ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

КВАЛІФІКАЦІЙНА РОБОТА

Тема: «Автоматизація тестування в Agile-процесах»

Ступінь вищої освіти – магістр
Спеціальність – 073 «Менеджмент»
Освітня програма «Agile-технології розробки програмного забезпечення»

ПОЯСНЮВАЛЬНА ЗАПИСКА

Керівник: кандидат економічних наук, доцент, завідувач кафедри інформаційного менеджменту, математики та статистики
Денис БАЛДИК

Виконав: здобувач
групи МЕН/Agile-24м
Самуїл СВЄЧНІКОВ

Засвідчую, що кваліфікаційна робота оформлена відповідно до ДСТУ 3008:2015 та не містить запозичень з праць інших авторів без відповідних посилань.
Здобувач: _____
(підпис)

Київ, 2026 р.

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

ЗАТВЕРДЖУЮ:
завідувач кафедри інформаційного менеджменту, математики та статистики
_____ Денис БАЛДИК
« 28 » жовтня 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
Свєчніков Самуїл Сергійович

Тема роботи	Автоматизація тестування в Agile-процесах
Номер та дата наказу про затвердження теми	№ 109-2 від 14 жовтня 2025 р.
Коротка постановка завдання	Проаналізувати особливості автоматизації тестування в Agile-командах розробки програмного забезпечення та дослідити її вплив на якість і стабільність мобільних продуктів. Розробити та впровадити кросплатформенну модель автоматизації тестування для мобільного застосунку AUTO.RIA з інтеграцією у CI/CD-процеси та оцінити її технічну й економічну ефективність.
Посилання на джерела інформації (не більше п’яти найменувань, які рекомендує науковий керівник)	1. Свєчніков С.С. Аналіз сучасних Agile-технологій та інструментів для автоматизації тестування / С. С. Свєчніков, Д. О. Балдик // Сучасний менеджмент організації: витоки, реалії та перспективи розвитку 2025: тези доповідей V Міжнародної Наукової конференції (17 квітня 2025 року). - Київ: Університет "КРОК", 2025 - URL: https://conf.krok.edu.ua/MMO/MMO-2025/paper/view/2782 2. RIA.com Marketplaces. Офіційний сайт компанії [Електронний ресурс]. URL: https://www.ria.com 3. Manifesto for Agile Software Development [Електронний ресурс] : презентація. – Agile Alliance, 2001. – URL: https://www.agilealliance.org/wp-content/uploads/2018/05/free-manifesto.pdf. 4. Appium Documentation. Appium Automation for Mobile Platforms [Електронний ресурс]. 2023. URL: https://appium.io/docs/en/latest/
Вимоги до кваліфікаційної роботи	Кваліфікаційна робота має містити теоретичне та/або практичне дослідження за темою роботи, яку слід розглядати як складне спеціалізоване завдання або практичну проблематику в галузі управління та адміністрування, яка характеризується комплексністю та невизначеністю умов і потребує застосування Agile-технологій.

Дата видачі завдання «27» жовтня 2025 р.

