МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

(повна назва)

Кафедра Програмної інженерії

(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів проектування автоматизованих тестів на основі BDD підходу

(тема)

Виконав:

Студент 2 курсу, групи ІПЗм-19-3

Безсмертний О.П

(прізвище, ініціали)

Спеціальність 121 Інженерія програмного забезпечення

(код і повна назва спеціальності)

Тип програми Освітньо-наукова

(освітньо-професійна або освітньо-наукова)

Керівник доц. Голян Н.В.

(посада, прізвище)

Допускається до захисту

Зав. кафедри _____ З.В. Дудар

(підпис) (прізвище, ініціали)

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 121 Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми Освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Інженерія програмного забезпечення
(повна назва)

ЗАТВЕРДЖУЮ:
Зав.кафедри _____
(підпис)
« 26 » березня 2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента Безсмертного Олександра Петровича
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів проектування автоматизованих тестів на основі BDD підходу

затверджена наказом університету від 26.03.2021 № 386 Ст

2. Термін подання роботи до екзаменаційної комісії 16 травня 2021р.

3. Вихідні дані до роботи *Використовувати середовище програмування Visual Studio 2019, мову програмування C#, фреймворк SpecFlow*

4. Перелік питань, що потрібно опрацювати в роботі *Вступ, аналіз предметної галузі, існуючі практики у роботі і BDD підходом, практичні правила BDD підходу, технічна імплементація, висновки, перелік джерел посилання.*

5. Перелік графічного матеріалу із зазначенням креслеників, схем, слайдів, ілюстрацій *Діаграма з описом принципів TDD, діаграма з описом принципів BDD, загальноприйняті практики BDD підходу, приклад тестового сценарію без табуляції кроків типу «And», приклад тестового сценарію з табуляцією кроків типу «And», приклад програмного коду де необхідна заміна*

списку використовуваних браузерів, приклад реалізації класу сторінки (моделі Page Object), приклад програмної реалізації методу пошуку веб-елемента на сторінці з певною кількість спроб, приклад програмної реалізації «загального» тестового кроку, приклад програмної реалізації порожнього тестового кроку, файл історії BDD з використання практик описаних у розділах 2 та 3.

6. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спцечастина	доц. Голян Н.В.		15.05.21

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання індивідуального завдання	25.01.2021	виконано
2	Аналіз предметної області	25.01.2021 – 01.02.2021	виконано
3	Постановка задачі	01.02.2021– 08.02.2021	виконано
4	Аналіз предметної галузі	08.02.2021 – 15.02.2021	виконано
5	Дослідження існуючих практик BDD підходу	15.02.2021 – 22.02.2021	виконано
6	Утворення нових правил використання BDD підходу	24.02.2021 – 05.03.2021	виконано
7	Написання програмної реалізації	19.03.2021 – 12.04.2021	виконано
8	Підготовка пояснювальної записки	21.04.2021 – 06.06.2021	виконано
9	Підготовка презентації та доповіді	06.06.2021 – 08.05.2021	виконано
10	Нормоконтроль	09.05.2021 – 18.05.2021	виконано
11	Рецензування	09.05.2021 – 18.05.2021	виконано
12	Занесення диплома в електронний архів	15.05.2021	виконано
13	Попередній захист	15.05.2021	виконано
14	Захист звіту	19.05.2021	виконано

Дата видачі завдання 25 січня 2021р.

Студент Безсмертний Олександр Петрович
(підпис)

Керівник роботи _____ доц. Голян Н.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Звіт з науково-дослідної практики: 61 с., 11 рис, 1 табл., 6 додатків, 20 джерел.

BEHAVIOR DRIVEN DEVELOPMENT, BDD, TEST DRIVEN DEVELOPMENT, TDD, GHERKIN, SPECFLOW, ПРАКТИКИ, ДОСЛІДЖЕННЯ, СЦЕНАРІЙ, ТЕСТУВАННЯ, АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, ФРЕЙМВОРК, КВАЛІФІКАЦІЙНА РОБОТА

Об’єкт дослідження - BDD-підхід в автоматизованому тестуванні.

Метою атестаційної роботи є покращення існуючих методів побудови автоматизованих тестів на основі BDD підходу, та побудова прототипу фреймворку для розробки та виконання цих тестів. Також у ході роботи будуть запропоновані та описані корисні практики використання BDD, відмінні від загальноприйнятих, котрі було почерпнуто під час використання підходу при розробці та контролю якості декількох проектів.

Метод рішення базується на використанні наступних технологій - VisualStudio 2019, мова C#, SpecFlow Framework. В результаті розробки сформульовано практики використання BDD підходу та створено фреймфорк для автоматизованого тестування програмних засобів з використанням BDD підходу.

BEHAVIOR DRIVEN DEVELOPMENT, BDD, TEST DRIVEN DEVELOPMENT, TDD, GHERKIN, SPECFLOW, PRACTICES, RESEARCH, SCENARIO, TESTING, AUTOMATED TESTING, FRAMEWORK, QUALIFICATION WORK

The object of research - BDD approach in the automation testing.

The purpose of the certification work is to improve the existing methods of building automated tests based on the BDD approach, and building a prototype framework for the development and implementation of these tests. The course will also suggest and describe useful practices for the use of BDD, different from the generally accepted ones, which were learned during the use of the approach in the development and quality control of several projects.

Solution method is based on the following technologies - VisualStudio 2019, C # language, SpecFlow Framework. As a result of the development, the practices of using the BDD approach were formulated and a framework for automated testing of software using the BDD approach was created.

Я, Безсмертний Олександр Петрович, студент гр. ІПЗм-19-3, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів проектування автоматизованих тестів на основі BDD підходу», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений (а) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

На сьогоднішній день розробка програмного забезпечення майже не можлива без його тестування. Все частіше контроль якості стає однім із ключових аспектів життєвого циклу розробки програмних продуктів. Особливо помітно увагу до якості у програмного забезпечення, що має довгий термін розробки чи відноситься до таких сфер, де ціна помилки є значною, наприклад, сфера охорони здоров'я, банківська справа.

При довгостроковій розробці ручне тестування стає неефективним , а тому використання автоматизованого тестування є дуже важливим для економії ресурсів. Існують різні методи розробки програмного забезпечення, зокрема Test Driven Development (TDD) та Behavior Driven Development (BDD) [1], які ставлять тестування на ключову позицію.

BDD підхід є навід'ємною частиною розробки сучасних програмних продуктів і на сьогодні існує безліч проектів, що його використовують. Але незважаючи на це, офіційна документація та керівництва, створені співтовариством користувачів, є дуже базовими і зазвичай застерігають чого не потрібно робити замість того, щоб пояснювати як це робити правильно [2].

Існує уже декілька досліджень які показують, що BDD підхід є доволі ефективним у тестуванні різних типів програмного забезпечення, наприклад: веб-систем [3], інтерфейсів програмного забезпечення та інших. Також воно є ефективним і використовується у різних типах тестування, таких як модульне тестування, приймальне тестування, тестування продуктивності програмного засобу та інтеграційне тестування.

Метою атестаційної роботи є покращення існуючих методів побудови автоматизованих тестів на основі BDD підходу, та побудова прототипу фреймворку для розробки та виконання цих тестів. Також у ході роботи будуть запропоновані та описані корисні практики використання BDD, відмінні від загальноприйнятих, котрі

було почерпнуто під час використання підходу при розробці та контролю якості декількох проектів.

Об’єктом дослідження є принципи правильного та продуктивного використання BDD підходу у автоматизованому та ручному тестуванні програмного забезпечення.

Предмет дослідження – існуючі загальноприйняті практики використання BDD підходу.

Методами дослідження є аналіз існуючих методів побудови автоматизованих тестів з використанням BDD, опис їх переваг та недоліків, представлення нових практик. Також одним із методів дослідження є розробка робочого прототипу фреймворку за допомогою обраних засобів – мови програмування C# та фреймворку SpecFlow [4].

Результат проведеного дослідження може бути використаний для розробки ефективних та зрозумілих автоматизованих тестів основі BDD підходу для веб-систем, програмних інтерфейсів користувача, трансформацій даних, хмарних застосунків та інших типів програмних продуктів, для написання правильних тестових сценаріїв, зручних у документуванні та перевірці аудиторами програмного забезпечення.

