -> '" +
processedUrl + "' [output value]</pre>");
	   }
	   waitForPageToLoad();
	}


	@When("^I click on the \"([^\"]*)\" (?:button|link|option|element)$")
	public void i_click_on_the_button(String element) {
	   Reporter.log("Executing step: I click on the '" + element + "' element");
	   clickOnElement(initElementLocator(element));
	}


	@When("^I click on the \"([^\"]*)\" (?:button|link|option|element) if
\"([^\"]*)\" \"([^\"]*)\"$")
	public void i_click_on_the_button_if(String element, String
conditionParameter, String condition) {
	   Conditions conditions = new Conditions();
	   if (conditions.checkCondition(condition, conditionParameter)) {
	       Reporter.log("Executing step: I click on the '" + element + "'
element");
	       clickOnElement(initElementLocator(element));
	   } else {
	       Reporter.log("Condition " + conditionParameter + condition + " is not
true, so '" + element + "' element step will not be clicked");
	   }
	}


	@When("^I fill the \"([^\"]*)\" field with \"([^\"]*)\"$")
	public void fill_field(String element, String text) {
	   Reporter.log("Executing step: I fill the  '" + element + "' field with '"
+ text + "'");
	   String processedText =
TestParametersController.checkIfSpecialParameter(text);
	   LogManager.getLogger().info("Setting: " + element + " with value: " +
text);
	   setText(initElementLocator(element), processedText);
	   if (!text.equals(processedText)) {
	       Reporter.log("<pre>[input test parameter] " + text + "' -> '" +
processedText + " [output value]</pre>");
	   }
	}
```

Контроллером виконання BDD тестів слугує клас CucumberUiTest.class, фрагменти коду якого представлені нижче:

```
	@CucumberOptions(
	       features = "src/test/resources/cucumber",
	       glue = {"cucumber.stepdefs"},
	       tags = {"@ProductFactory and not @BlockedByIssue and not @DoNotRun"},
```

```
            plugin = {"pretty"})

      public class CucumberUITest extends BaseUITest {
         private TestNGCucumberRunner testNGCucumberRunner;
         public String scenarioName;

         public CucumberUITest() {
         }

         @BeforeClass(alwaysRun = true)
         public void setUpClass() {
             this.testNGCucumberRunner = new
TestNGCucumberRunner(this.getClass());
         }

         @Test(
                 groups = {"BPP Automation"},
                 dataProvider = "scenarios",
                 retryAnalyzer = RetryAnalyzer.class
         )
         public void runScenario(PickleEventWrapper pickleWrapper,
CucumberFeatureWrapper featureWrapper) throws Throwable {
             scenarioName = pickleWrapper.getPickleEvent().pickle.getName();
             Reporter.node("Executing: " + scenarioName + " scenario",
                     "It contains " +
pickleWrapper.getPickleEvent().pickle.getSteps().size() + " steps");

this.testNGCucumberRunner.runScenario(pickleWrapper.getPickleEvent());
         }

         @DataProvider(parallel = true)
         public Object[][] scenarios() {
             return this.testNGCucumberRunner == null ? new Object[0][0] :
this.testNGCucumberRunner.provideScenarios();
         }

         @AfterClass(alwaysRun = true)
         public void tearDownClass() {
             if (this.testNGCucumberRunner != null) {
                 this.testNGCucumberRunner.finish();
             }
         }
      }
```

Розроблюваний фреймворк взяв найкращі практики з обидвох моделей та адаптував Page Obgect Model під потреби BDD. Таким чином, в програмі є клас BasePage, який містить базову функціональність для роботи з веб-сторінкою, і також дочірні класи для кожного веб-застосунку, які розширюють функціональність BasePage специфічними для окремих застосунків фкнкціями.

Приклад скрипту на мові Gherkin проілюстрований в додатку А.

2.8.3. Архітектура модуля графічного інтерфейсу

Для спрощення взаємодії з основними цільовими користувачами даної системи - нетехнічними спеціалістами, а також для покращення ефективності роботи з основними компонентами BDD системи, було вирішено розробити модуль графічного інтерфейсу. Розробка велася на платформі Java FX.

Графічний інтерфейс має забезпечувати наступний функціонал:

– огляд існуючих тестових файлів;

– редагування раніше створених тестових файлів;

– створення нових тестових файлів;

– підказки для написання тестових кроків

– підказки для параметрів, які можуть застосовуватися до певних тестових кроків

– визначення неправильно написаних тестових кроків

– визначення неправильно написаних параметрів тестових кроків

– підсвітка синтаксису Gherkin notation

– можливість створювати та редагувати локатори елементів

На рисунку 2.6 зображено вигляд головного вікна програми

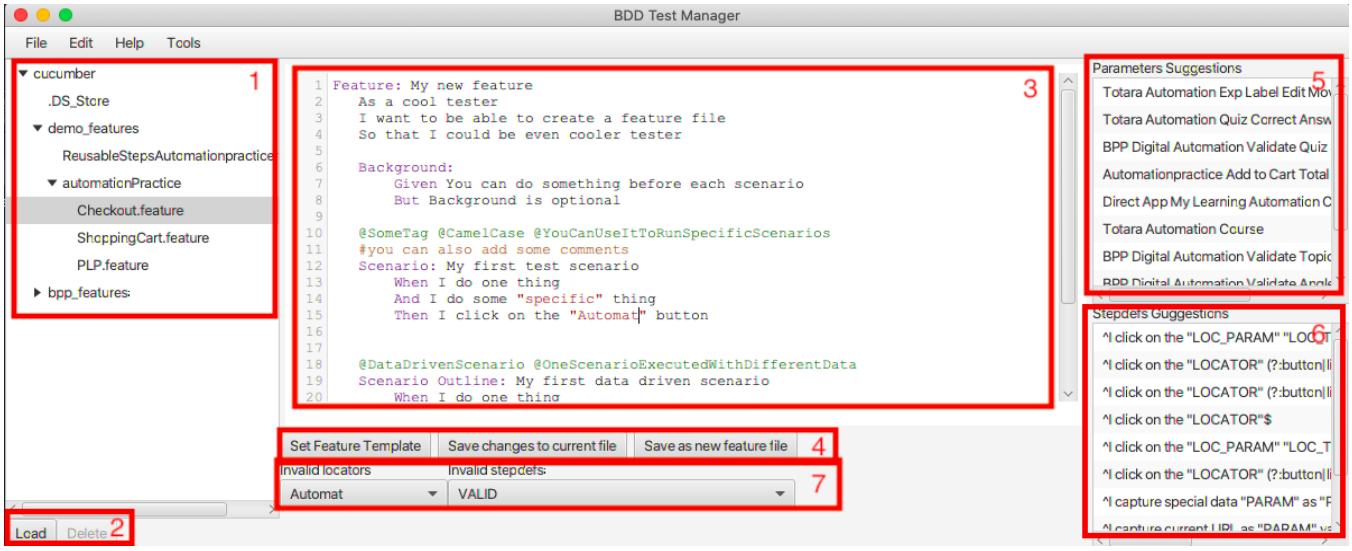

Рисунок 2.6 – вигляд головного вікна програми

Елементи головного вікна програми:

1. Меню навігації по тестових файлах
2. Кнопки для завантаження/видалення вибраного тестового файлу
3. Область редагування тестового файлу
4. Кнопки збереження змін / створення нового тестового файлу
5. Вікно з підказками для можливих тестових кроків
6. Вікно з підказками для можливих параметрів тестових кроків
7. Випадаючі списки з неправильно написаними тестовими кроками та їх параметрами

При відкритті програми є 2 шляхи щоб почати роботу з нею:

– вибрати файл з меню справа і натиснути кнопку Load;

– натиснути кнопку Set Feature Template. В такому випадку в області редагування з’явиться зразок тестового сценарію, орієнтуючись на який можна починати писати свій сценарій

При натисканні кнопки Save as a new file з’являється діалогове вікно, яке пропонує вказати назву файла та шлях до нього. Вигляд діалогового вікна для збереження нового файлу показаний на рисунку 2.7.

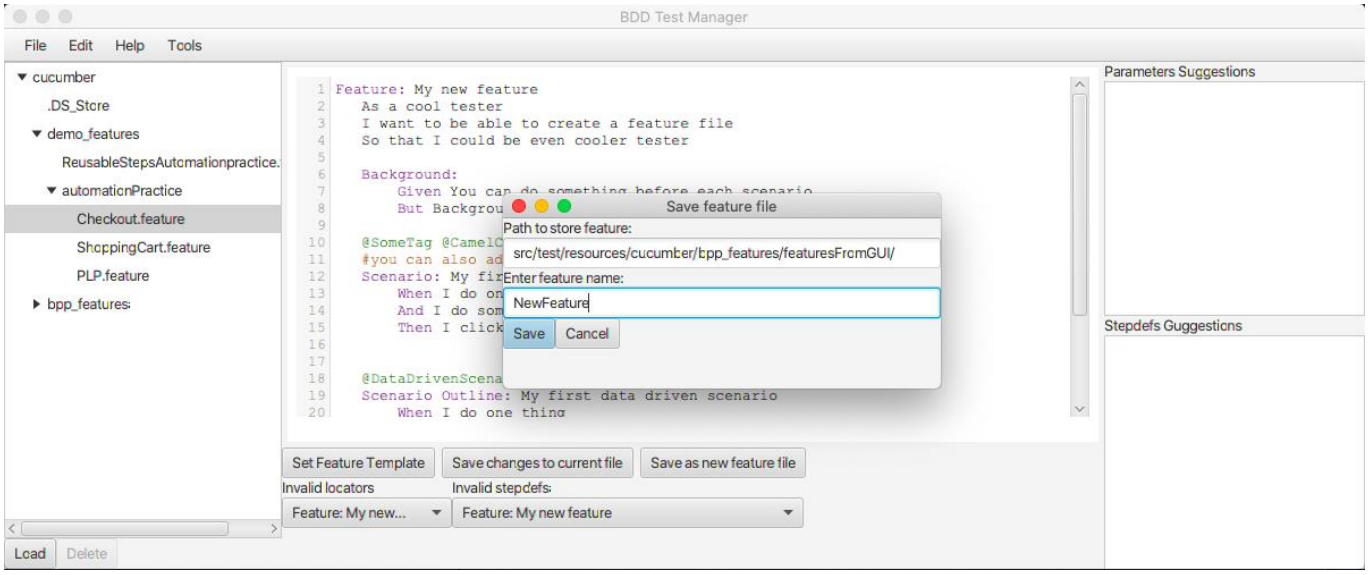

Рисунок 2.7 – Вигляд вікна збереження нового файлу

Для додавання нових локаторів елементів передбачено спеціальне вікно для керування локаторами, яке можна активувати вибравши з меню Tools опцію Locators Manager… . Вигляд даного вікна зображено на рисунку 2.8.

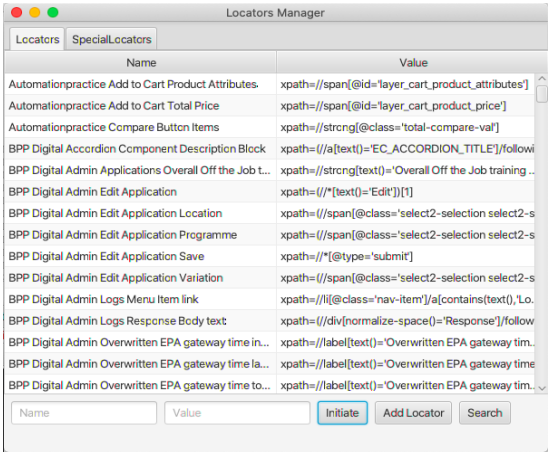

Рисунок 2.8 – Вигляд вікна управління локаторами

За допомогою даного вікна можна зручно здійснювати пошук по доступним локаторам, а також додавати нові локатори натисканням кнопки Add Locator, попередньо заповнивши поля Name і Value.

З точки зору структури проекту, даний модуль є окремим підпроектом (gui), поруч з основним кодом програми (framework), об’єднаним у батьківському проекті (JavaTestFramework). Як це виглядає в IDE можна побачити на рисунку 2.9.

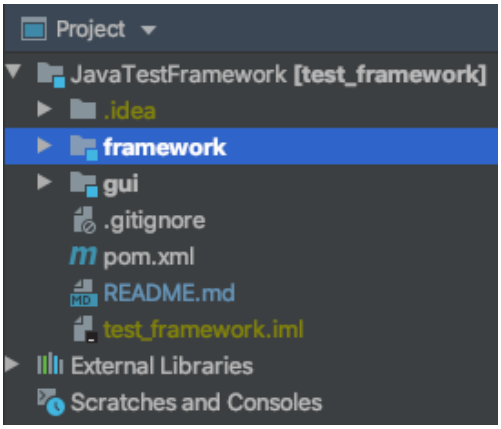

Рисунок 2.9 – Вигляд проекту в IntelliJ IDEA

Розроблений модуль запускається з під IntelliJ IDEA шляхом виконання класу CodeEditorExample.class

2.9.Прийняті архітектурні рішення

В ході розробки даного фреймворку було використано декілька популярних патернів проектування, в тому числі: Simple Factory, Singleton.

Simple Factory

Основне застосування в розроблюваному програмному продукті породжуючий патерн Simple Factory знайшов у класі DriverProvider, задачею якого є створення об’єкту певного класу веб-браузера (ChromeDriver, GeckoDriver, IEDriver та інших) залежно від потреби. Це і є основна задача, яку покликаний реалізувати патерн Simple Factory, і його суть [14].

В об'єктно-орієнтованому програмуванні (ООП), фабрика - це об'єкт для створення інших об'єктів. Формально фабрика - це функція або метод, який повертає об'єкти мінливого прототипу або класу з деякого виклику методу, який вважається «новим».

Приклад використання патерну Simple Factory в розроблюваному програмному продукті наведений нижче:

```
public static WebDriver getDriver() throws MalformedURLException {

    if (instance.get() == null)
        if (getCurrentBrowserName().equalsIgnoreCase(BrowserType.FIREFOX)) {
            instance.set(getFirefox());
        } else if (getCurrentBrowserName().equalsIgnoreCase(BrowserType.CHROME)) {
            instance.set(getChrome());
        }
    return instance.get();

}

static public FirefoxDriver getFirefox() {

    System.setProperty("webdriver.gecko.driver", FIREFOX_PATH);

    try {
```

```
        FirefoxProfile profile = new FirefoxProfile();
        profile.setPreference("browser.download.folderList", 2);
        profile.setPreference("browser.helperApps.neverAsk.saveToDisk",
                "image/jpeg, application/pdf, application/octet-stream,
application/vnd.openxmlformats-officedocument.spreadsheetml.sheet");
        profile.setPreference("pdfjs.disabled", true);


        FirefoxOptions options = new FirefoxOptions();
        options.setCapability(FirefoxDriver.PROFILE, profile);

        return new FirefoxDriver(options);
    } catch (Exception e) {
        throw new WebDriverException("Unable to launch the browser", e);
    }
}


static public ChromeDriver getChrome() {

    try {
        File folder = new File("downloads");
        if (folder != null) {
            folder.mkdir();
        }

        System.setProperty("webdriver.chrome.driver", CHROME_PATH);
        ChromeOptions options = new ChromeOptions();
        Map<String, Object> prefs = new HashMap<String, Object>();
        prefs.put("credentials_enable_service", false);
        prefs.put("profile.password_manager_enabled", false)

        HashMap<String, Object> chromePreferences = new HashMap<>();
        chromePreferences.put("profile.password_manager_enabled", "false");
        chromePreferences.put("credentials_enable_service", "false");
        chromePreferences.put("profile.default_content_settings.popups", 0);
        chromePreferences.put("download.default_directory",
folder.getAbsolutePath());

        options.setCapability("chrome.prefs", chromePreferences);

        return new ChromeDriver(options);
    } catch (Exception e) {
        throw new WebDriverException("Unable to launch the browser", e);
    }
}
```

Тут метод getDriver() повертає спеціалізований об’єкт типу WebDriver, в залежності від того, який тип драйвера вказаний в конфігураційному файлі. Конфігураційний файл перевіряється методом getCurrentBrowserName

Singleton

В цьому ж класі, DriverProvider, знайшов своє застосування ще один породжуючий патерн. Оскільки, в ході запуску тестів на тестовому фреймворку має

сенс обмежити кількість відкритих вікон браузера (в контексті програмування – об’єктів класу WebDriver) доцільно буде використати патерн Singleton.

Singleton — це породжувальний патерн проектування, який гарантує, що клас має лише один екземпляр, та надає глобальну точку доступу до нього.

Ще один приклад використання цього патерну в коді програми – це клас BasePage, який надає фреймворку всю функціональність SeleniumWebDriver’a, і об’єкт якого також є сенс тримати під час виконання програми лише в одному екземплярі.

Приклад використання патерну Simple Factory в розроблюваному програмному продукті наведений нижче:

```
public static WebDriver getDriver() throws MalformedURLException {

    if (instance.get() == null)
        if (getCurrentBrowserName().equalsIgnoreCase(BrowserType.FIREFOX)) {
            instance.set(getFirefox());
        } else if (getCurrentBrowserName().equalsIgnoreCase(BrowserType.CHROME)) {
            instance.set(getChrome());
        }    return instance.get();