Керівник Денис БАЛДИК

Здобувач Самуїл СВЄЧНІКОВ

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання	Примітка
Підготовчий етап			
1	Вибір напряму дослідження та керівника.	01.09.2025 р.	*виконано*
2	Формування теми та призначення керівника.	22.09.2025 р.	*виконано*
3	Затвердження теми кваліфікаційної роботи.	14.10.2025 р.	*виконано*
4	Затвердження завдання на кваліфікаційну роботу.	16.10.2025 р.	*виконано*
Основний етап			
5	Розробка концепції та змісту кваліфікаційної роботи, погодження їх з науковим керівником	06.11.2025 р.	*виконано*
6	Підбір та вивчення джерел інформації з напряму дослідження.	08.11.2025 р.	*виконано*
7	Теоретико-методичний аналіз предметної області. Підготовка та подання керівнику розділу 1 кваліфікаційної роботи.	13.11.2025 р.	*виконано*
8	Реалізація гнучкого управління розробкою продукту. Підготовка та подання керівнику розділу 2 кваліфікаційної роботи.	20.11.2025 р.	*виконано*
9	Розробка рекомендацій щодо вдосконалення управління із застосуванням Agile-технологій. Підготовка та подання керівнику розділу 3 кваліфікаційної роботи.	27.11.2025 р.	*виконано*
10	Підготовка та подання керівнику першого варіанту всієї кваліфікаційної роботи.	01.12.2025 р.	*виконано*
11	Доопрацювання кваліфікаційної роботи з урахуваннямзауважень керівника та представлення керівнику доопрацьованого варіанту кваліфікаційної роботи	03.12.2025 р.	*виконано*
Завершальний етап			
12	Представлення рукопису для перевірки на плагіат.	08.12.2025 р.	*виконано*
13	Підготовка презентації та доповіді на передзахист.	22.12.2025 р.	*виконано*
14	Передзахист кваліфікаційної роботи.	23-24.12.2025 р.	*виконано*
15	Технічна самоекспертиза роботи на відповідність вимогам до оформлення та виправлення недоліків.	12-16.01.2026 р.	*виконано*
16	Експертиза роботи керівником та зовнішнім експертом (рецензентом).	20.01.2026 р.	*виконано*
17	Доопрацювання доповіді та презентації для захисту.	22.01.2026 р.	*виконано*
18	Захист кваліфікаційної роботи.	26-30.01.2026 р.	*виконано*

Керівник Денис БАЛДИК

Здобувач Самуїл СВЄЧНІКОВ

АНОТАЦІЯ

Свєчніков С. С. Автоматизація тестування в Agile-процесах

Пояснювальна записка кваліфікаційної роботи за спеціальністю 073 – Менеджмент (освітня програма – Agile-технології розробки програмного забезпечення), СО Магістр. – ВНЗ «Університет економіки та права «КРОК», Навчально-науковий інститут інформаційних та комунікаційних технологій, кафедра інформаційного менеджменту, математики та статистики, Київ, 2025р.

Кваліфікаційна робота присвячена дослідженню та впровадженню автоматизації тестування в умовах гнучких методологій розробки програмного забезпечення. У роботі проаналізовано теоретичні засади Agile-підходів, роль автоматизованого тестування у забезпеченні якості програмних продуктів, а також сучасні інструменти й фреймворки автоматизації для веб-, мобільних та API-рівнів. Особливу увагу приділено практичній реалізації кросплатформенної моделі автоматизації тестування для мобільного застосунку AUTO.RIA, що функціонує в умовах високого навантаження та частих релізів.

У межах роботи розроблено масштабований тестовий фреймворк на базі Java, Appium та TestNG, побудований за архітектурою Page Object і Helper Pattern, а також реалізовано його інтеграцію в CI/CD-процеси з використанням Jenkins, GitLab CI, Device Farm та Allure. Запропонована модель забезпечує автоматичний запуск тестів, централізовану звітність і підтримку паралельного тестування на реальних пристроях Android та iOS. Проведено економічну оцінку ефективності впровадження автоматизації, яка підтвердила доцільність інвестицій та позитивний фінансовий результат проєкту.

Практичні результати роботи можуть бути використані для впровадження та масштабування автоматизації тестування в Agile-командах продуктових IT-компаній, що розробляють мобільні застосунки.

Ключові слова: автоматизація тестування, Agile, Scrum, CI/CD, мобільні застосунки, Appium, QA.

ANNOTATION

Sviechnikov S. S. Test automation in Agile processes

Qualification paper explanatory note in specialty 073 – Management (educational program – Agile Software Development Technologies), Master's Degree. – HEI «University of Economics and Law «KROK», Educational and Scientific Institute of Information and Communication Technologies, Department of Information Management, Mathematics and Statistics, Kyiv, 2025.

The qualification work is devoted to the study and implementation of test automation within agile software development processes. The paper analyzes the theoretical foundations of Agile methodologies, the role of automated testing in ensuring software quality, and modern tools and frameworks for web, mobile, and API testing. Special attention is paid to the practical development of a cross-platform test automation model for the AUTO.RIA mobile application, which operates under high load and frequent release conditions.

As part of the research, a scalable test automation framework based on Java, Appium, and TestNG was developed using the Page Object and Helper Pattern architecture. The framework was fully integrated into CI/CD pipelines using Jenkins, GitLab CI, Device Farm, and Allure, providing automated test execution, centralized reporting, and parallel testing on real Android and iOS devices. An economic efficiency assessment was conducted, confirming the feasibility of investments and the positive financial impact of the automation project.