Далі прототип фреймворку може бути використаним для впровадження зручного і ефективного автоматизованого тестування для майже будь-яких програмних проектів з використанням різних типів тестування із незначними доробками.

За результатами кваліфікаційної роботи магістра було розроблено презентацію (див. додаток В).

Було опубліковано статтю «Behavior Driven Development Approach in the Modern Quality Control Process» в рамах науково-технічної конференції «Problem of Infocommunications. Science and Technolpgy (PIC S&T′2020)». Матеріали статті наведено в додатку Д.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Програмні застосунки сьогодні вже є невід’ємною частиною людського життя, вони проникли у всі його сфери та використовуються майже всіма, адже комп’ютеризація охоплює все нові аспекти життєдіяльності. Така тенденція продовжується уже достатній час та не збавляє оберти.

Таким чином розробка програмного забезпечення поступово стала дійсно важливою, на неї звернули увагу і вона формалізувалася та збагачувалася стандартами, керівництвами, правилами. І якщо раніше розробка могла вестися без повноцінного тестування, то зараз цей її аспект є ледь не найголовнішим серед усього життєвого циклу, на ряду із проектуванням. Адже відсутність достатнього рівня контролю якості на програмному проекту може призвести до значних наслідків. Звісно, незначна помилка у роботі інтернет магазину якщо і завдасть комусь шкоди, то цьому можна буде не надавати великого значення, але якщо, наприклад, виникне помилка у програмному забезпеченні системи контролю за польотами, то це все будуть зовсім інші наслідки. Це все веде до того, що зі зростанням покриття галузей програмними продуктами, зростає і можлива ціна його помилки, що в свою чергу підтверджує необхідність його валідації.

Із розвитком засобів та методів розробки розвивалися і підходи до тестування ПЗ. Спочатку розвивалося мануальне тестування. З’являлися техніки які дозволяли робити все ефективніше та продуктивніше. Але з часом стало зрозуміло, що мануальне тестування обмежене швидкістю роботи людини, її уважністю та звичайним людським фактором. І рішенням стала автоматизація цього процесу, як і у будь-якій іншій сфері діяльності. Автоматизоване тестування ввібрало в себе процеси мануального і також розвивало та оптимізувало власні методи. Автоматизація допомогла зменшити необхідну кількість людських ресурсів при збільшенні потреби у кваліфікації та кількості необхідного часу на розробку тестів. Саме цей аспект і по сей день є одною із причин частого уникання автоматизованого тестування на малих

проектах, розрахованих на швидку розробку і випуск, без подальшого супроводу. Натомість для проектів, розрахованих на довгу розробку та довгостроковий супровід (декілька версій), автоматизацій контролю якості є найкращим рішенням через те, що вона дає виграш лише з часом, але цей виграш того вартий.

Behavior Driven Development або ж просто BDD є одним із підходів у тестуванні та розробці програмного забезпеченні в цілому. Це підхід у якому при розробці поведінка функціоналу грає основну роль [5]. BDD було представлено у 2006 році Даніелем Нортом. Цей підхід є розширеною версією іншої техніки, а саме Test Driven Development або скорочено TDD [6] і вони обидва є орієнтованими на контролі якості. Для написання BDD сценаріїв використовується читабельний текст написаний мовою домену (domain-specific language- DSL) [7] – Gherkin [8]. BDD - це не нова річ у тестуванні, але навіть незважаючи на це, більшість застарілих проектів далекі від стандартів цього підходу, тому він здебільшого використовується в автоматизованому тестуванні [9] нових проектів, рідше це використовується у застарілих довгострокових проектах як еволюційний перехід від старих засобів. Він також може бути використаний для ручного тестування [10].

Найпоширеніша помилка людей, які не беруть участь у тестуванні, полягає в тому, що основним завданням процесу контролю якості є пошук помилок та помилок у розробленому проекті.

З одного боку, це правильне твердження, але лише частково. Завдання процесів забезпечення якості полягає в тому, щоб перевірити, чи протестований проект відповідає вимогам, а потім підготувати та надати документацію, яка це підтверджує. Крім того BDD сценарії, реалізовані відповідно до цього підходу, уже можна називати документацією, наприклад, документація попередньо виконаних тестових сценаріїв. Єдине, що потрібно, це написати сценарії BDD, дотримуючись усіх найкращих практик, та додати деякі метадані для доповнення сценаріїв поведінки:

- посилання на історії користувача;
- посилання на особливості;
- посилання на дефекти;
- номери версій;

– унікальні ідентифікатори;

– посилання на документацію розробників (наприклад, проектна документація низького рівня);

– посилання на вимоги (наприклад, проектна документація вимог або специфікація вимог користувача);

– посилання на ризики.

Крім того, потрібно створити інструмент, який буде аналізувати метадані та поєднувати їх із сценаріями. Цей метод замінить частину необхідної документації сценаріями, оскільки вони є зручними для читання та зрозумілими для аудиторів. Це також збільшує можливість перегляду сценаріїв тестування не залученими людьми (бізнес-аналітиками, розробниками, менеджерами, аудиторами, замовниками, власниками продуктів), оскільки читати такі сценарії легше, ніж інші типи тестів.

1.2 Аналіз існуючих рішень

Для того, щоб краще зрозуміти, що з себе мають техніки орієнтовані на поведінці спочатку необхідно проаналізувати аналоги.

Одним із таких аналогів є підхід Test Driven Development. Головну ідею можна описати наступним циклом:

– спочатку пишуться тести, що покривають бажані зміни чи новий функціонал, при цьому вони не виконуються через відсутність необхідної реалізації;

– розробники пишуть програмний код, що дозволить тестам вдало виконуватися;

– тести доробляються, покращуються, приводяться до остаточного вигляду і далі переходимо до першого пункту.

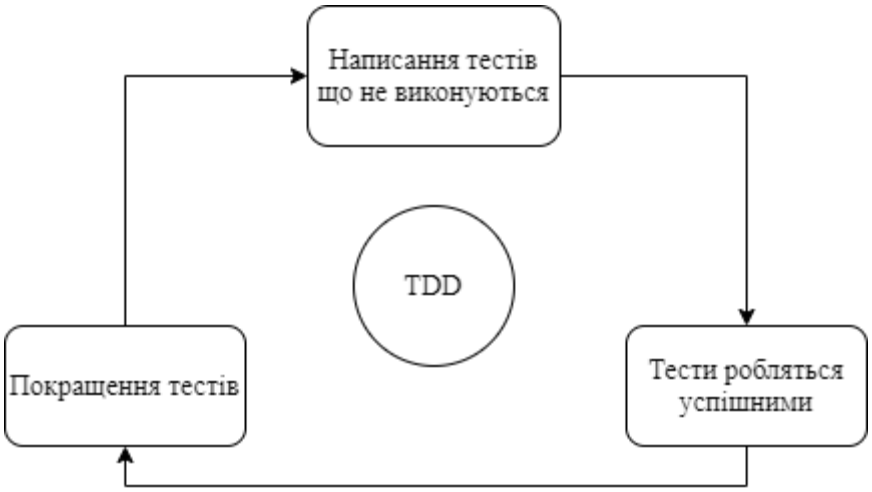

Рисунок 1.1 - Діаграма з описом принципів TDD

На рисунку 1.1 зображено скорочений варіант циклу розробки орієнтованої на тестуванні.

У свою чергу BDD має схожу послідовність дій яка має значну відміну у тому що замість написання не успішних тестів для нового функціоналу потрібно написати не виконувану специфікацію.