}
```

2.10. Специфікації для основних класів

Діаграма класів - це структурна діаграма UML, яка показує структуру спроектованої системи на рівні класів та інтерфейсів, показує їх особливості, обмеження та взаємозв'язки - асоціації, узагальнення, залежності тощо.

Загальні типи діаграм класів:

– діаграма моделі домену,

– схема реалізації класів.

Діаграму об'єктів можна розглядати як діаграму класів рівня екземпляра, яка показує специфікації екземплярів класів та інтерфейсів (об'єктів), слотів із специфікаціями значень та посилань (екземплярів асоціації).

Структурні діаграми показують статичну структуру системи, що моделюється. фокусування на елементах системи, незалежно від часу. Статична структура передається, показуючи типи та їх екземпляри в системі. Окрім відображення типів систем та їх екземплярів, структурні діаграми також показують принаймні деякі взаємозв'язки між цими елементами та між ними та потенційно навіть показують їх внутрішню структуру.

Загалом, ці діаграми дозволяють перевіряти дизайн та спілкуватись між проектами між людьми та командами. Наприклад, бізнес-аналітики можуть використовувати діаграми класів або об’єктів для моделювання поточних активів та ресурсів бізнесу, таких як реєстри рахунків, товари чи географічна ієрархія. Архітектори можуть використовувати схеми компонентів та розгортання, щоб перевірити / перевірити правильність їх конструкції. Розробники можуть використовувати схеми класів для проектування та документування кодованих (або незабаром кодованих) класів системи.

UML 2 розглядає структурні діаграми поза класифікацією. Немає самої діаграми, яка називається "Структурна діаграма". Однак діаграма класів пропонує найкращий приклад типу структурної діаграми та надає нам початковий набір елементів позначень, які використовують усі інші структурні діаграми. І оскільки діаграма класів є такою основоположною, решта цієї статті буде зосереджена на наборі позначень діаграми класів. Наприкінці цієї статті ви повинні мати розуміння того, як намалювати схему класів UML 2, і мати міцну основу для розуміння інших структурних діаграм, коли ми розглянемо їх у наступних статтях.

На фрагменті нижче показано фрагмент діаграми класів, на якому відображено структура класів патерну Page Object.

Для реалізації функцій, описаних в технічному завданні дипломного проекту, необхідне створення наступних класів:

– BaseTest – клас, який відповідатиме за ініціалізацію виконання тестових сценаріїв та зв’язуватиме роботу всіх компонентів системи

– BasePage – клас, який надаватиме базову функціональність для роботи на веб сторінці, операції над об’єктами на веб сторінці: пошук елементу, клік, введення тексту, перевірка присутності елементу і т.д.

– DriverProvider – клас, який слугуватиме фабрикою для об’єктів типу WebDriver для виконання у різних браузерах та середовищах.

– ReporterManager – клас, який відповідатиме за формування логів під час виконання тестів, і звітів після завершення їх виконання.

– CucumberRunner – клас, який відповідатиме за виконання тестів, написаних на Gherkin notation, та виконуватиме роль точки входу в цикл виконання тестових сценаріїв.

– StepDefinitions – клас, в якому будуть реалізовані тестові кроки і прив’язані до тестових файлів.

Взаємозв’язки між основними класами системи показано на рисунку 2.10.

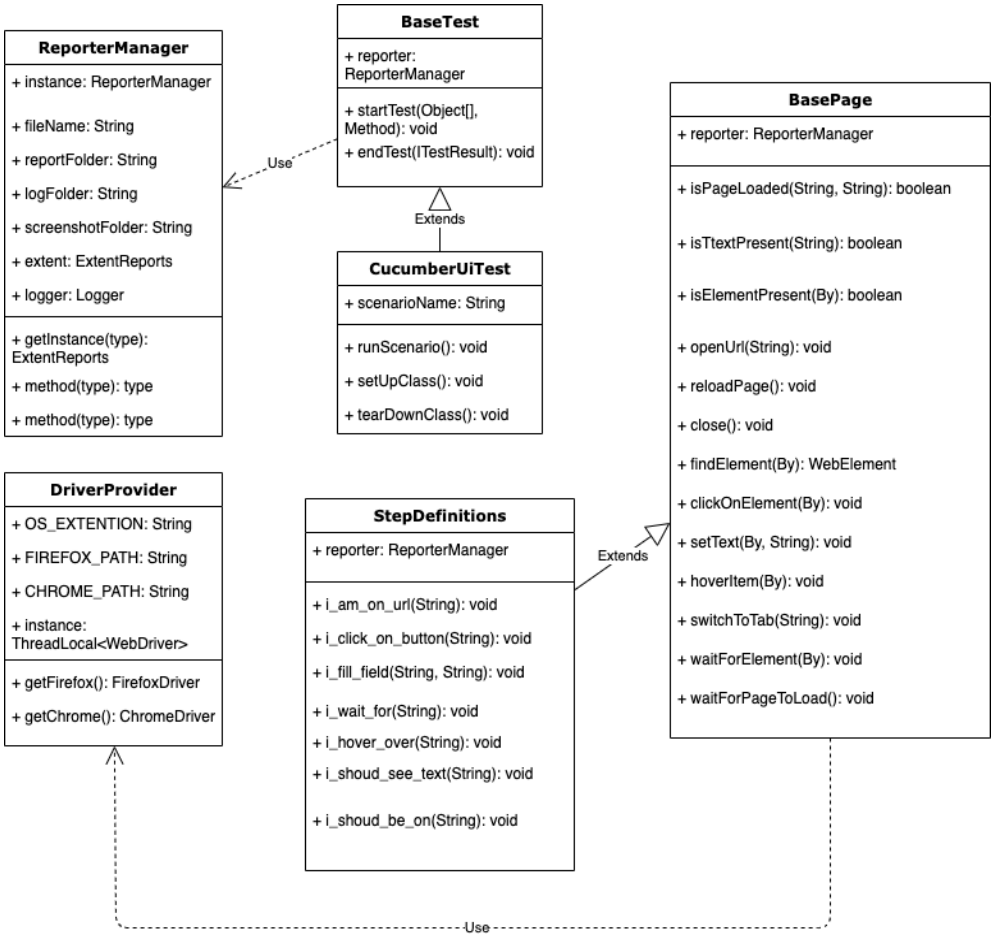

Рисунок 2.10 – Діаграма основних класів системи

Окремим структурним елементом системи варто виділити класи, які реалізуватимуть користувацький інтерфейс програми, тим самим переводячи її із площини фреймворку, чистого інструменту для розробки, ближче до повноцінного додатку. Це наступні класи:

– CodeEditorExample - точка входу для графічного інтерфейсу. Контроллер головного вікна програми;

– CodeEditor - клас, в якому знаходитимуться основні методи для елементів графічного інтерфейсу

– FeatureCRUD - Контроллер вікна створення нового .feature файлу. Клас, в якому будуть реалізовані основні операції з тестовими файлами: створення, редагування, перейменовування та інші.

– GherkinValidator - клас, який реалізує всю логіку написання тестових сценаріїв синтаксисом Gherkin notation

– GuiHelper - клас, в якому зібрані методи, які допомагають працювати з файлами, та перетворювати інформацію в них з одного формату в інший

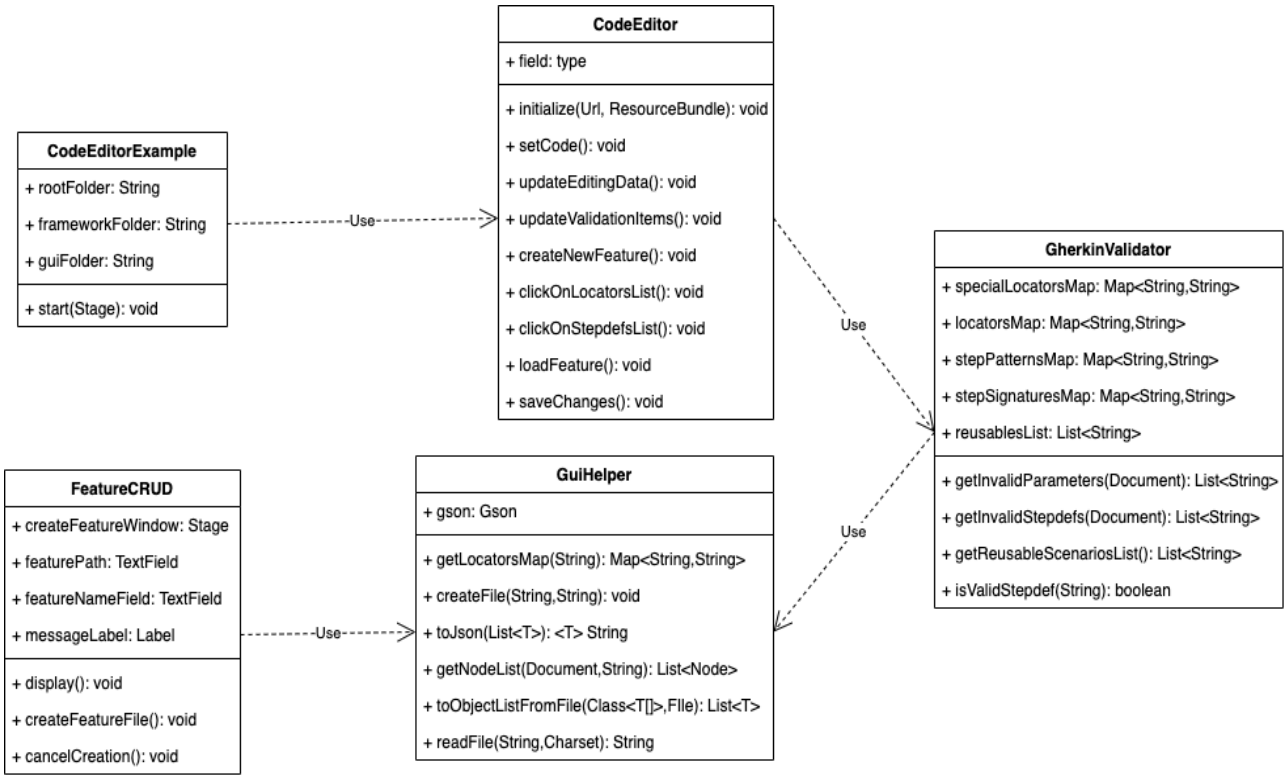

Рисунок 2.11 – Діаграма класів підсистеми графічного інтерфейсу

Модуль графічного інтерфейсу також включає в себе класи бібліотеки CodeMirror, написаній на JavaScript, яка дозволяє підсвічувати синтаксис мови Gherkin.

2.11. Відмовостійкість та обробка виключних ситуацій

Оскільки поведінка браузера не завжди передбачувана, тестування графічного інтерфейсу веб-програми є складним завданням, яке може залежати від багатьох факторів: швидкості та якості підключення до Інтернету, якості написаних локаторів для елементів та обладнання, на якому запущено тест. Важко повністю розглянути цю непередбачувану поведінку під час написання тестів, і це може бути джерелом виключних ситуацій.

Основний механізм обробки виключних ситуацій, зв'язаних з особливостями тестування графічного інтерфейсу, реалізований в класі UiHandlers.

Даний клас являться, по суті, перечисленням, елементами якого є лямбда-вирази, кожен з яких відповідає за обробку певної виключної ситуації, і реалізує функціональний інтерфейс UiHandler з методом handle().

Нижче приведений лістинг коду основних елементів даного перечислення:

```
public enum UiHandlers {

   SCROLL_HANDLER((element, e) -> {
       SeleniumHelper.isHandled.put("pfScrollHandler", false);

if(e.getMessage().contains("jss1bndbcl")||e.getMessage().contains("jss1wye7u4")){
           Reporter.log("Handling click overlay by scrolling 300 pixels down");
           SeleniumHelper.scrollBy(0,300);
           SeleniumHelper.isHandled.put("pfScrollHandler", true);
       }
   }),

   SCROLL_TO_ELEMENT_HANDLER((element, e) -> {
       SeleniumHelper page = new SeleniumHelper();
       SeleniumHelper.isHandled.put("pfScrollToElementHandler", false);
       if(e.getMessage().contains("Other element would receive the click:
<button")){
           Reporter.log("Handling click overlay by scrolling to element");
           SeleniumHelper.scrollToElement(page.findElement(element));
```

```
            SeleniumHelper.isHandled.put("pfScrollToElementHandler", true);
        }
    }),

    PAGE_NOT_LOAD_HANDLER((element, e) -> {

        SeleniumHelper page = new SeleniumHelper();
        StackTraceElement[] stackTraceElementArray = e.getStackTrace();
        StackTraceElement s = stackTraceElementArray[2];
        Boolean bool = s.getMethodName().contains("waitForPageToLoad")
                && e.getCause()==null;

        SeleniumHelper.isHandled.put("pageNotLoadHandler", false);
        if(bool){
            Reporter.log("Handling Page not load");
            page.executeJSCode("window.stop()");
            SeleniumHelper.repeatAction = false;
            SeleniumHelper.isHandled.put("pageNotLoadHandler", true);
        }
    }),

    DEFAULT_HANDLER((element, e) -> {
        boolean handled = false;
        for (Boolean value : SeleniumHelper.isHandled.values()) {
            if (value) handled = true;
        }
        if (!handled){
            SeleniumHelper.repeatAction = false;
            Reporter.log("Default Handler: FAIL");
            Reporter.fail(Tools.getStackTrace(e));
            throw new RuntimeException("Failure clicking on element");
        }
    });
```

Дане архітектурне рішення призначене для того, щоб передаючи дані лямбди як параметри у метод, де очікується виникнення виключної ситуації, можна було легко вибирати, який код виконати для обробки того чи іншого виключення.

2.12. Багатопоточність та синхронізація потоків

В рамках розробки передбачена можливість багатопотокового виконання тестів. Для реалізації багатопоточності в цій системі ми повинні спочатку забезпечити потокобезпечність класу DriverProvider, оскільки його об’єкт є головним виконавцем усіх тестових операцій у браузері.

Методом забезпечення багатопоточності в даному випадку є поміщення об’єкту драйвера у змінну ThreadLocal<WebDriver>

Клас Java ThreadLocal дозволяє створювати змінні, які можуть бути прочитані та записані лише одним потоком. Отже, навіть якщо два потоки виконують один і той же код, і код посилається на одну і ту ж змінну ThreadLocal, ці два потоки не можуть бачити змінну ThreadLocal один одного. Отже, клас Java ThreadLocal забезпечує простий спосіб зробити безпечний потік коду, який не був би таким в інших випадках.

2.13. Мережева взаємодія

Даний фреймворк в ході виконання тесту взаємодіє з деякими іншими зовнішніми компонентами:

BrowserStack - це хмарна веб-платформа для тестування мобільних пристроїв, яка дозволяє розробникам тестувати свої веб-сайти та мобільні додатки у браузерах на вимогу, операційних системах та реальних мобільних пристроях, не вимагаючи від користувачів встановлення або обслуговування внутрішньої лабораторії віртуальних машин, пристроїв чи емуляторів. Вони мають чотири основні продукти - Live, App Live, Automate та App Automate.

Користувачі можуть вибирати з більш ніж 1200 справжніх мобільних пристроїв, браузерів та операційних систем на вимогу та покладатися на безпечну, стабільну та масштабовану інфраструктуру для підтримки тисяч паралельних ручних та автоматизованих тестів. Скорочуючи витрати, витрати та технічне обслуговування, пов’язані з тестуванням, BrowserStack допомагає програмним командам тримати увагу на найважливішому - створення кращих продуктів та послуг із чудовим досвідом.

qTest - це інструмент управління тестами, який використовується для управління проектами, відстеження помилок та управління тестами. Він дотримується концепції централізованого управління тестами, яка допомагає легко

спілкуватися та допомагає у швидкій розробці завдання для команди QA та інших зацікавлених сторін.

qTest - це хмарний інструмент і був розроблений QASymphony. Він підтримує всі браузери, особливо Chrome, Firefox, IE тощо, а також підтримує різні версії ОС Windows - Windows XP, Vista, 7 і т.д. qTest можна інтегрувати з багатьма іншими інструментами - JIRA, Bugzilla, FogBugz, Версія перша тощо.

3. РОЗГОРТАННЯ ТА ТЕСТУВАННЯ

3.1. Розгортання програмної системи та системні вимоги

Для запуску даного програмного забезпечення, машина, на якій воно буде запускатися, має відповідати наступним вимогам:

- ОС – Windows (7, 8, 10), MacOS (Sierra або новіше)
- Не менше ніж 1 Гб вільного простору на диску
- Не менше ніж 4 Гб пам'яті ОЗУ
- Встановлено IntelliJ IDEA 2018.1 або новіше
- Встановлено Java Virtual Machine
- Встановлено JDK 8 або новіше
- Встаовлено Apache Maven 3.0.0 або новіше

Дана програмна система поставляється у вигляді бібліотеки класів, тому для розгортання її необхідно імпортувати до Maven проекту, а також під директорією test створити папку bdd, звідки фреймворк братиме тестові файли.

Бібліотека класів або пакет (package) використовується в додатках Java для вирішення проблеми конфлікту імен класів, інтерфейсів і методів. Зазвичай вони об'єднують класи подібного призначення. Всі стандартні бібліотеки класів Java розроблені як пакети і поставляються у вихідних текстах разом з JDK і Sun Java WorkShop.

Ви можете створити свою бібліотеку класів, розмістивши в першому рядку вихідного тексту оператор package:

```
package some.pack.name;
```

Тут після оператора package розташовується ім'я бібліотеки. Зазвичай воно складається з декількох рядків, між якими ставиться крапка. Для виключення конфлікту імен можна додавати до імені бібліотеки префікс, наприклад, доменну адресу сервера Web вашої фірми або просто назва фірми.

В результаті компіляції проекту бібліотеки класів виходить файл з розширенням імені class. Його необхідно розмістити в каталозі, де знаходяться всі локальні бібліотеки класів Java. Цей каталог визначається змінного середовища з ім'ям CLASSPATH.

Для використання класів і інтерфейсів, визначених у бібліотеці, необхідно включити в вихідний текст програми оператор import, наприклад:

```
import some.pack.name. *;
```

В даному випадку імпортуються всі класи і інтерфейси, визначені в бібліотеці some.pack.name. Зауважимо, що поза бібліотекою можливе використання тільки таких класів і інтерфейсів, які визначені як public.

3.2. Тестування

Створюючи структуру для автоматизованого тестування, слід переконатися, що програмне забезпечення працює належним чином і результати правильні, що може бути використано як інформація про правильність веб-програми.

Саме тому необхідно створити, або використати наявний набір тест-кейсів, який буде контрольним пакетом для перевірки розробленого програмного забезпечення. Одні й ті ж операції необхідно виконати вручну, а потім повторити за допомогою автоматизованих тестів.

Для такої перевірки необхідна наявність персонального комп’ютера з ресурсами вище середнього, щоб отримати оптимальні результати часу виконання тестів. Також необхідний доступ до мережі Інтернет, швидкість з’єднання близько 50 мБіт, оскільки автоматизовані тести розраховані на певний період очікування. Тому елементи web-сторінки повинні завантажуватись недовго.

В ідеалі різні набори автоматизованих тестів можна запускати паралельно в різних потоках. Таким чином можна пришвидшити виконання автоматизованих тестів, хоча для таких рішень необхідно використовувати потужніші ПК.

Для перевірки розробленого програмного забезпечення було обрано набір тест-кейсів та виконано їх вручну. Після цього той же набір тест-кейсів було пройдено за допомогою розробленого програмного комплексу.

В якості тестового середовища було обрано ресурс, призначений для перевірки автоматизованих тестів Automationpractice.com [9]. Даний ресурс являє собою зразок типового інтернет магазину.

Серед інших подібних ресурсів, Automationpractice.com було обрано перш за все через те, що електронна комерція зараз займає левову частку всіх IT-проектів, а саме тому є найбільш наближеним середовищем до реального проекту, і саме тому найкраще підходить для його тестування.

Результати було співставлено і проаналізовано. Було отримано співпадіння позитивних та негативних тестів у 100% випадків. Порівняльна характеристика мануальних та автоматизованих тестів наведена в Таблиці 3.1

Таблиця 3.1 – Результати перевірки тестового набору даних

Назва тестового сценарію	Очікуваний результат	Результат мануального проходження	Результат роботи розробленої програми
Додати продукт до кошика	Продукт додано до кошика	PASSED	PASSED
Видалити продукт з кошика	Продукт видалено з кошика	PASSED	PASSED
Перевірити, що ціна продукту на сторінці лістингу співпадає з ціною на сторінці продукту	Ціна продукту на сторінці лістингу співпадає з ціною на сторінці продукту	PASSED	PASSED
Додати продукт до списку порівняння	Продукт додано до списку порівняння	PASSED	PASSED
Перевірити, що ціна продукту на сторінці лістингу співпадає з ціною на сторінці корзини	Ціна продукту на сторінці лістингу співпадає з ціною на сторінці корзини	PASSED	PASSED

Продовження таблиці 3.1

Реєстрація користувача на сайті	Користувач має змогу зареєструватися на сайті	PASSED	PASSED
Підписатися на розсилку	Користувач отримує розсилку на вказану пошту	PASSED	PASSED
Перевірити коректність роботи пошуку на сайті	Пошук по продуктам видає релевантні результати	PASSED	PASSED

Зазначені вище тести в достатній мірі розкривають можливості роботи розробленого програмного забезпечення. Повний опис даних тестових сценаріїв представлений у Додатку В.

Таким чином, можемо зробити висновок, що розроблене програмне забезпечення працює коректно, та може бути застосоване для отримання інформації про правильність роботи web-сторінки, зокрема типових елементів електронної комерції. З виходом нових версій проекту, зміною розташування, атрибутів елементів web-сторінки, редагування і підтримки потребуватимуть лише локатори, вигесені в окремий файл. Автоматизовані тести, натомість, змін зазнавати не будуть, оскільки використовують локатори лише за їх іменем, яке, при зміні самих значень локаторів, змінюватися не буде.

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Охорона праці

У наш час, завдяки швидкому розвитку ІТ-технологій (комп'ютерного обладнання), мобільний зв'язок та електронна інформаційна безпека різко зросли, і надалі збільшуватиметься кількість видів діяльності, які використовують інформаційні технології. Будь-яке заняття характеризується певними професійними недугами або захворюваннями, пов'язаними з виробництвом. Професійні захворювання, пов'язані з використанням технічних засобів в ІТ-індустрії не є винятком. «Комп'ютерні» хвороби та галузі, що призвели до них, все ще молоді і рідко вивчаються.

Тому при користуванні комп'ютером найбільше страждають органи зору, кістково-м'язова система, репродуктивна функція та центральна нервова система. Крім того, на користувачів комп'ютерів впливає також ряд факторів низької інтенсивності, і ці негативні наслідки поступово і приховано виявляються. Тому хвороба з'являється лише після роботи протягом декількох місяців, а то й років. У цей час боротьба із недугом стає надзвичайно складною. У багатьох випадках навіть важко визначити першопричину захворювання. За цієї обставини необхідно формувати відповідні знання про конкретну ситуацію впливу несприятливих виробничих факторів на робоче місце серед майбутніх фахівців та застосовувати інформаційні технології для мінімізації цього впливу та збереження здоров'я та зайнятості користувачів. Заходи та інструменти довголіття.

Не існує абсолютно безпечних та нешкідливих умов праці. Характеристиками виробничого середовища завжди є наявність певних факторів ризику, що можуть загрожувати здоров'ю працівників. Складні умови праці, пожежі, вибухи, аварії та інші причини призводять до того, що у світі щороку на виробництві трапляється до 50 мільйонів нещасних випадків, що призводить до загибелі понад 250 000 здорових людей (в Україні більше, ніж щороку до 1000 осіб).

Законодавство України про охорону праці включає конституційну гарантію прав громадян у цій галузі, Закону України «Про охорону праці», українське законодавство про працю та інші закони, пов'язані із захистом життя та здоров'я громадян під час працевлаштування та ведення трудової діяльності.

У Конституції України затверджується: «Людина, її життя й здоров’я, безпека визнаються в Україні найвищою соціальною цінністю». В інших статтях проголошені права громадян на:

– належні, безпечні й здорові умови праці (ст. 43);

– соціальний захист громадян, забезпечення їх у випадку повної, часткової або тимчасової втрати працездатності, втрати годувальника (ст. 46);

– охорону здоров’я, медичну допомогу й страхування (ст.49);

– безпечне для життя й здоров’я навколишнє середовище й відшкодування заподіяної порушенням цього права шкоди (ст. 50).

Основним законодавчим документом що регулює відносини у сфері охорони праці є Закон України "Про охорону праці", який поширюється на всі компанії, установи та організації, незалежно від форм власності та виду діяльності, на всіх громадян, які працюють і працюють у цих компаніях.

Цей Закон визначає основні положення по реалізації конституційного права трудящих на охорону їх життя й здоров’я в процесі трудової діяльності, на належні, безпечні й здоровіші умови праці, регулює відносини між роботодавцем і працівником з питань безпеки, гігієни праці й виробничого середовища й установлює єдиний порядок організації охорони праці на Україні.

У цьому Законі визначені основні принципи державної політики в галузі охорони праці, серед яких:

– пріоритет життя й здоров’я працівників стосовно результатів виробничої діяльності підприємства;

– повна відповідальність власника за створення безпечних і нешкідливих умов праці;

– соціальний захист працівників, які постраждали від нещасних випадків на виробництві й професійних захворювань;

– встановлення єдиних нормативів з охорони праці для всіх форм власності й видів їх діяльності;

– здійснення навчання населення, професійної підготовки й підвищення кваліфікації працівників з питань охорони праці;

– використання економічних методів керування охороною праці й т.п.

У процесі розробки програмного забезпечення для роботи з BDD тестами було встановлено, що робота людини яка проводилася в тривалих статичних навантаженнях та умовах обмеженої рухливості, може викликати різноманітні захворювання серцево-судинної та опорно-рухової систем, а також проблеми з зором. Для

Засоби відображення інформації, які вимагають точного й швидкого зчитування показань, повинно розміщувати у вертикальній площині під кутом ± 15° від нормальної лінії погляду та в горизонтальній площині під кутом ± 15° від сагітальної площини.

Організація комп'ютеризованого робочого місця повинна відповідати вимогам ергономіки ДСТУ 8604:2015. «Робоче місце для виконання робіт сидячи. Загальні ергономічні вимоги», НПАОП 0,00-7,15-18, а також специфіці роботи за ПК.

Конструкція виробничого устаткування й робочого місця повинна забезпечувати оптимальне положення тіла працюючого, що досягається регулюванням:

- висоти робочої поверхні, сидіння й простору для ніг. Регульовані параметри потрібно вибирати відповідно до вимог ДСТУ ISO 14738;

- висоти сидіння та підставки для ніг (у разі нерегульованої висоти робочої поверхні). У цьому разі висоту робочої поверхні встановлюють за номограмою (рисунок 4) для працюючих зростом 1800 мм.

Монітор повинен розміщуватися у вертикальній площині під кутом ± 15° від нормальної лінії погляду та в горизонтальній площині під кутом ± 15° від сагітальної площини, для відображення інформації, яка вимагає точного й швидкого зчитування показань.

Для менш точного й швидкого зчитування показань, допускається розташовувати монітор у вертикальній площині під кутом ± 30° від нормальної лінії погляду й у горизонтальній площині під кутом ± 30° від сагітальної площини

Для того, щоб оцінити належність підготовки робочого простору відповідно до закону потрібно виміряти його розмір. Тому в кімнаті площею 12 квадратних метрів (4х3 метри) та висотою стелі 3,4 метра, можна розмістити до двох робочих місць з комп'ютерами (тобто на одне робоче місце повинно бути виділено не менше 6 квадратних метрів).

Всі нові та реконструйовані електроустановки напругою до 500 кВ повинні відповідати вимогам Правилам Улаштування Електроустановок (ПУЕ). Якщо ПУЕ може спростити електроустановку, довести, що вартість реконструкції є обґрунтованою за допомогою технічних та економічних розрахунків, або якщо реконструкція спрямована на забезпечення вимог безпеки, що застосовуються до існуючих електроустановок, деякі вимоги ПУЕ можуть бути застосовані і до існуючих електроустановок.

ПК та обладнання, що використовуються для технічного обслуговування, ремонту та регулювання, повинні мати захист від струму короткого замикання (максимальний струмовий захист). Провід і силові кабелі повинні використовувати переважно негорючі ізоляційні матеріали.