The results of the study can be applied to the implementation and scaling of test automation in Agile teams of product-oriented IT companies developing mobile applications.

Keywords: test automation, Agile, Scrum, CI/CD, mobile applications, Appium, QA.

ЗМІСТ

ВСТУП

Сучасна індустрія розробки програмного забезпечення характеризується високою динамікою змін, зростанням конкуренції між продуктовими компаніями та скороченням часу виведення цифрових продуктів на ринок. У цих умовах якість програмного забезпечення стає одним із ключових чинників конкурентоспроможності, оскільки користувачі очікують стабільної, безперебійної та прогнозованої роботи застосунків незалежно від частоти оновлень. Особливо актуальною ця проблема є для мобільних продуктів, які функціонують у високонавантажених цифрових екосистемах та регулярно оновлюються в межах коротких релізних циклів.

Актуальність теми дослідження зумовлена обмеженістю традиційних підходів до тестування в умовах гнучких методологій розробки. Agile-підходи, зокрема Scrum і Kanban, передбачають короткі ітерації, швидкий зворотний зв’язок та постійну адаптацію до змін вимог. У таких умовах ручне тестування не завжди здатне забезпечити достатній рівень покриття та стабільності перевірок, оскільки характеризується високою трудомісткістю, повторюваністю операцій і значною залежністю від людського фактора. Це створює ризики затримок релізів, накопичення дефектів і зниження якості продукту, що підтверджується результатами сучасних досліджень у сфері управління якістю програмного забезпечення [1].

Незважаючи на активний розвиток інструментів і фреймворків автоматизації тестування, питання їх ефективної інтеграції в Agile-процеси, зокрема в мобільних проєктах, залишається предметом наукових і практичних дискусій. Автоматизація тестування потребує не лише вибору технічних засобів, а й формування цілісної архітектури, узгодженої зі структурою команди, бізнес-логікою продукту та CI/CD-процесами. Особливого значення набуває поєднання автоматизованих UI- та API-тестів, забезпечення стабільності тестових сценаріїв і економічне обґрунтування доцільності таких інвестицій [2].

Практичну значущість зазначеної проблеми демонструє досвід продуктових компаній, що працюють із високонавантаженими мобільними

застосунками. Одним із таких прикладів є мобільний застосунок AUTO.RIA — один із найбільших автомобільних маркетплейсів в Україні, який щоденно обробляє понад 1,5 млн пошукових запитів і має складну бізнес-логіку з великою кількістю інтеграцій із бекенд-сервісами [3]. До впровадження автоматизації тестування QA-процеси в команді характеризувалися значною тривалістю регресійних перевірок, перевантаженням ручних тестувальників і підвищеною кількістю дефектів у продакшн-середовищі. Це обумовило необхідність розроблення системної моделі автоматизації тестування, адаптованої до умов Agile-розробки.

Метою кваліфікаційної роботи є дослідження теоретичних засад автоматизації тестування в Agile-середовищі та розроблення практичної моделі автоматизації тестування для мобільного застосунку AUTO.RIA з оцінкою її технічної та економічної ефективності.

Завданням дослідження є: проаналізувати особливості Agile-методологій і їхній вплив на організацію тестування; дослідити роль автоматизації тестування в сучасних Agile-командах; розглянути актуальні інструменти та фреймворки автоматизованого тестування; розробити архітектуру тестового фреймворку для мобільного застосунку; інтегрувати автоматизовані тести в CI/CD-процеси; провести економічний аналіз ефективності впровадження автоматизації та оцінити отримані результати.

Об’єктом дослідження кваліфікаційної роботи є процес автоматизації тестування в Agile-командах розробки програмного забезпечення.

Предметом дослідження є методи, принципи, інструменти та засоби автоматизованого тестування мобільних застосунків у межах гнучких моделей розробки програмного забезпечення. Предмет дослідження розглядається в межах об’єкта та конкретизує практичні аспекти реалізації автоматизації.

Методами дослідження є аналіз і узагальнення наукових і прикладних джерел з тематики Agile та автоматизації тестування, порівняльний аналіз інструментів і фреймворків, моделювання архітектури тестового фреймворку, експериментальне впровадження автоматизованих тестів, а також економічні методи оцінки ефективності інвестицій (NPV, ROI, BCR).