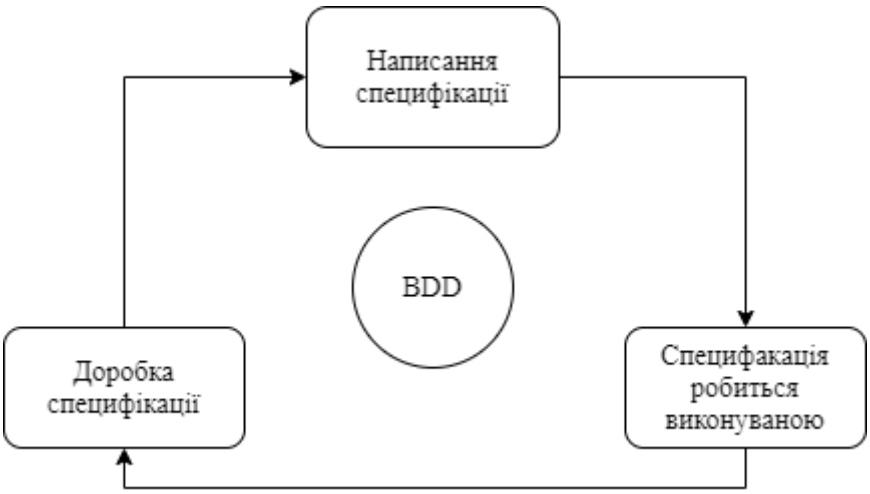

Рисунок 1.2 - Діаграма з описом принципів BDD

На рисунку 1.2 наведено діаграму схожу до TDD для наглядного порівняння фундаментальних принципів, що відрізняють ці техніки один від одного. Звісно різниця не лише у цьому, але це головне, що їх розрізняє.

Наступним аналогом є Domain Driven Design або скорочено DDD [11]. Це підхід у розробці програмного забезпечення метою якого є робота з проектами з достатньо складними бізнес правилами, де дуже необхідна взаємодія між розробниками та експертами в тій чи іншій галузі, до якої проект належить. Це потрібно для того, щоб встановити спільну мову та модель галузі, котра потім може бути трансформована у детальну документацію із вимогами до продукту.

BDD вбирає в себе особливості своїх аналогів, та поєднує їх у достатньо гармонічну комбінацію. Таким чином виходить підхід де тестування грає важливу роль на проекті, при цьому люди, котрі розробляють специфікації мають достатню експертизу у цільовій сфері бізнесу.

Далі наведено переваги BDD серед аналогів:

– низький поріг входження – це одна із головних переваг принципів BDD. Для написання тестових сценаріїв не потрібні технічні навички, адже сценарії написані читабельною мовою, а значить це може зробити навіть бізнес аналітик. Далі тестувальник або розробник доповнить сценарії реалізацією за допомогою тої чи іншої мови програмування;

– простота генерації тестової документації. Для того, щоб отримати гарну документацію необхідно лише доповнити сценарії деякими мета даними, адже сценарії уже написані у форматі, необхідному для документування;

– зрозумілість тестів для не залучених до проекту осіб. Ця перевага більш характерна для автоматизованих тестів, де зазвичай важко зрозуміти тестові сценарії комусь окрім команди розробки.

Усі переваги так чи інакше походять від застосування мови домену для написання тестів. Ця мова ідеально підходить для бізнес людей, а з іншого боку достатньо зрозуміла і зручна у роботі для технічних експертів. Це і є причиною того чому BDD підхід є ідеальним поєднанням технічних та бізнес аспектів процесів тестування.

1.3 Постановка задачі

Мета роботи досягається рішенням наступних задач:

– дослідження існуючих підходів до тестування програмного забезпечення, зокрема існуючих практик роботи з підходом що базується на поведінці (Behavior Driven Development);

– опис нових методів роботи з даним процесом на основі проведеного дослідження;

– розробка фреймворка автоматизованого тестування програмних засобів із користувацьким інтерфейсом, як приклад використання результатів проведеного дослідження.

У результаті роботи запропоновано нові практики для розробки тестових BDD сценаріїв, їх виконання, та розробки фреймворку для їх реалізації. Для демонстрації дослідження реалізовано фреймворк для автоматизованого тестування веб-систем з користувацьким інтерфейсом з використанням BDD підходу.

2 ІСНУЮЧІ ПРАКТИКИ У РОБОТІ З BDD ПІДХОДОМ

2.1 Детальніший опис структури BDD

Для подальшого розуміння необхідно зробити короткий екскурс у особливості структури BDD сценаріїв та мови Gherkin.

Тести написані з використанням BDD підходу є сценаріями, які у свою чергу складаються із назви сценарія та кроків. Назва описує суть тесту, та зазвичай є стислим переказом кроків. Кроки, у свою чергу описують послідовність дій, від початку до перевірки результату. Сценарії об’єднуються певною історієї, яка має відповідну коротку назву, та певну історію користувача, яка їй відповідає.

Щодо мови Gherkin, то вона не дуже відрізняється від звичайної, яку ми використовуємо для спілкування (наприклад, англійської), з тою лише різницею, що має певні ключові слова, які допомагають описувати тестові сценарії. Наведемо список необхідних для розуміння матеріалу ключових слів:

– Feature (Story)/Історія – позначає окрему історію, після ключового слова йде її назва, а з наступного рядка пишеться історія користувача, що її характеризує. Далі йдуть сценарії;

– Scenario/Сценарій – позначає окремий сценарій, після цього слова через пробіл йде його назва, а далі йдуть кроки з нового рядка, кожен крок у окремому рядку і так до нового ключового слова Scenario;

– Given/Дано – ключове слово, що позначає певний тип тестових кроків, що відповідає за умови сценарію;

– When/Коли – ключове слово, що позначає певний тип тестових кроків, що відповідає за дію сценарію;

– Then/Тоді – ключове слово, що позначає певний тип тестових кроків, що відповідає за результат сценарію;

– And/I – заміняє останній попередній тип кроку.

2.2 Загальноприйняті практики роботи з BDD підходом

Behavior Driven Development підхід існує уже достатній час та використовується на багатьох проектах. Беручи це до уваги, можна зробити висновок, що повинно існувати досить багато різних джерел інформації стосовно нього. Зокрема різних керівництв, створених співтовариством користувачів як самого BDD підходу, так і фреймворків для автоматизованого тестування, що його використовують.

Згідно з офіційною документацією найпоширеніших фреймворків BDD (Cucumber та SpecFlow [2]) та керівництвом щодо синтаксису мови Gherkin [6], можна виділити основні задокументовані правила та практики використання цього підходу, що зображено на рисунку 2.1.

Рисунок 2.1 – Загальноприйняті практики BDD підходу

Як видно із рисунку 2.1 існує 4 основні практики, що зустрічаються у офіційних та користувацьких джерелах.

Далі наведено більш детальне описання кожного із пунктів:

– порядок тестових кроків – кроки "Given", "When", "Then" слід використовувати лише у такому порядку. Це поширена помилка, коли сценарій містить кілька комбінацій «When - Then», це показує, що сценарій написаний

неправильно, і одне з найбільш вірогідних рішень - розділити його на кілька незалежних сценаріїв, які перевірятимуть окремі функціональні можливості;

– використання теперішнього часу – сценарії та кроки слід писати лише у теперішньому часі. Поширеною помилкою є використання минулого часу для кроків типу «Given», а майбутнього часу для кроків «Then», маючи на увазі, що перший є передумовою минулого, а другий - результатом у майбутньому. Насправді сценарій - це опис поточного покрокового процесу;

– повторне використання кроків – реалізація тестових кроків має бути атомарною та придатною до повторного використання. Це означає, що кожен крок повинен виконувати одну дію низького рівня та мати параметри, щоб мати можливість виконувати одні й ті ж самі дії по-різному. Наприклад, крок для вибору конкретного елемента зі списку: "When the user selects the <element name> element from the list";

– точне призначення кожного типу кроків – кроки "Given" повинні описувати контекст або передумову, кроки "When" повинні описувати дію, а кроки "Then" повинні описувати результат, який перевіряється. Достатньо поширеною помилкою є використання кроку «When» для опису передумови, а не дії у сценарії, його слід перетворити на крок «Given». Крім того, поширена помилка, коли крок “When” не використовується взагалі, а дію у сценарії замість нього описує крок типу “Given”.

Проблема описаних практик у тому, що вони досить базові, та не надають достатньо інформації для підвищення навичок у роботі із BDD підходом. Звісно, ці офіційні практики дуже важливі, і їх дотримання є обов’язковим. Однак здебільшого цього недостатньо, особливо коли контроль якості проводиться для галузей з високим рівнем контролю та перевірок, таких як фармацевтична, медична, харчова тощо. Аудиторські організації можуть відхилити тестову документацію (і весь проект) через грубість документації, файли історій BBD є частиною цієї документації [2].