Таким чином, проаналізувавши приведені вище вимоги щодо обладнання робочих місць відповідно до діючих норм охорони праці та безпеки життєдіяльності, можна зробити висновок, виробничий процес та умови праці на проекті розробки системи для написання та менеджменту автоматизованих BDD тестів на базі Java та Cucumber, організовані згідно з чинним законодавством України у сфері охорони праці, та відповідають всім нормам, описаним у цьому підрозділі.

4.2. Безпека в надзвичайних ситуаціях

Фінансування заходів ЦЗ відбувається за рахунок державного та місцевих бюджетів, а також коштів госпрозрахункових підприємств і організацій.

Центральні та місцеві органи державної виконавчої влади і підприємства відраховують кошти на проведення заходів щодо навчання і захисту населення, включаючи витрати на утримання і підготовку територіальних органів управління і формувань, призначених для ліквідації наслідків надзвичайних ситуацій.

При виникненні надзвичайних ситуацій, основну та найбільшу небезпеку для користувачів комп'ютерів становить електричний струм під високою напругою, джерелом якого можуть стати пошкоджені лінії електромережі, або пошкоджені електроприлади.

Пошкодження організму людини, спричинене дією електричного струму та / або дуги, називається електричним пошкодженням. Наприклад, через опіки або сліпоту, спричинені дугами, електрошоки можуть виникнути під час процесу пропускання струму крізь тіло людини чи через нього. Явище, що характеризується групою уражень електричним струмом, називається ураженням електричним струмом.

За статистичними даними, середня частка уражень електричним струмом у всіх виробничих травмах в Україні становить близько 1%, а летальний результат сягає 20%, що більше за інші причини. Слід зазначити, що до 80% смертельних уражень електричним струмом відбуваються в електрообладнанні напругою до 1000 В. Це пов'язано з всюдисущістю низьковольтного електрообладнання та тим, що майже всі працівники можуть ним користуватися, а електричне обладнання напругою більш ніж 1000 В може відремонтувати лише висококваліфікований персонал.

Що стосується умов праці, то в абсолютних показниках в Україні в середньому спостерігається 500 смертей від ураження електричним струмом на рік, у тому числі смертей - 150 випадків на рік. Широке використання електроенергії у всіх галузях економіки призводить до збільшення кількості людей, залучених до

експлуатації електрообладнання. Тому питання електробезпеки при експлуатації електрообладнання є особливо важливими.

У порівнянні з іншими видами електротравматизму характерні такі особливості:

– Людина не може віддалено визначити існування напруги без спеціального обладнання, тому вплив струму, як правило, виникає раптово, і лише після впливу напруги він виявить захисну реакцію на організм людини;

– Струм, що протікає по людському тілу, діє не тільки на частини та струми, що контактують з частинами, що знаходяться під напругою, але і діє на тіло рефлекторно. Це дуже сильний подразник, що впливає на весь організм людини, що може призвести до функції життєво важливої системи Аномалії: нерви, дихання, серцево-судинна система та інші;

– Навіть якщо ніхто не контактує з струмоведучими частинами, це може спричинити ураження електричним струмом - утворюється дуга через розрив повітряного зазору між струмоведучими частинами або між струмоведучими частинами та людьми або землею.

Вплив електричного струму на живі тканини різноманітний і унікальний. Електричний струм, що проходить через тіло людини, має теплову, електролітичну та біологічну дію.

Тепловий ефект електричного струму полягає у нагріванні біологічних тканин, а вода випаровується, що може спричинити опіки певних частин людського тіла та пошкодження біологічних тканин парою. Нагрівання до високої температури органу, розташованого на поточному шляху, може спричинити серйозні порушення функції.

Електроліз електричного струму проявляється як розкладання органічних рідин, включаючи кров, і порушує його фізичний та хімічний склад.

Біологічний ефект електричного струму полягає у стимулюванні та втручанні в живі тканини людського тіла та руйнуванні внутрішніх біологічних процесів, які можуть проявлятися у формі мимовільних та непередбачуваних скорочень м’язів та захворювань життєво важливих органів, включаючи серце та легені.

Електричні травми умовно поділяють на місцеві, загальні і змішані.

Місцеві травми включають електричні опіки, електричні сліди, металізацію шкіри, механічні пошкодження та електроокулопатію.

Найпоширеніша електрична травма - це електричні опіки. За умовами походження вони поділяються на контактні, дугові та змішані. Контактні опіки зазвичай трапляються в низьковольтному обладнанні і викликані тепловою дією електричного струму. Вони покривають ділянки шкіри та тканин поблизу місця контакту. У разі небезпечного контакту між струмопровідними частинами електрообладнання та між струмопровідними частинами електрообладнання та тілом людини, якщо в електрообладнанні трапляється випадкове коротке замикання, через дуговий розряд може виникнути дуга.

Електричні знаки – різко окреслені плями сірого чи блідо-жовтого кольору, які з являються на поверхні тіла людини в місці контакту із струмовідними елементами. Особливого больового відчуття електричні знаки не спричиняють і з часом безслідно зникають.

Металізація шкіри пов’язана з проникненням відкритих ділянок людського тіла в шкіру дрібних металевих частинок, як правило, таючих під дією електричної дуги. Металізація особливо небезпечна для очей.

Механічна травма спричинена неконтрольованим скороченням м’язів, що смикаються, спричиненими стимуляцією електричним струмом. Це проявляється як сухожилля, шкіра, судина, розрив нервової тканини, вивих суглоба, перелом та інші форми.

Електроофтальмія - це запалення зовнішньої оболонки ока, спричинене дією ультрафіолетового випромінювання, що генерується дугою. Запалення виникає через кілька годин після опромінення і проявляється у вигляді почервоніння, сльозотечі, гнійних виділень та світлобоязні шкіри та слизової оболонки повік. Тривалість захворювання 3-5 днів.

ВИСНОВКИ

В результаті виконаної роботи розроблено програмну систему для написання та менеджменту автоматизованих BDD тестів на базі Java та Cucumber.

Також в роботі було розглянуто різні засоби для автоматизації тестування web-сервісів. Після аналізу для даної розробки було обрано Selenium WebDriver, оскільки даний продукт має ряд переваг над аналогами.

Для розробки було обрано мову програмування Java, оскільки це сучасна об’єктно-орієнтована мова, яка дозволяє реалізувати весь необхідний функціонал. Дана мова програмування має ряд переваг, такі як можливість створення делегатів, використання lambda-виразів, а також має одну з найвищих швидкодій серед об’єктно-орієнтованих мов. Також робота побудована з використанням TestNG - потужного фреймворка для тестування, який був розроблений для створення автоматизованих тестів.

Ядро системи розроблене на вигідному гібриді Page Object Model та Behaviour Driven Development моделей, суміщає в собі модульність та організованість першої із зрозумілістю та універсальністю другої. Завдяки тій же модульності, ядро даної системи може використовуватися як фреймворк для розробки тестового покриття Unit-тестами та Api-тестами, а також для подальшої розробки корисних надбудов для нього.

Графічний інтерфейс програми розроблений на платформі JavaFX і представлений у вигляді окремого підпроекту. Даній інтерфейс забезпечує користувача можливістю на інтуїтивно зрозумілому рівні керувати файлами з тестовими сценаріями: створювати, редагувати, видаляти та перейменовувати, а також реалізує зручне користування усіма надбудовами над фреймворком Cucumber для ефективної реалізації BDD методології в IT-проекті.

В результаті роботи було розроблено програмне та алгоритмічне забезпечення для автоматизованого тестування, яке задовільняє всі вимоги, поставлені до нього в підрозділі 2.2 “Аналіз вимог. Визначення вимог.” даної роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Майерс Г. Мистецтво тестування програм: пер. з англ. / Майерс Г., Баджет Т., Сандлер К. – М.: Діалектика, 2012. – 270 с.

2. Кріспін Л. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд : пер. з англ. / Л. Кріспін., Д. Грегорі – М.: «Вільямс», 2011. – 463 с. – ISBN 0-471-46912-2.

3. Липаев В. В.Тестирование компонентов и комплексов програм: підруч. / Липаев В. В. – М.-Берлін: Діректт-Медіа, 2015. – 528с.

4. Азарский К. И. Тестирование. Легкий старт / Азарский К. И. – Mumai: Copyright, 2014. – 226 с. – ISBN 978-5-496-00893-8.

5. Бейзер Б. В. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем / Бейзер Б. В. – СПб.: Питер, 2012. – 318 с. – ISBN 5-94723-698-2.

6. Advolodkin N. Is BDD Automation Actually Killing Your Project? [Electronic resource] – 01.08.2020 – Mode of access: https://saucelabs.com/blog/is-bdd-automation-actually-killing-your-project – SauseLabs.

7. Nagy G., Rose S. Discovery. Explore behaviour using examples [Electronic resource] – 01.08.2020 – Mode of access: http://bddbooks.com/ - The BDD Books.

8. Copeland L. A Practitioner’s Guide to Software Test Design / Copeland L. – London: Artech House Publishers, 2013. – 238 с. – ISBN 1-503-791-X.

9. Automation Practice [Electronic resource] – 01.08.2020 – Mode of access: http://automationpractice.com/

10. Whittaker J. A. Exploratory Software Testing / Whittaker J. A. – Boston: Addison-Wesley, 2011. – 253 с. – ISBN 978-0-321-63641-6.

11. Whittaker J. A. Как тестируют в Google / Whittaker J. A., Arbohn J, Karolo J.– СПб.: Питер, 2014. – 75 с. – ISBN 978-5-496-00893-8.

12. Osherove R. The Art of Unit Testing / Osherove R. – Connecticut: Manning, 2014. – 296 с.– ISBN 978-1617290893.

13. UML 2.0 Object Constraint Language [Electronic resource] – 11.10.2014 – Mode of access: https://www.omg.org– Title from screen.

14. Gamma E. Design Patterns: Elements of Reusable Object-Oriented Software [Text] / E. Gamma, R. Helm, R. Johnson, J. Vlissides. – Addison Wesley, 2005. – 213 p.

15. Ларман К. Применение UML 2.0 и шаблонов проектирования; пер. с англ. [Текст] / К Ларман – М.: Вильямс, 2009. – 304 с.

16. Тротт Д. Р. Шаблоны проектирования. Новый подход к объектноориентированному анализу и проектированию. Design Patterns Explained: A New Perspective on Object-Oriented Design [Текст] / Д. Р. Тротт. – М.: «Вильямс», 2002. – 288 с.

17. Лафоре Р. Объектно-ориентированное программирование в C++. – 4-е издание [Текст] / Р. Лафоре – СПб.: Питер, 2005. – 928 с.

18. Петрик М. Моделювання та аналіз програмного забезпечення [Текст] / Петрик М., Петрик О. -Тернопіль: Видавництво ТНТУ, 2015. – 210 с.