Практична значущість роботи полягає у розробленні та впровадженні масштабованого фреймворку автоматизації тестування для мобільного застосунку AUTO.RIA, який використовується в реальному виробничому середовищі та може бути адаптований іншими мобільними командами компанії RIA.com Marketplaces. Отримані результати сприяють підвищенню стабільності релізів, оптимізації QA-процесів і зменшенню витрат на тестування.

Апробація результатів дослідження здійснювалася в межах практичної діяльності автора у складі Agile-команди AUTO.RIA, де розроблений фреймворк було інтегровано в CI/CD-процеси та використано для автоматизованого тестування критичних сценаріїв мобільного застосунку.

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів і висновків, викладених на 55 сторінках тексту. Матеріали роботи містять 15 таблиць, 1 схема та 3 рисунки. Список використаних джерел налічує 30 найменувань, розміщених на 3 сторінках.

РОЗДІЛ 1
ОСОБЛИВОСТІ AGILE-ПІДХОДІВ ТА АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ

1.1. Основи Agile-методологій та їх вплив на тестування

Метою цього розділу є теоретичне обґрунтування ролі автоматизації тестування в межах Agile-процесів, аналіз принципів гнучких методологій розробки програмного забезпечення та визначення ключових передумов, які впливають на трансформацію підходів до забезпечення якості у швидких ітераційних циклах. Матеріал розділу формує теоретичну основу для подальшого аналізу інструментів і практичних рішень, що будуть розглянуті у наступних розділах роботи.

Упродовж останніх двох десятиліть Agile-підходи стали домінуючою парадигмою у сфері розробки програмного забезпечення. Це зумовлено зростанням складності інформаційних систем, підвищенням вимог до швидкості виведення продуктів на ринок і необхідністю оперативно реагувати на зміну бізнес-вимог. Гнучкі методології орієнтовані на ітеративну та інкрементну розробку, постійний зворотний зв’язок і тісну взаємодію із замовником, що принципово відрізняє їх від традиційних каскадних моделей. На відміну від Waterfall-підходу, де розробка, тестування та впровадження відбуваються послідовно, Agile дозволяє командам працювати у коротких спринтах тривалістю від одного до чотирьох тижнів, постійно постачаючи інкременти готової до використання функціональності.

Концептуально Agile-підхід ґрунтується на положеннях «Маніфесту гнучкої розробки програмного забезпечення» (2001 р.), який визначає чотири базові цінності, що формують філософію сучасних гнучких методологій. У центрі Agile перебуває пріоритет людей та їхньої взаємодії над формальними процесами й інструментами, оскільки саме ефективна комунікація в команді забезпечує швидке ухвалення рішень і зменшення кількості помилок. Створення працюючого продукту розглядається як більш значущий результат, ніж підготовка вичерпної документації, яка може швидко втратити актуальність у

динамічному середовищі. Співпраця з клієнтом оцінюється вище за суворе дотримання контрактних умов, а готовність до змін визнається важливішою за слідування початковому плану [4]. Сукупність цих цінностей формує підхід, у якому гнучкість, адаптивність і швидкість реагування на нові вимоги є ключовими чинниками успішної реалізації програмних проєктів.

Важливо підкреслити, що Agile не диктує жорстко визначеного набору технічних інструментів або конкретних методів реалізації, натомість він задає загальні принципи та поведінкові патерни команди. Саме тому Agile-практики можуть по-різному реалізовуватися залежно від контексту проєкту, розміру команди та організаційної культури компанії. Така гнучкість дозволяє адаптувати процеси розробки й тестування під конкретні бізнес-потреби, що є особливо важливим для продуктових компаній із постійно змінюваними вимогами.

У межах гнучких методологій найбільш поширеними є фреймворки Scrum, Kanban та Extreme Programming (XP), а також масштабовані моделі, зокрема SAFe і LeSS, що застосовуються у великих організаціях. Scrum базується на чіткому розподілі ролей, таких як Product Owner, Scrum Master і команда розробки, а також на визначеному наборі артефактів і ритуалів, спрямованих на забезпечення прозорості, регулярного зворотного зв’язку та безперервного вдосконалення процесів. Kanban, своєю чергою, зосереджується на оптимізації потоку задач, зменшенні кількості незавершеної роботи та скороченні часу виконання завдань, що особливо актуально для команд із нерівномірним навантаженням. XP акцентує увагу на інженерних практиках, спрямованих на підвищення якості коду, зокрема на парному програмуванні, безперервному рефакторингу та тестуванні, орієнтованому на поведінку системи. Масштабовані підходи SAFe і LeSS дозволяють поширити принципи Agile на рівень великих програм і портфелів, забезпечуючи узгодженість роботи десятків або сотень команд.