3 ПРАКТИЧНІ ПРАВИЛА BDD ПІДХОДУ

3.1 Загальний опис запропонованих практик

Для доповнення методів роботи з Behavior Driven Development підходом будуть використовуватися знання, отримані у процесі роботи з ним на різних проектах. Вони є вибіркою того, що було тим чи іншим чином прийнято як стандарт для роботи учасниками розробки проектів.

Далі буде наведено лише ті практики, що варто використовувати на усіх проектах на відміну від тих порад, що можуть бути корисними, але є дуже специфічними для окремих проектів і малоймовірно, що зможуть прижитися у іншому проекті. До таких практик відносяться наступні:

- використання оптимального шаблону для формування назви сценаріїв;
- обмеження на кількість тестових кроків;
- написання «загальних» кроків;
- використання правильної точки зору для опису сценаріїв;
- обмеження на кількість дій у сценарії;
- правильне використання кроків типу «Given»;
- табуляція кроків типу «And».

Далі приведений детальний опис кожної із запропонованих практик.

3.2 Використання оптимального шаблону для формування назви сценаріїв

Суть цієї практики полягає у використанні певного шаблону для написання назви сценаріїв. Після роботи над кількома проектами було вирішено, що шаблон "Коли ..., тоді ..." є кращим для імен сценаріїв BDD як частина документації. Наприклад наступна назва, “Коли користувач входить, тоді відображається головна

сторінка веб-додатку”. У таких випадках заголовок описує дію та результат, і навіть не потрібно дивитися на назви кроків.

Використовуючи цей шаблон, важливо відстежувати довжину заголовка, оскільки перш за все він повинен бути зручним для читання. Не належне використання цього шаблону може збільшити довжину назви сценарію, що призведе то того, що він буде занадто великим, щоб помістится на одній лінії екрану без перенесення.

Ця практика є однією із простіших для використання, але при цьому відчутно покращує розуміння сценарії, а з часом, після тривалого використання, також підвищує ефективність написання нових сценаріїв, адже дотримання одного шаблону допомагає швидше придумувати назви, змінюючи лише частину заголовку на необхідну для нового тесту інформацію.

3.3 Обмеження на кількість тестових кроків

Ця практика полягає у важливості обмеження кількості кроків у одному сценарії, а також у обмеженні кількості сценаріїв на одну історію. Це допоможе спростити перегляд файлів історій, особливо для не залучених людей, котрі будуть проглядати сценарії з метою аудиту. На основі робочого досвіду було встановлено, що ліміт у 10 тестових кроків є достатнім для гарного опису сценарію. Для кількості сценаріїв точних обмежень встановити неможливо, адже тут більш ситуативна річ, іноді велика кількість тестових сценаріїв у одній історії це необхідність.

Ця практика є більш опціональною, адже навіть обмеження у 10 кроків це не завжди добре. Хоча в більшості випадків це є ознакою, що сценарій перевантажений діями, або просто дії описані не дуже влучно і варто їх об`єднати, щоб покращити його читабельність.

3.4 Написання «загальних» кроків

Практика полягає у знаходженні золотої середини між створенням «загальних» та специфічних, зручних для розуміння, тестових кроків. Згідно кращих практик використання BDD тестові кроки повинні бути придатними для повторного використання, найкращий спосіб досягти цього – це створювати «загальні» кроки з параметрами, наприклад, «Коли користувач натискає на ‘<назва елементу>’ на сторінці ‘<назва сторінки>’». Подібний крок може бути використаним для виконання будь-якого натискання на будь-який елемент на певній веб-сторінці і тому виглядає досить зручним.

Після певного дослідження було виявлено, що тестові кроки такого зразку дуже зручні у використанні, але їх краще поєднувати з певними специфічними кроками, наприклад, «Коли користувач натискає на посилання Персонального кабінету». Такі кроки є більш зручними для читача, особливо для того, у якого немає досвіду розробки подібних сценаріїв. Також подібні кроки мають перевагу, яка походить від їх недоліку, а саме від того, що вони слабо придатні для повторного використання у інших сценаріях через відсутність параметрів, але це надає змогу задати кожному такому кроку свою логіку роботи, при реалізації програмного коду. Це є перевагою у системах, де можлива різна поведінка при виконанні схожих операцій, адже при реалізації «загального» кроку доведеться додавати певну кількість умов, що зробить такий крок досить громіздким і ускладнить подальший його супровід.

3.5 Використання правильної точки зору для опису сценаріїв

Суть практики полягає у використання точку зору «від третього обличчя» для написання назв сценаріїв та тестових кроків. У результаті дослідження було вирішено використовувати цю точку зору для опису дій користувача у сценарії. Наприклад,

писати «Коли користувач вводить дані у поле ‘ім`я користувача’» замість того щоб писати «Коли я ввожу дані у поле ‘ім`я користувача’». Така практика має свої переваги. По-перше, це звільняє сценарій від персоналізації і допомагає не залученим людям краще розуміти, про кого йде річ. По-друге, це дозволяє у будь-який момент замінити одного актора на іншого, без переробки сценаріїв, адже замість слова «користувач» може бути використаним інший іменник (наприклад, оператор, адміністратор, модератор, чи інші). Такий підхід робить BDD сценарії більш формальними, те технічними, та сприяє певному шаблону їх проектування, що допомагає у роботі різних людей над однією задачею.

3.6 Обмеження на кількість дій у сценарії

Практика стосується обмеження кількості дій у одному сценарії поведінки. Тестовий сценарій повинен мати лише один крок типу дії, а саме крок з ключовим словом «Коли» / «When». Якщо у сценарії більше одної дії то він вважається не коректним і потрібно розділити його на декілька сценаріїв або ж зайві дії варто винести у кроки підготовки з ключовим словом «Дано» / «Given». Зазвичай зайві дії це ознака того, що у сценарії намагаються встановити забагато перевірок, але це не відповідає принципам BDD. З іншого боку ці дії можуть бути просто не влучно названими передумовами і тоді дійсно варто перетворити її на кроки типу «Дано».

Іноді гарним рішенням такої проблеми є об`єднання декількох дій в одному тестовому кроці типу «Коли». Звісно, в такому разі необхідно йому дати назву, котра не буде описувати всі дії, а лише ключову, котра важлива для перевірки. Наприклад, якщо для того, щоб відправити форму реєстрації необхідно після натиску на кнопку відправки ще підтвердити дії у вікні, яке запитує про згоду, то у такому разі можна назвати крок наступним чином: «Коли користувач відправляє форму реєстрації», при цьому при програмній реалізації кроку виконувати не тільки натискання на кнопку відправки а і на підтвердження дії у вікні з запитом.

Ця практика допоможе інженерам контролю якості які звикли писати довгі тестові сценарії типу «дія – перевірка, дія – перевірка…» і не звикли писати тести використовуючи BDD підхід.

3.7 Правильне використання кроків типу «Given»

Дана практика пояснює як правильно використовувати тестові кроки типу «Given». Основне призначення кроків цього типу це підготовка певних умов для успішного виконання дії що перевіряється, наприклад, налаштування оточення чи приведення продукту у деякий необхідний стан.

Іноді для виконання дії що перевіряється додаткова підготовка не потребується, тому що програма уже знаходиться у необхідному стані з самого початку, наприклад, якщо йдеться про перевірку якоїсь початкової точки інтерфейсу яка завжди йде першою у списку дій користувача. У такому разі необхідно вказати цей початковий стан як передумову, адже тоді у сценарії буде відсутній крок типу «Given», а це не дуже добре у розрізі BDD підходу, хоча за стандартами звичайних тестів такий випадок є нормою.