19. Osmani A. UML 2.0 et UML 2.0 Infrastructure [Text] / Benoît Charoux, Aomar Osmani, Yann Thierry-Mieg. Editions Pearson, Education France, 2008. – 87 p.

20. В.І. Голінько, М.Ю. Іконніков, Я.Я. Лебедєв Охорона праці в галузі інформаційних технологій [Текст] – Дніпропетровськ : НГУ – 2015. – 247 с. 55. М.С. Одарченко, А.М. Одарченко, В.І. Степанов, Я.М. Черненко. Основи охорони праці [Текст] – Харків : Стиль-Издат – 2017 – 341 с

ДОДАТКИ

ДОДАТОК А

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ
КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання атестаційної роботи магістра на тему
«Розробка системи для написання та менеджменту автоматизованих BDD тестів на базі Java та Cucumber»

Розробники:
виконавець ст. гр. СПм-61
Левицький Руслан Васильович

_____
(підпис)

керівник проекту
Бойко Ігор Володимирович

_____
(підпис)

Тернопіль 2020

ЗМІСТ

1. ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до графіку навчального плану на 2019 рік, та згідно наказу на виконання дипломної роботи студента-магістра.

Тема проекту: «Розробка системи для написання та менеджменту автоматизованих BDD тестів на базі Java та Cucumber».

Термін виконання: до __.___._____р.

2. ПРИЗНАЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Тестовий фреймворк призначений для написання автоматизованих тестових сценаріїв, зручного відслідковування тестового покриття AUT(Application Under Test), вичерпного репортінгу про виконання тестів та можливістю запуску з систем безперервої інтеграції (CI/CD).

Інформаційна система призначена для використання як automation так і manual QA інженерами за рахунок BDD (Behavior Driven Testing) підходу до написання тестових сценаріїв. Система буде також корисною для Проектних Менеджерів, які зможуть відслідковувати існуюче тестове покриття, та пропонувати свої варіанти тестових сценаріїв для його розширення.

Система дозволить власникам створити стабільне, якісне і відслідковуване тестове покриття свого програмного продукту, забезпечити належний рівень його якості на всіх етапах життєвого циклу, на регулярній основі надавати інформацію про статус продукту, рівень його «здоров'я», завершеності та покриття тестами. Система надаватиме можливість створення окремих тестових «с'ютів» для проведення різноманітних типів тестування: Regression, Smoke, Sanity та інших.

3. ВИМОГИ ДО ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Функціональні вимоги

Система повинна надавати можливість написання та запуску автоматизованих тестових сценаріїв для таких користувачів:

- QA Automation Engineer
- Manual QA Engineer
- Project Manager

Система повинна надавати доступ до наступного переліку функцій:

- створення, редагування, видалення тестових сценаріїв [лише для QA Automation Engineer];
- перегляд списку усіх існуючих тестових сценаріїв [Project Manager];
- написання тестових сценаріїв [Manual QA Engineer];
- запуск автоматизованих тестових сценаріїв [Project Manager, Manual QA Engineer]
- зміна конфігурації запуску автоматизованих тестових сценаріїв [QA Automation and Automation Engineer]
- перегляд репортів про виконання автоматизованих тестових сценаріїв [Project Manager, Manual QA Engineer]

3.2 Технічні вимоги

ОС Windows або MacOS, не менше ніж 8Гб ОЗП, не менше 512 гігабайт на жосткому диску;

3.3 Програмні вимоги

Мова розробки: Java
Версія Development Kit: JDK8
Тестове ядро: TestNG
Тестова модель: Page Object Model, Behavior Driven Testing
Репортинг: Extent Reports
Додаткові вимоги: Інтеграція з CI/CD сервером, розміщення в системі контролю версій GIT (Git Hub або Bitbucket);

4. ЕТАПИ РОЗРОБКИ

Розробка інформаційної системи проводиться в наступному порядку:

- аналіз предметної області, виявлення акторів та варіантів використання системи;
- проектування архітектури програмної системи;
- моделі аналізу (побудова UML-діаграм)
- розробка програмного забезпечення системи;
- тестування інформаційної системи на реальних даних;
- оформлення супровідної документації;
- здача проекту.

Результати виконання кожного етапу проекту погоджуються з керівником проекту.

5. СУПРОВІДНА ДОКУМЕНТАЦІЯ

Для інформаційної системи повинні бути розроблені наступні документи:

- пояснювальна записка до проекту;
- презентація проекту;
- рецензія на проект;
- диск з проектом.

Пояснювальна записка до проекту оформляється згідно діючих вимог до нормоконтролю проектів.

6. ПОРЯДОК ЗДАЧІ ПРОЕКТУ

Розроблена інформаційна система повинна відповідати вимогами, що складаються з перерахованих у п.3.1 цього документу характеристик.

Для здачі проекту необхідно підготувати весь перелік документів зазначений у п.5 цього документу.

Приймання проекту проводиться спеціально створеною комісією в термін зазначені в п.1 цього документу.

7. ВІДМІТКИ ПРО ВИКОНАННЯ ЕТАПІВ ТА ЗМІНИ В ПРОЕКТІ

Назва етапу	Відмітка*
Аналіз предметної області	
Проектування архітектури системи	
Моделі аналізу	
Реалізація системи	
Супровідна документація	

* відмітки про виконання етапу ставляться керівником проекту

ДОДАТОК Б

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

М А Т Е Р І А Л И

VIII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

9-10 грудня 2020 року

ТЕРНОПІЛЬ
2020

УДК 001
М34

ПРОГРАМНИЙ КОМІТЕТ

Голова: Лупенко Сергій Анатолійович - докт. техн. наук, професор.

Співголови: Марущак Павло Орестовищ - проректор з наукової роботи, докт. техн. наук, професор.
Баран Ігор Олегович - канд. техн. наук, доцент, декан факультету ФІС.

Науковий секретар: Семенишин Галина Мирославівна - старший викладач.

Члени: докт. фіз.-мат. наук, професор В. Кривень; докт. техн. наук, професор М. Приймак; канд. техн. наук, доцент, Г. Осухівська; докт. техн. наук, професор М. Карпінський; канд. пед. наук, доцент Ж. Баб'як; докт. фіз.-мат. наук, професор М. Петрик; канд. техн. наук, доцент Н. Загородна.

ОРГАНІЗАЦІЙНИЙ КОМІТЕТ

Голова: Скоренький Юрій Любомирович - канд. техн. наук, доцент.

Члени: канд. екон. наук, доцент І. Струтинська; канд. техн. наук, доцент Я. Кінах; асистент М. Стадник; асистент Н. Шаблій; ст. викладач Л. Джиджора.

Матеріали VIII науково-технічної конфції «Інформаційні моделі, системи та технології» Тернопільського національного технічного університету імені Івана Пулюя, (Тернопіль, 9 - 19грудня 2020р.). - Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2020. - 196 с.

Адреса оргкомітету: ТНТУ ім. І. Пулюя, м. Тернопіль, вул. Руська, 56, 46001,
тел. (0352) 52-41-33, факс (0352) 254983.
E-mail: conffis2020@gmail.com
Редагування, оформлення, верстка: Семенишин Г.М.

СЕКЦІЇ КОНФЕРЕНЦІЇ, ЯКІ ПРЕДСТВЛЕНІ В ЗБІРНИКУ

- Математичне моделювання;
- Інформаційні системи та технології;
- Комп'ютерні системи та мережі;
- Програмна інженерія та моделювання складних розподілених систем;
- Новітні фізико-технічні та освітні технології.

В збірнику надруковано тези доповідей УШ науково-технічної конференції «Інформаційні моделі, системи та технології» (Тернопіль, 9-10 грудня 2020 р.) за такими науковими напрямками: математичне моделювання; інформаційні системи та технології; комп'ютерні системи та мережі; програмна інженерія та моделювання складних розподілених систем; новітні фізико-технічні та освітні технології.

Розрахований на науковців, викладачів та студентів вузів.

За зміст тез та дотримання норм академічної доброчесності відповідальність несе автор.

УДК 004.891.3: 004.4
Р. Левицький
(Тернопільський національний технічний університет імені Івана Пулюя)

ВПРОВАДЖЕННЯ BDD МЕТОДОЛОГІЇ В ІТ ПРОЕКТІ

UDC 004.891.3: 004.4
R. Levytskyi

ADOPTING BDD METHODOLOGY IN IT PROJECT

Поведінкова розробка (BDD) - це процес розробки програмного забезпечення, який спрямований на вирішення проблеми реалізації невизначених вимог. Він призваний використовувати доменні знання бізнесу та професіоналів із забезпечення якості, щоб розробники створювали правильне програмне забезпечення.

У якийсь момент організація вирішує, що потрібно пришвидшити релізи і швидше вийти на ринок. Тож вона переходить до Agile методології для досягнення цих цілей. Це може бути ще не повноцінний Agile, але команда працює по спринтах із частими релізами.

Однак, оскільки системи контролю якості намагаються йти в ногу з розробкою, тестування часто стає вузьким місцем при переході до Agile. У відповідь на це, команди вбудовують тестування в процес розробки, що полегшує запобігання лишнім валідаціям, які можуть загрожувати приросту продуктивності, отриманого шляхом переходу на Agile методологію.

На даному етапі не тільки пришвидшується розробка, але й забезпечується якість програмного забезпечення завдяки постійним практикам тестування. Але звідки команда може знати, що процес побудований правильно? Звичайно, це може працювати, але чи це те, що хоче або потребує клієнт?

Призначення BDD якраз і полягає у забезпеченні чіткого визначення та правильного виконання вимог до керівництва роботою з розробки.

Згідно з підходом BDD, під час написання визначення кроку, розробник повинен написати код, щоб визначити операцію, яку слід виконати, коли виконується тестовий крок, який відповідає визначенню.

Деякі засоби автоматизації тестів можуть зменшити технічні бар'єри на шляху впровадження BDD, дозволяючи розробникам автоматизованих тестів писати файли з тестовими кроками з більш простим синтаксисом, ніж Java, .NET або іншу мова програмування. Ці ж іструменти дозволяють в подальшому виконувати визначені кроки як автоматизовані тести, а також надавати звіти про їх виконання в зручному для сприйняття вигляді.

Прикладом такого засобу автоматизації може слугувати реалізація BDD підходу на мові Java, який має назву Cucumber. Даний фреймворк дозволяє записувати тестові кроки англійською, або однією з багатьох інших доступних мов, та асоціювати ці кроки з написаними на мові Java методами, які реалізують ці кроки.

Наприклад, визначення кроку може посилатися на окремий тест, який викликає REST API. Визначення другого кроку може стосуватися іншого тесту, який перевіряє дані відповіді з іншого REST API. Визначення третього кроку може стосуватися тесту, який виконує запит до бази даних і перевіряє дані у наборі результатів.

Як результат, засоби автоматизації тестування забезпечують тестувальникам більше контролю. Їхня впевненість у повноті написаного тестового покриття також зростає. Використовуючи зріле, повністю функціональне рішення для наскрізного тестування, можна легко розпочати автоматизацію тестування, підтримку тестів та природну інтеграцію в існуючий робочий процес CI/CD. На більш високому рівні це дозволяє організації використовувати менше технічних ресурсів для впровадження BDD, тим самим звільняючи ресурси для власне розробки програмного забезпечення.

ДОДАТОК В

Тестові сценарії на мові Gherkin