Ітеративний та інкрементний характер Agile-розробки суттєво впливає на організацію тестування. Кожна ітерація охоплює повний життєвий цикл створення функціональності — від формування вимог до реалізації, перевірки та

випуску інкременту продукту. За таких умов тестування не може бути відкладеним етапом і повинно виконуватися часто, швидко та безпосередньо інтегруватися у процес розробки. Постійний зворотний зв’язок є невід’ємною складовою Agile-підходу: команди отримують інформацію від користувачів, замовників, результатів автоматизованих тестів і систем безперервної інтеграції, що дозволяє виявляти дефекти практично одразу після їх появи та оперативно реагувати на них.

Особливе значення в Agile має командна взаємодія та роль фахівців із забезпечення якості. У Scrum-командах тестувальник не розглядається як окрема функціональна одиниця, відокремлена від розробки, а є повноправним учасником команди, відповідальним за якість кінцевого продукту. QA-фахівець залучається до планування спринтів, бере участь в уточненні вимог, формуванні критеріїв приймання (Acceptance Criteria) та спільно з розробниками визначає критерії готовності (Definition of Done). Такий підхід сприяє тому, що тестування інтегрується у всі етапи розробки, а питання якості обговорюються ще на етапі проєктування функціональності, а не після її реалізації.

Готовність до змін є ще однією фундаментальною характеристикою Agile-методологій. Якщо в традиційних моделях розробки зміна вимог часто розглядається як серйозний ризик і джерело додаткових витрат, то в Agile вона сприймається як природна складова процесу створення програмного продукту. Це означає, що тестова документація, тестові сценарії та загальний підхід до тестування повинні бути гнучкими, легко адаптованими та стійкими до змін інтерфейсу й бізнес-логіки. Саме в цьому контексті автоматизація тестування набуває особливої значущості, оскільки вона дозволяє швидко повторювати перевірки та підтримувати актуальність тестового покриття навіть за умов частих змін вимог. Практика роботи з мобільним застосунком AUTO.RIA показала, що регулярні оновлення між спринтами суттєво підвищували трудомісткість ручного тестування, що стало одним із ключових чинників переходу до автоматизованих підходів.

У гнучких методологіях тестування не є завершальною фазою життєвого циклу розробки, як це характерно для каскадних моделей, а виконується

паралельно з реалізацією функціональності. Відповідальність за якість розподіляється між усіма членами команди, що сприяє підвищенню прозорості процесів, покращенню комунікації та формуванню спільного розуміння цілей і критеріїв успіху. Такий підхід створює передумови для ефективного впровадження автоматизації тестування як невід'ємного елементу Agile-процесів та закладає основу для практичних рішень, розглянутих у наступних розділах роботи.

1.2. Роль автоматизації тестування в Agile-процесах

Автоматизація тестування набуває критичного значення в умовах гнучких методологій розробки програмного забезпечення, де релізи відбуваються часто, а обсяг функціональності зростає інкрементально від спринту до спринту. Agile-підходи орієнтовані на короткі ітерації, швидкий зворотний зв'язок і постійну адаптацію до змін вимог, що суттєво підвищує навантаження на процеси забезпечення якості. За таких умов традиційні підходи до тестування, засновані переважно на ручних перевірках, поступово втрачають ефективність і не здатні забезпечити необхідний рівень стабільності продукту без істотного зростання витрат часу та ресурсів.