Наприклад, уявимо, що є деякий програмний продукт з певними налаштуваннями мови користувацького інтерфейсу. У цього налаштування є значення за замовчуванням, але у вимогах до проекту не зазначено яка мова має бути за замовчуванням при розгортанні додатку і команда розробки встановлює англійську мову на свій вибір. Припустимо що програмне рішення має три мови налаштування (Англійську, Українську, Німецьку) і це вказано у вимогах, тоді нам необхідно створити мінімум троє тестових сценаріїв для перевірок відображення локалізації. У результаті отримуємо такий сценарій: «Коли користувач відкриває Головну сторінку, тоді текст відображається Англійською мовою» з наступними тестовими роками:

- коли користувач натискає на посилання Головна;
- тоді текст на сторінці Головна відображається Англійською мовою.

Такий сценарій перевірки мовних налаштувань буде працювати коректно, проте він буде менш інформативним для не залучених людей. Тому необхідно створити крок с наступною назвою: «Дано що встановлено Англійські мовні налаштування». З технічної точки зору (програмної реалізації тестового кроку) цей крок не дуже необхідний, тому при реалізації програмного методу для нього можна залишити його пустім з відповідним коментарем, який зазначить що цей метод не просто так пустий і що він є необхідним. Після усіх цих маніпуляцій тестовий сценарій виглядатиме наступним чином:

- дано що встановлено Англійські мовні налаштування;
- коли користувач натискає на посилання Головна;
- тоді текст на сторінці Головна відображається Англійською мовою.

3.8 Табуляція кроків типу «And»

Ця практика цілком і повністю стосується зручності візуального сприйняття тестових сценаріїв, особливо кимось не залученим у проект, або і зовсім кимось, хто є далеким від сфери розробки або контролю якості програмного забезпечення і просто переглядає сценарії з метою аудиту.

Як зазначалося раніше, кроки типу «And» використовуються для заміни ключових слів інших кроків («Given / When / Then») при їх послідовному використанні.

Також із попередніх практик виходить, що хоча «And» і можна використовувати для кроків типу «Коли» / «When», але цього не варто робити через особливість кроків пов`язаних із діями для збереження принципу «одна дія – один сценарій».

Що стосується відсутності табуляції, то далі на рисунку 3.1 наведено невеликий приклад кроків тестового сценарію.

```
Given the application is available
And the database is availabe
And test data is imported to the datbase
And the user is authorized
When the user send the request to the web API
Then the response code is 200
And the contract of the response model is valid
And the response boby contains the list of entities
```

Рисунок 3.1 – Приклад тестового сценарію без табуляції кроків типу «And»

Як видно із рисунка 3.1 при збільшенні кроків з трьох до дев`яти вже важко з першого погляду зрозуміти структуру сценарію, не кажучи про його суть. Таких сценаріїв буде набагато більше ніж один, а тому і складність сприйняття значно зросте.

Далі на рисунку 3.2 наведено приклад того ж самого тестового сценарію, але уже з використанням табуляції при його написанні.

```
Given the application is available
    And the database is availabe
    And test data is imported to the datbase
    And the user is authorized
When the user send the request to the web API
Then the response code is 200
    And the contract of the response model is valid
    And the response boby contains the list of entities
```

Рисунок 3.2 – Приклад тестового сценарію з табуляцією кроків типу «And»

На прикладі рисунків 3.1 та 3.2 одразу видно різницю при використанні такої незначної функції будь яких текстових редакторів як відступ або ж табуляція. Ця дрібниці значною мірою покращить розуміння тестового сценарію та в першу чергу одразу дасть змогу візуально побачити три групи кроків які відносяться до трьох основних типів.

4 ТЕХНІЧНА ІМПЛЕМЕНТАЦІЯ

4.1 Загальні відомості про програмну реалізацію

У результаті атестаційної роботи було розроблено програмну реалізацію фреймворку автоматизованого тестування з використанням BDD підходу, як інструмент демонстрації результатів проведених досліджень. Фреймворк розроблений для тестування програмних засобів із візуальним користувацьким інтерфейсом [3], хоча досліджувані практики та особливості реалізації можна використовувати для побудови фреймворків і для інших типів додатків, таких як тестування API, веб-функцій, трансформацій даних та баз даних.

Програмна реалізація несе у собі як практичне використання описаних практик, так і внесення нових практик, що не стосуються BDD напряму, але є корисними рішеннями при роботі із автоматизованим тестуванням в цілому. Адже BDD це лише спосіб у автоматизованому тестуванні, котрий використовується для підвищення його якості та зручності, інші частини схожі на звичайні структурні ланки звичайних фреймворків автоматизованого тестування але пристосовані для того щоб доповнювати файли історій BDD.

Фреймворк розроблено на мові C# сімейства .NET, де для інтеграції BDD рішень використовується фреймворк SpecFlow який є аналогом Cucumber, розробленим для інших мов програмування (Ruby, Java, Python). Для роботи із веб-браузером у автоматизованому режимі використовується пакет застосунків Selenium, в тому числі ChromeWebDriver для керування браузером Google Chrome за допомогою програмного коду.

Далі у роботі буде наведено детальний опис наступних складових систем реалізації BDD фреймворку:

- компонент управління веб-браузером;
- компонент управління налаштуваннями;
- компонент керування сторінками моделі Page Object;
- компонент хуків;

- компонент управління тестовими кроками;
- файли бізнес історії BDD.

Кожен із наведених компонентів несе у собі функціонал, який можна вважати окремим і який може бути окремо використаний для побудови будь-якого іншого рішення для автоматизованого тестування після деякої адаптації під потреби іншого проекту.

Подібна програмна реалізація з використанням BDD може бути використана для різних типів тестування [12] як шаблон з деякими змінами під потреби того чи іншого виду тестування що наведено у таблиці 4.1.

Таблиця 4.1 – Використання шаблону тестового фреймворку з використанням BDD підходу у різних типах тестування

Тип тестів	Опис	Інтеграція з BDD
Модульне тестування	Це метод тестування програмного забезпечення, за допомогою якого тестуються окремі одиниці вихідного коду, набори одного або декількох комп'ютерних програмних модулів разом із відповідними даними управління, процедурами використання та операційними процедурами, щоб визначити, чи придатні вони для використання. Перевірка найменших частин тестованого програмного забезпечення в додатку, щоб визначити, чи поводяться вони належним чином.	Зазвичай модульним тестуванням займаються розробники проекту і BDD у цих тестах використовується не часто. Але така інтеграція іноді викликана потребами замовниками через зручність для нього у сприйнятті тестів написаних із використанням BDD підходу.

Продовження таблиці 4.1

Тип тестів	Опис	Інтеграція з BDD
Інтеграційне тестування	Це етап тестування програмного забезпечення, в якому окремі програмні модулі поєднуються та тестуються як група. Інтеграційне тестування проводиться для оцінки відповідності системи або компонента заданим функціональним вимогам.	При розробці інтеграційних тестів використання технік BDD це звичайна справа. Особливістю використання у цьому типі є те, що зазвичай інтеграційні тести є більш верхньорівневими і тому тестові кроки зазвичай включають в себе більше функціоналу при то му що їх назви звучать досить просто, адже тут необхідно виконати більше дій для досягнення простих бізнес результатів.
Приймальне тестування	Офіційне тестування з урахуванням потреб користувачів, вимог та бізнес-процесів, проведене для того, щоб визначити, чи відповідає система критеріям прийнятності, а також для того, щоб користувач, клієнти чи інша уповноважена особа могли визначити, чи приймати систему.	На цьому рівні зазвичай тести є більш деталізованими та перевіряють кожну частину функціоналу окремо, тому використання BDD для цього типу є найбільш продуктивним. При детальному описі сценаріїв та кроків з’являється велика їх різноманітність. BDD допомагає краще їх структурувати при збереженні максимальної

Кінець таблиці 4.1

Тип тестів	Опис	Інтеграція з BDD
		зрозумілості усіх назв, а також зручності використання тестових кроків при розробці тестувальниками. Звісно звичайна організація програмних методів також дає можливість параметризації тестів та створенню загальних кроків, але це погіршує їх зрозумілість, та ускладнює їх пошук при необхідності у розробці.
Тестування продуктивності	Це загальна практика тестування, що проводиться для визначення ефективності роботи системи з точки зору швидкості реагування та стабільності при певному навантаженні. Він також може служити для дослідження, вимірювання, перевірки або перевірки інших атрибутів якості системи, таких як масштабованість, надійність та використання ресурсів.	При використанні BDD підходу у тестах продуктивності варто зазначити, що спосіб використання схожий із інтеграційними тестами з деякими особливостями. Так при написанні кроків котрі включають в себе велику кількість дій їх назви будуть не дуже загальними а більш конкретними з конкретним описом вимірюваних величин та їх показників (часу завантаження, навантаження на систему, тощо).