```
@Demo @AutomationPratice @Checkout
Feature: Checkout Process
  As a Customer
  I want to be able to go through checkout process
  So that I can purchase a product in the online-store


  @Positive @ShoppingCart @PLP #First test scenario
  Scenario: Add Product to Cart
    Given I am on "http://automationpractice.com/index.php" URL
    When I set "Dress" text to the "search_query" "element by name"
    And I click on the "submit_search" "element by name"
    And I hover over the "Printed Summer Dress" "AutomationPractice
PLP Product Tile"
    And I click on the "Printed Summer Dress" "AutomationPractice
PLP Add to Cart Button"
    Then I should see the "Product successfully added to your
shopping cart" "exact element"

  @Negative @ShoppingCart @PLP #And example of negative scenario
  Scenario: Delete Product from Mini Cart
    Given I am on "http://automationpractice.com/index.php" URL
    When I set "Dress" text to the "search_query" "element by name"
    And I click on the "submit_search" "element by name"
    And I hover over the "Printed Summer Dress" "AutomationPractice
PLP Product Tile"
    And I click on the "Printed Summer Dress" "AutomationPractice
PLP Add to Cart Button"
    Then I should see the "Product successfully added to your
shopping cart" "exact element"
    And I click on the "Close window" "element by title"
    When I hover over the "Cart" button
    And I click on the "Printed Su..." "AutomationPractice Mini Cart
Delete Button"
    When I hover over the "Cart" button
    Then I shouldn't see the "Printed Summer Dress"
"AutomationPractice Mini Cart Delete Button"
```

```
@Demo @AutomationPratice @PLP
Feature: Checkout Process
  As a Customer
  I want to be able to go through checkout process
  So that I can purchase a product in the online-store


  @Positive @PDP
  Scenario: Validate Price on PLP and PDP
    When I execute "Add Dress to Cart" reusable step
    And I wait for "3" seconds
    And I capture text data "Automationpractice Add to Cart Total
Price" as "EC_PRICE" variable
    And I click on the "Close window" "element by title"
    And I scroll the page by "0" horizontal and "300" vertical
    And I hover over the "Printed Summer Dress" "AutomationPractice
PLP Product Tile"
    And I click on the "Printed Summer Dress" "AutomationPractice
PLP More Button"
    And I wait for "3" seconds
    And I validate text "EC_PRICE" to be displayed for
"our_price_display" "element by id"

  @Positive
  Scenario: Validate Compare Functionality
    Given I am on "http://automationpractice.com/index.php" URL
    When I set "Dress" text to the "search_query" "element by name"
    And I click on the "submit_search" "element by name"
    And I hover over the "Printed Summer Dress" "AutomationPractice
PLP Product Tile"
    And I click on the "Printed Summer Dress" "AutomationPractice
PLP Add to Compare Button"
    And I hover over the "Printed Dress" "AutomationPractice PLP
Product Tile"
    And I click on the "Printed Dress" "AutomationPractice PLP Add
to Compare Button"
    And I wait for "2" seconds
    And I validate text "1" to be displayed for "Automationpractice
Compare Button Items" element
```

```
@Demo @AutomationPratice @ShoppingCart
Feature: Shopping Cart
  As a Customer
  I want to be able to add products to shopping cart
  So that I can purchase a product in the online-store

  @Positive @MiniCart
  Scenario: Validate Price in Mini Cart
    When I execute "Add Dress to Cart" reusable step
    And I wait for "3" seconds
    And I capture text data "Automationpractice Add to Cart Total
Price" as "EC_PRICE" variable
    And I click on the "Close window" "element by title"
    And I hover over the "Cart" button
    And I validate text "EC_PRICE" to be displayed for "Printed
Su..." "AutomationPractice Mini Cart Price"

  @Positive @MiniCart
  Scenario: Validate Price on the Shopping Cart Page
    When I execute "Add Dress to Cart" reusable step
    And I wait for "3" seconds
    And I capture text data "Automationpractice Add to Cart Total
Price" as "EC_PRICE" variable
    And I click on the "Close window" "element by title"
    And I click on the "Cart" button
    And I validate text "EC_PRICE" to be displayed for "Printed
Summer Dress" "AutomationPractice Shopping Cart Price"
```