Однією з ключових причин зростання ролі автоматизації в Agile є необхідність скорочення часу регресійного тестування. Регресія у гнучких проєктах виконується значно частіше, ніж у класичних каскадних моделях, оскільки кожен спринт передбачає додавання або зміну функціональності, яка потенційно може вплинути на вже реалізовані модулі. У практиці AUTO.RIA до впровадження автоматизації повна регресія мобільного застосунку вимагала залучення чотирьох Manual QA та тривала від шести до восьми робочих днів, що безпосередньо впливало на терміни релізів і створювало ризики затримок. Після запровадження автоматизованих тестів час регресійних перевірок скоротився до двох–трьох днів, при цьому тестове покриття ключових сценаріїв зросло, а ймовірність пропуску критичних дефектів суттєво зменшилася. Такий результат демонструє, що автоматизація є не лише технічним удосконаленням, а стратегічним інструментом оптимізації Agile-процесів.

Важливим аспектом є інтеграція тестування з системами безперервної інтеграції та доставки програмного забезпечення. У більшості сучасних Agile-команд розробка тісно пов’язана з CI/CD-інфраструктурою, зокрема з використанням таких інструментів, як GitLab CI або Jenkins. У межах цієї моделі автоматизовані тести запускаються після кожного merge-запиту, виконуються регулярні nightly-прогони та проводяться smoke-перевірки перед релізами. Такий підхід дозволяє забезпечити безперервний контроль якості та оперативно виявляти дефекти на ранніх етапах життєвого циклу розробки. У результаті автоматизоване тестування стає невід’ємною складовою Definition of Done, оскільки саме воно гарантує, що функціональність відповідає встановленим вимогам і не порушує стабільність системи.

Запуск unit-, API- та UI-тестів після кожного коміту відіграє ключову роль у зменшенні вартості виправлення дефектів. Чим раніше помилка виявляється, тим менше зусиль потрібно для її усунення, що підтверджується як теоретичними дослідженнями, так і практикою Agile-проєктів. Автоматизовані перевірки дозволяють швидко локалізувати проблеми, мінімізувати ризики виникнення регресій та підвищити загальну стабільність процесу розробки. У контексті мобільних застосунків це особливо важливо, оскільки помилки, що потрапляють у продакшн, безпосередньо впливають на користувацький досвід і репутацію продукту.

Окремої уваги заслуговує вплив людського фактора на якість тестування. Повторювані ручні перевірки однакових сценаріїв призводять до так званого «замилення ока», втоми тестувальників і зниження концентрації, що підвищує ризик пропуску дефектів. Навіть за високого рівня професійної підготовки людський фактор залишається обмеженням, яке неможливо повністю усунути без автоматизації. Автоматизовані тести, на відміну від ручних перевірок, виконуються стабільно та однаково щоразу, забезпечуючи передбачуваний і відтворюваний результат незалежно від тривалості чи частоти прогонів.

Автоматизація також відіграє вирішальну роль у забезпеченні кросплатформенної стабільності, що є критичним для мобільних проєктів. Один і той самий автотест може бути виконаний на різних пристроях, версіях

операційних систем і тестових середовищах без зміни логіки сценарію. Це дозволяє оперативно виявляти проблеми, пов’язані з фрагментацією платформ, відмінностями в реалізації UI-компонентів або поведінці системних API. У випадку мобільних застосунків, таких як AUTO.RIA, де продукт підтримує Android та iOS і використовується на великій кількості реальних пристроїв, ця властивість автоматизації є однією з ключових переваг.

Зі зростанням функціональності продукту неминуче зростає й кількість тестових сценаріїв, які необхідно виконувати для забезпечення належного рівня якості. У таких умовах виключно ручне тестування призводить до експоненційного збільшення витрат часу та ресурсів і стає стримувальним фактором для розвитку продукту. Автоматизація дозволяє масштабувати тестове покриття без пропорційного зростання витрат, зберігаючи швидкість випуску релізів і підтримуючи темп розробки, характерний для Agile-процесів.

Таким чином, автоматизація тестування в Agile-середовищі виступає не допоміжним інструментом, а фундаментальним елементом системи забезпечення якості. Вона забезпечує скорочення часу регресії, підвищення стабільності продукту, зменшення впливу людського фактора та створює умови для масштабування тестування разом із розвитком функціональності. У поєднанні з CI/CD-процесами автоматизоване тестування формує основу для безперервного контролю якості та є необхідною передумовою успішного функціонування Agile-команд у сучасній індустрії програмного забезпечення.

1.3. Основні методики та інструменти автоматизації тестування

Цей підрозділ описує фундаментальні підходи до автоматизації, які застосовуються в Agile-проєктах.