4.2 Компонент управління веб-браузером

До цього компоненту включені усі класи, котрі пов`язані із керуванням браузером, шляхом застосування WebDriver (далі драйвер, або веб-драйвер) зокрема версії, розробленої для роботи із браузером Google Chrome.

Драйвер це виконувана програма і є два шляхи його включення до проекту:

– додати виконуваний файл до проекту та при підключенні вказувати шлях і потім оновлювати його вручну;

– додати до проекту пакет із репозиторію пакетів NuGet, котрий окрім виконуваного файлу також додасть деяку обгортку, що спростить доступ до нього, а також допоможе легко його оновити, адже з кожною новою версією браузера старий веб-драйвер стає не актуальним і не може використовуватися для роботи.

Для розробки будь-яких рішень не направлених на навчання розробці використовується другий варіант [13], адже використання репозиторію NuGet є стандартом .NET проектів.

Цей компонент складається із наступних ключових класів:

– DriverManager – клас включає в себе основні методи управління веб-драйвером, такі як створення драйвера з встановленням налаштувань та його зупинка. Це досить важлива частина адже якщо метод зупинки драйвера не буде використано, то при виконанні тестів є ймовірність порушити роботоздатність агента (фізичного чи віртуального комп’ютера) через те, що його оперативна пам’ять буде повністю зайнята запущеними вікнами браузера та веб-драйверами, адже вони з часом самі не вимикаються;

– DriverTypes – клас відповідає за встановлення відповідності між різними видами веб-браузерів, на яких розраховується виконувати тестування, та типами драйверів і типами їх налаштувань;

– DriverManagerFactory – клас відповідає за створення менеджера драйвера (DriverManager) для окремих типів веб-браузерів у залежності від того який тип браузера було обрано для поточного виконання тестового набору.

Для подальшого використання цього компоненту у іншому проекті необхідно лише внести зміни у частини що стосуються набору тестованих веб-браузерів, що зображено на рисунку 4.1.

```
/// <summary>
/// Create driver manager.
/// </summary>
/// <param name="driverType">Web driver type to be encapsulated into Driver Manager.</param>
/// <returns>Driver manager for chosen driver type.</returns>
– references
public DriverManager CreateDriverManager(DriverType driverType)
{
    switch (driverType)
    {
        case DriverType.GoogleChrome:
            {
                return new DriverManager(
                    DriverTypes.DriverTypesDictionary[DriverType.GoogleChrome],
                    this.configManager.DriverConfig);
            }

        case DriverType.Firefox:
            {
                return new DriverManager(
                    DriverTypes.DriverTypesDictionary[DriverType.Firefox],
                    this.configManager.DriverConfig);
            }

        case DriverType.InternetExplorer:
            {
                return new DriverManager(
                    DriverTypes.DriverTypesDictionary[DriverType.InternetExplorer],
                    this.configManager.DriverConfig);
            }

        default:
            {
                return new DriverManager(
                    DriverTypes.DriverTypesDictionary[DriverType.InternetExplorer],
                    this.configManager.DriverConfig);
            }
    }
}
```

Рисунок 4.1 – Приклад програмного коду де необхідна заміна списку використованих браузерів

Зазвичай список веб-браузерів які необхідно перевіряти визначається у вимогах до програмного продукту, зокрема його можна знайти у документі User Requirement Specification або його аналогах.

На деяких малих проектах такі конкретні вимоги можуть бути опущені, тому цей список буде сформовано командою розробки у процесі обговорень із замовником.

4.3 Компонент управління налаштуваннями

Цей компонент є одним із універсальних і може бути використаним як шаблон не лише у автоматизованому тестуванні програмних засобів з користувацьким інтерфейсом [14] а і в будь-яких інших типах, адже він не має ніякої прив’язки до інтерфейсу користувача.

Взагалі робота із налаштуваннями може здатися надлишковою, адже завжди де використовуються значення із файлу налаштувань через низку посередників набагато простіше вказати необхідне значення напряму. З одного боку це дійсно зручніше і не потребує додаткового часу на розробку керівників налаштуваннями. Але це правдиво лише для дуже малих рішень, де важко буде заплутатися у програмному коді. Звідси і виходить основна але не єдина перевага централізації налаштувань.

Ця перевага доволі очевидна і в ній немає нічого особливого, іншою ж перевагою подібної розробки є можливість зміни налаштувань без зміни програмного коду у репозиторії. Може здатися, що це незначна перевага, але у циклі розробки програмного забезпечення настає момент призупинення кодування і «заморозки» конкретної версії проекту. Далі змінювати код неможливо, але налаштування це якраз ті значення які дуже часто потребують змін у різних ситуаціях, наприклад якщо необхідно змінити одну базу даних на іншу, або змінити шлях до стороннього сервісу, чи найчастіший варіант у автоматизованому тестуванні – змінити паролі користувача для доступу тестів у систему. І саме використання єдиного файлу налаштувань дає змогу простої замі їх значень для подальшого використання без зміни програмного коду.

Подібні компоненти використовуються в усіх типах проектів (не тільки для автоматизованого тестування) у різних формах. Для цього фреймворку було обрано варіант з менеджером налаштувань та різними зчитувачами, які будуть надавати проекту значення із файлу налаштувань.

Цей компонент складається із наступних класів:

– ConfigurationManager – клас відповідає за централізований доступ до налаштувань і може використовуватися для ін’єкції залежностей, для подальшо використання;

– ConfigReader – клас є основним зчитувачем налаштувань і містить у собі методи для доступу до необхідних секцій файлу, а також методи для загально зчитування у разі необхідності використання налаштувань як єдиного об’єкту;

– Інші додаткові зчитувачі – класи які є подібними до ConfigReader, але призначені для зчитування окремих підрозділів файлу налаштувань, зазвичай необхідні при явному розподілі налаштувань на групи для подальшого зручного використання, щоб відокремити значення по різним не великим спискам замість одного централізованого і громіздкого. Також включають в себе функціонал по додатковій обробці значень налаштувань, їх трансформації чи об’єднання із іншими даними для подальшого зручного використання.

4.4 Компонент керування сторінками моделі Page Object

Компонент є програмною реалізацією принципу Page Object Model [15] у поєднанні із особливостями BDD підходу.

Зазвичай принцип Page Object Model реалізується наступним шляхом: для кожної сторінки веб-системи створюється клас, який включає в себе поля, що відповідають веб-елементам на веб-сторінці а також методи для роботи з цими елементами, наприклад натискання, чи ввід тексту. Далі при роботі автоматизованого тесту при переході на будь-яку сторінку виконується ініціалізація усіх полів, через пошук веб-елементів по їх локаторам, для подальшої роботи з ними [16].

Поточна реалізація має свої особливості, а саме:

– використання атрибуту [17] для присвоювання назв елементам для зручності використання цих назв як параметрів кроків у BDD сценарії;

– методи для роботи з веб-елементами винесено у окремі класи, для уникнення копіювання коду;

– пошук кожного веб-елементу на веб-сторінці не при її відкритті а за потребою, що зменшує час на пошук, а також допомагає уникнути зайвих помилок через нестабільність Selenium WebDriver. Це часта проблема драйверу, що іноді веб-елемент на сторінці може бути не знайденим, при його присутності, це пов’язано з різними факторами, одним із яких є швидкість загрузки сторінки. Таким чином при одночасному пошуку всіх елементів сторінки шанс отримати одну помилку значно зростає ніж при виконанні пошуку одного елементу за раз.

Таким чином отримуємо реалізаціє Page Object більш зручну у використанні в цілому а також придатну до використання у BBD кроках.