Unit-тестування охоплює бізнес-логіку застосунку, виконується з високою швидкістю та характеризується найнижчою вартістю розробки порівняно з іншими рівнями тестування. У мобільних командах до таких тестів належать перевірки компонентів ViewModel, UseCase, Presenter та Repository, що забезпечують коректність роботи ключових логічних модулів.

Інтеграційне тестування спрямоване на оцінку взаємодії між окремими компонентами системи, включаючи API та зовнішні сервіси. У контексті застосунку AUTO.RIA цей рівень тестування широко застосовується для перевірки JSON-відповідей, функціонування фільтрів та коректної обробки серверних запитів у пошукових механізмах.

UI-тестування, або end-to-end-перевірки, охоплює найбільш критичні для користувача сценарії, зокрема авторизацію, пошук транспортних засобів, застосування фільтрів, взаємодію з обраними елементами, перегляд і створення оголошень та роботу з пакетами публікацій. У проєкті ці сценарії автоматизуються за допомогою фреймворку Appium, що забезпечує можливість комплексної перевірки мобільного інтерфейсу. Зазначені підходи органічно вписуються у модель Agile Testing Quadrants (рис 1.1, табл 1.1), запропоновану Л. Кріспін і Дж. Грегорі, яка структурує різні типи тестувань залежно від їхнього призначення та рівня взаємодії з продуктом.

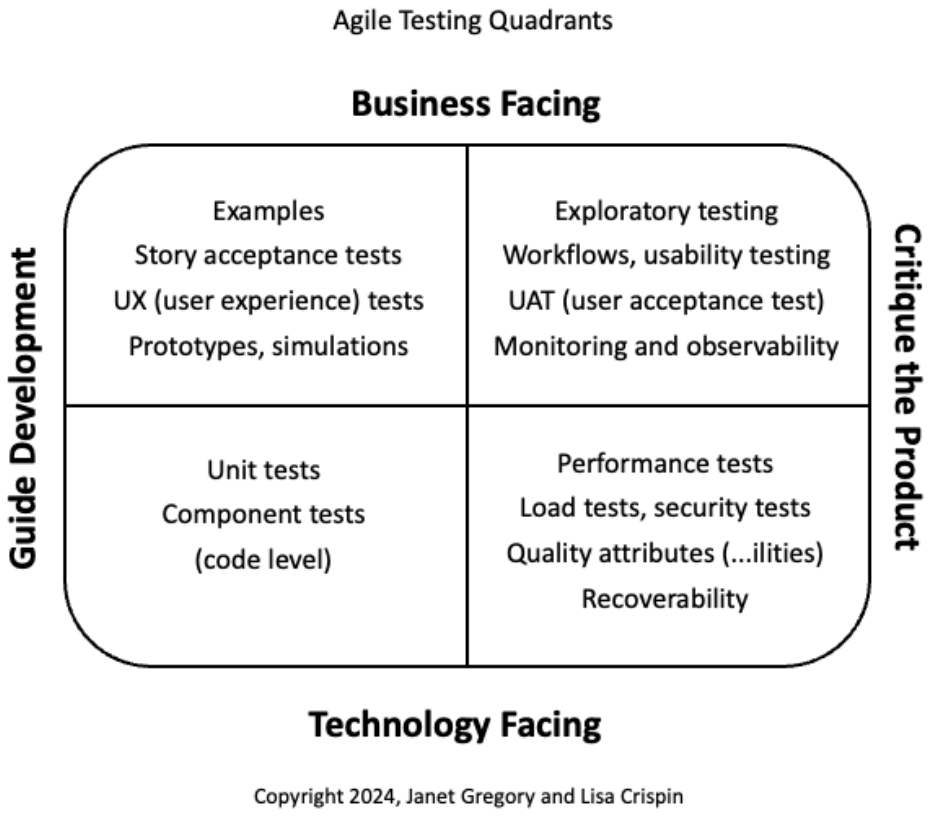

Рисунок 1.1 – Таблиця Agile Testing Quadrats

Джерело: [5]

Квадранти гнучкого тестування – це інструмент мислення, який допомагає командам планувати та виконувати тестові дії, щоб вони могли впевнено надавати цінність для клієнтів [5].