Цей компонент складається із наступних ключових класів:

– BasePage – базовий клас для усіх моделей Page Object. Відповідає за реалізацію спільної логіки усіх сторінок фреймворку;

– PageNaming – клас утворює словник відповідності назв веб-сторінок до відповідних класів у проекті, для подальшого використання у якості параметрів кроків. Для зручності майже завжди назви класів сторінок і назви веб-сторінок, що використовуються у сценаріях співпадають, але іноді це не можливо;

– клас сторінки – клас наслідує базовий та містить у собі поля, що відповідають певним елементам на веб-сторінці;

– ElementConverter – клас відповідає за перетворення назв записаних у рядках в повноцінні об’єкти веб-елементів (WebElement) за їх атрибутом ElementName. Це надає змогу використовувати назву та тим елементу у BDD сценаріях як рядковий параметр, що дозволяє створювати загальні параметризовані кроки та слідувати принципам BDD;

– ElementNameAttribute – це реалізація атрибуту ElementName що використовується з полями класів для призначення рядкових назв елементам моделі Page Object.

Далі на рисунку 4.2 наведено приклад класу сторінки SearchPage:

```
public class SearchPage : BasePage
{
    – references
    public SearchPage(IWebDriver driver, RetryPolicyFactory retryPolicyFactory) : base(driver, retryPolicyFactory)
    {
    }

    – references
    public override By IFrameLocator => null;

    [ElementName(@"Search field")]
    – references
    public IWebElement SearchField => this.WrappedDriver
        .FindElementWithRetry(By.Id("sb_form_q"), this.RetryPolicyFactory);

    [ElementName(@"Search button")]
    – references
    public IWebElement SearchButton => this.WrappedDriver
        .FindElementWithRetry(By.CssSelector(".search.icon.tooltip"), this.RetryPolicyFactory);

    [ElementName(@"Search Suggestion elements")]
    – references
    public IReadOnlyCollection<IWebElement> SerachSuggestionElements => this.WrappedDriver
        .FindElementsWithRetry(By.CssSelector("li.sa_sg"), this.RetryPolicyFactory);
}
```

Рисунок 4.2 – Приклад реалізації класу сторінки (моделі Page Object)

З рисунку 4.2 видно, що кожне поле має атрибут ElementName який вказує рядкову назву, це необхідно, адже якщо використовувати назву поля, то через те що відповідно до синтаксису мови C# та будь-якої іншої мови програмування, назва поля не може розриватися пробілом. Назви написані таким чином знижують ефект підходу BDD який стосується покращення зручності читанням тестових сценаріїв. Також на рисунку видно що кожне поле здійснює пошук за заданим локатором лише при зчитуванні цього поля.

4.5 Компонент хуків

Хук – це метод, виконується перед чи після тестового сценарію та не враховується у результаті [18]. Тобто це метод який виконує певні дії, але потім вони не відображаються у вихідній документації. Хуки реалізовані шляхом використання атрибутів NUnit на методах для позначення приналежності до тої чи іншої їх групи. SpecFlow в свою чергу має власні атрибути, які є обгортками над атрибутами NUnit та більш придатні для використання з BDD. Наприклад, особливі Before методи, а

саме: BeforeScenario та BeforeFeature що надають змогу запускати хуки перед конкретними тестовими сценаріями або ж перед BDD історіями.

У даній реалізації використовуються 2 види хуків:

– TestRunInitializer – містить у собі метод що виконується перед виконанням усього набору тестів і виконує самі базові налаштування, а саме ін’єкцію залежностей різних сервісів;

– ScenarioInitializer – містить методи що виконуються до та після кожного тестового сценарію. Зазвичай цей тип хуків відповідає за налаштування тестового оточення під конкретний тест. Наприклад, запуск веб-драйвера, навігація на базову веб-сторінку, очистка куків, налаштування моків [19], створення тестових даних у базах даних та відповідне очищення після виконання.

У представленій імплементації створено хуки лише для базових дій, необхідних для кожного сценарію, з плином часу у процесі розробки тестів колекція цих методів буде постійно зростати.

4.6 Компонент управління тестовими кроками

Компонент відповідає за головну частину тестового фреймворку – за реалізацію тестових кроків BDD сценаріїв.

Співвідношенням програмних методів і рядкових назв кроків займається фреймворк SpecFlow, нам же необхідно наповнити методи функціоналом що відповідає назві та меті тестового кроку.

Даний компонент складається із наступних класів:

– BaseSteps – це базовий клас, що включає в себе реалізацію усіх методів та налаштувань, що будуть необхідні у більшості тестових кроків, зокрема метод для пошуку об’єкта веб-сторінки за її назвою, та подальшого збереження для використання об’єкту різними методами;

– CommonSteps – реалізує загальні тестові, кроки з обширною параметризацією. Зазвичай це кроки пов’язані із тривіальними діями, які можуть бути необхідними у будь-якому місці тестового сценарію;

– класи імплементації кроків – класи що реалізують більш специфічні тестові кроки із конкретними задачами, зазвичай пов’язуються із певною веб-сторінкою, або із певною частиною функціоналу програмного продукту, для якого виконується тестування;

– WebDriverExtensions – включає в себе «обгортки» над методами Selenium (FindElementBy(), Click(), SendKeys(), тощо) для більш точного їх налаштуванням під потреби проекту. Наприклад метод який чекає доки елемент зникне зі сторінки, або метод який шукає елемент на сторінці з певною кількість спроб, приклад його реалізації зображено на рисунку 4.3.

```
public static IWebElement FindElementWithRetry(this IWebDriver driver, By by, RetryPolicyFactory policyFactory)
{
    return policyFactory.CreateConfiguredRetryPolicy<NoSuchElementException, IWebElement>(
         (elem) => !elem.Displayed)
        .Execute(() =>
        {
            var result = driver.FindElement(by);
            return result;
        });
}
```

Рисунок 4.3 – Приклад програмної реалізації методу пошуку веб-елемента на сторінці з певною кількість спроб

На рисунку 4.3 видно що звичайний метод FindElement() покривається політикою повтору за допомогою RetryPolicyFactory [20], що дозволяє здійснювати пошук за декілька спроб з певним заданим часом очікування між ними. Це допомагає зменшити вірогідність помилки через невдачу Selenium WebDriver при виконанні пошуку веб-елементу на сторінці.

Згідно практики «Написання «загальних» кроків», що описується у розділі 3.4 одною із переваг BDD фреймворку є написання цих «загальних» тестових кроків і на рисунку 4.4 зображено приклад реалізації такого.

```
[When(@"the user clicks on the ""(.*)"" (field|button) on the ""(.*)"" page")]
– references
public async Task WhenTheUserClicksOnTheButtonOnThePage(string elementName, string elementType, string pageName)
{
    var element = $"{elementName} {elementType ?? string.Empty}".ConvertToElement(this.GetPageInstance(pageName));
    element.Click();
    await Task.Delay(1000);
}
```

Рисунок 4.4 – Приклад програмної реалізації «загального» тестового кроку

На рисунку видно, що для зв’язку методу із рядковою назвою кроку використовується атрибут When, що відповідає типу кроку. Атрибут має регулярний вираз, що і дозволяє йому співвідноситися із всіма кроками що підходять, а також допомагає зчитати значення його параметрів.

У практиці «Правильне використання кроків типу «Given», що описується детальніше у розділі 3.7, йдеться про обов’язкове використання кроків типу «Дано» у сценарії для встановлення передумови сценарію, навіть якщо програмна реалізація у такому разі буде відсутня, приклад чого зображено на рисунку 4.5.

```
[Given(@"the application is available")]
– references
public void GivenTheApplicationIsAvailable()
{
    // This step is used for the readability purpose only
}
```

Рисунок 4.5 – Приклад програмної реалізації порожнього тестового кроку

На рисунку 4.5 видно що він містить у собі коментар у якому йдеться про те, що цей крок існує лише з метою покращення читабельності. Такий коментар є необхідним, для того, щоб інші тестувальники знали про це та не видалили тестовий крок, або не почали б його реалізовувати.

При виконані тестового сценарію такий крок буде записано у результаті як виконаний, незважаючи а те, що він не має нічого що можна виконати. Що і потрібно для подальшої документації.

4.7 Файли бізнес історії BDD

Файли історій BDD віднесені це дійсно окремий компонент, тому що якщо взяти і винести його із усього фреймворку, то ці файли зможуть цілком і повність функціонувати і окремо. Звісно їх не можна буде виконати автоматизовано, але це все ще будуть тестові сценарії, готові до ручних перевірок програмного проекту.

На рисунку 4.6 зображено приклад файлу історії, з дотриманням усіх практик, що описуються у розділах 2 та 3.

```
@F1234
Feature: 1. Bing Search
    As an simple user
    I want to use the Bing to search through the Internet
    So the Bing serach should work properly

@US12345_Version_1.0.0
@Scenario_1.1
@UserReq_URS001
Scenario: 1.1. When the user opens the Bing search page, then the search field and button are displayed
    Given the application is available
    When the user opens the Search page
    Then the "Search" field is displayed on the "Search" page
        And the "Search" button is displayed on the "Search" page

@US12345_Version_1.0.0
@Scenario_1.2
@UserReq_URS001
Scenario: 1.2. When the user types the text in the search field, then 8 search suggestions are displayed
    Given the application is available
        And the user is on the Search page
    When the user types "star" text in the Search field
    Then "8" search suggestions are displayed on the Search page

@US12345_Version_1.0.0
@Scenario_1.3
@UserReq_URS001
Scenario: 1.3. When the user types the text in the search field, then all search suggestions started from the typed text
    Given the application is available
        And the user is on the Search page
    When the user types "star" text in the Search field
    Then all search suggestions started from the text "star"

@US12345_Version_1.0.0
@Scenario_1.4
@UserReq_URS001
Scenario: 1.4. When the user initiates the search, then the 10 results are displayed on the Search Result page
    Given the application is available
        And the user is on the Search page
        And the search field containt the "star" text
    When the user clicks on the "Search" button on the "Search" page
    Then the "Result" list is displayed on the "Search Result" page
        And "10" search results are displayed on the Search Result page
```

Рисунок 4.6 – Файл історії BDD з використання практик описаних у розділах 2 та 3

На рисунку 4.6 видно, що кожен сценарій має конкретну задачу для перевірки конкретної дії. Також історію зручно читати і не потрібно знати деталей програмної реалізації як тестового фреймворку так і самого програмного продукту, що перевіряється, щоб зрозуміти суть. Також завдяки відступам кроків типу «And» текст не зливається і візуально простіше сприймати інформацію.

Також на рисунку 4.6 видно теги, які можуть бути використані як метадані для генерації тестової документації при наявності відповідного додатку для зчитування та обробки «сирого» файлу результату виконання тестів. Проте навіть не зважаючи на перевагу у документуванні BDD підхід все ще є корисним для команди розробки та тестування своєю зручністю та читабельністю, адже не залученим до написання тестів членам команди, набагато легше сприймати подібний текст при виконанні перевірки коду, що додається до репозиторію.

ВИСНОВКИ

Підхід BDD є однією з найкорисніших речей у сучасному процесі контролю якості, він допомагає створювати значущі та зрозумілі сценарії тестування та має низький поріг входу.

Цей підхід можна використовувати як для ручного тестування, так і для автоматизації, оскільки функції та сценарії BDD самі по собі не належать до одного з видів тестування. Незважаючи на низький поріг входу, згідно з офіційними документами, він вимагає певних удосконалень, коли він використовується на практиці, і такі вдосконалення (практики) були описані в цій роботі.

Крім того, слід зазначити, що існує більше практик, які можуть покращити використання BDD, але вони є особливими випадками і повністю залежать від перевіреного проекту, в деяких випадках вони можуть навіть погіршити ситуацію, тому вони не описувались.

При проектуванні фреймворку для автоматизованого тестування з використанням BDD підходу варто звернути увагу на особливості використання цього підходу та на описані практики при програмній імплементації, також пам’ятати, що така програмна реалізація може використовуватися як базовий шаблон для багатьох типів тестування різних видів програмного забезпечення.

Було опубліковано статтю «Behavior Driven Development Approach in the Modern Quality Control Process» в рамах науково-технічної конференції «Problem of Infocommunications. Science and Technolpgy (PIC S&T′2020)».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1 Смарт Д. Ф. BDD in Action: Behavior-driven development for the whole software lifecycle / Джон Ф. Смарт., 2014. – 384 с.

2 Behavior Driven Development Approach in the Modern Quality Control Process / О. П.Безсмертний, Н. В. Голян, І. В. Афанасьєва, В. В. Голян. // Problem of Infocommunications. Science and Technolpgy (PIC S&T′2020), Kharkiv, Ukraine.- 6-9 October 2020. – 2020.

3 Шатовська Т. Б. Дослідження ефективності застосування BDD-фреймворков у тестуванні безпеки WEB- орієнтованого ПЗ / Т. Б. Шатовська, І. В. Афанасьєва. // Вісник НТУ "ХПІ". – 2015. – С. 69–75.

4 Welcome to SpecFlow’s documentation! [Електронний ресурс] // Docs.specflow.org. – 2020. – URL: https://docs.specflow.org/projects/specflow/en/latest/ (дата звернення: 02.05.2021).

5 Нагі Г. Discovery: Explore behaviour using examples (BDD Books) / Г. Нагі, С. Роуз., 2018. – 99 с.

6 Ленка К. Behavior Driven Development: Tools and Challenges / К. Ленка, С. Кумар, С. Мамейн. // 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN). – 2018.

7 Фаулер М. Domain-Specific Languages / Мартін Фаулер., 2010. – 640 с.

8 Вінн М. The Cucumber Book: Behaviour-Driven Development for Testers and Developers / М. Вінн, А. Геллесой., 2012. – 336 с.

9 Грехем Д. Experiences of Test Automation: Case Studies of Software Test Automation / Д. Грехем, М. Фестер., 2012. – 672 с.

10 Майерс Г. The Art of Software Testing / Г. Майерс, К. Сендлер, Т. Бадгетт., 2011. – 256 с. – (3rd edition).

11 Dribbling complexity in model driven development using Naked Objects, domain driven design, and software design patterns / С. А.Соарес, М. Брандао, М. І. Кортес, Е. С. Кортес. // 2015 Latin American Computing Conference (CLEI). – 2015.

12 Безсмертний О. П. Використання метрик на різних етапах тестування / О. П. Безсмертний, Н. В. Голян, І. В. Груздо. // Актуальные научные исследования в современном мире // Журнал − Переяслав-Хмельницкий. – 2018. – №12. – С. 6–11.

13 Formal representation of knowledge for infocommunication computerized training systems / І.Шубін, І. Кириченко, П. Гончаров, С. Снісар. // 2017 4th International Scientific-Practical Conference Problems of Infocommunications Science and Technology, PIC S and T. – 2017. – С. 287–291.

14 Туревська О. Improving the automated testing of Web-based services by reflecting the social habits of target audiences / О. Туревська, І. Ю. Шубін. // 2015 Information Technologies in Innovation Business Conference, ITIB 2015 - Proceedings. – 2015. – С. 93–96.

15 Сімінюк А. Improve Selenium Code with Automation Patterns: Page Object Model Page Factory Page Elements Base Page Loadable Component / Алекс Сімінюк., 2017. – 112 с.

16 Чаубал П. А. Page Object Model using Selenium WebDriver & Java / Пінакін Ашок Чаубал., 2018. – 55 с.

17 Гріффітс Я. Programming C# 8.0: Build Cloud, Web, and Desktop Applications / Ян Гріффітс., 2020. – 800 с.

18 Ангелов А. Advanced SpecFlow: Using Hooks to Extend Test Execution Workflow [Електронний ресурс] / Антон Ангелов // Automate The Planet. – 2016. – URL: https://www.automatetheplanet.com/extend-test-execution-workflow-specflow-hooks/ (дата звернення: 02.05.2021).

19 To Mock or Not to Mock? An Empirical Study on Mocking Practices / Д.Спадіні, М. Аніче, М. Брантінк, А. Баккеллі. // 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). – 2017.

20 Ovais Mehboob Ahmed Khan. C# 7 and .NET Core 2.0 High Performance: Build highly performant, multi-threaded, and concurrent applications using C# 7 and .NET Core 2.0 / Ovais Mehboob Ahmed Khan., 2018. – 302 с.