Таблиця 1.1 – Модель Л. Кріспін та Дж. Грегорі

Q1	Unit tests
Q2	Automated Tests (functional)
Q3	Exploratory / Usability
Q4	Performance, Security

Автоматизація закриває Q1 + Q2, а Manual QA зосереджується на Q3 + Q4 - це оптимально.

Для забезпечення підтримуваності, розширюваності та надійності автоматизованих тестових рішень у проєкті використовуються кілька ключових архітектурних підходів, серед яких центральне місце займає шаблон Page Object Model (POM). Цей підхід передбачає подання кожного екрану або функціонально значущого інтерфейсного компонента у вигляді окремого класу, що інкапсулює локатори й допустимі дії з елементами користувацького інтерфейсу. Завдяки цьому тестові сценарії не містять безпосередніх селекторів і взаємодіють з інтерфейсом виключно через методи відповідних Page-класів. Така організація істотно спрощує підтримку тестів, оскільки будь-які зміни в UI - наприклад, модифікація ідентифікаторів елементів або структури екранів - потребують корекції лише у відповідному Page-класі, а не у всіх тестових наборах одночасно. POM забезпечує чітке розділення відповідальностей, коли опис UI та елементарних дій зосереджено у Page-класах, тоді як тестова логіка та перевірки реалізуються на рівні тестових сценаріїв, що підвищує структурованість, масштабованість і прозорість фреймворку [6].

Доповненням до POM виступає Helper Layer - архітектурний шар, у якому реалізується бізнес-логіка взаємодії користувача із застосунком. Оскільки реальні сценарії рідко обмежуються однією операцією, такий підхід дозволяє поєднувати послідовності дій, що охоплюють, наприклад, авторизацію, пошук транспортних засобів із застосуванням фільтрів, створення оголошень,

керування обліковими записами або здійснення покупок у межах маркетплейсу. Завдяки Helper Layer тести містять лише високорівневі виклики бізнес-процесів, а складні низькорівневі взаємодії з UI приховуються за спеціалізованими методами. Це зменшує дублювання коду, запобігає накопиченню надмірної логіки у Page-класах і значно підвищує стабільність та читабельність тестів. Крім того, на цьому рівні централізовано реалізуються механізми очікування, повторних спроб виконання дій, а також логування, що позитивно впливає на стійкість тестового середовища.

Наступним важливим елементом архітектури є Watcher Utility Layer - спеціалізований набір технічних утиліт, призначених для роботи зі сценаріями, у яких взаємодія з інтерфейсом або даними має складний, динамічний характер. Такий підхід відповідає рекомендаціям провідних фахівців у сфері розробки та тестування (зокрема описаним у працях М. Фаулера та в офіційних документаціях Selenium і Appium), які наголошують на необхідності винесення низькорівневих операцій у окремі допоміжні модулі задля забезпечення принципу розділення відповідальностей [7]. У рамках мобільної автоматизації до цього шару входять утиліти ScrollWatcher, JsonFilterHelper та OrientationHelper.

ScrollWatcher вирішує одну з фундаментальних проблем автоматизації мобільних застосунків - обробку елементів, видимість або наявність яких залежить від прокручування списків. Через особливості нативних інтерфейсів мобільних ОС елементи часто не розташовані у межах поточної області перегляду, підвантажуються асинхронно або відображаються залежно від конфігурації пристрою. Офіційна документація Appium наголошує, що операції гортання можуть бути нестабільними і потребують реалізації додаткових стратегій пошуку, циклічних алгоритмів та умов видимості.

Враховуючи це, ScrollWatcher забезпечує автоматизоване прокручування списків до появи цільового елемента, відстежує кількість ітерацій, застосовує механізми повторних спроб і використовує динамічні умови очікування, рекомендовані інструментами Selenium та Appium. Такий підхід є особливо корисним під час взаємодії з довгими та динамічними списками, що широко

використовуються в мобільному застосунку AUTO.RIA, зокрема у каталозі оголошень, списках фільтрів або формах створення оголошення [8].

JsonFilterHelper покликаний забезпечити коректну взаємодію з бізнес-логікою застосунку на рівні API. У сучасній архітектурі мобільних застосунків значна частина логіки фільтрації, сортування та формування списків реалізується на стороні сервера, а в AUTO.RIA це особливо актуально через складність пов'язаних JSON-структур, що містять параметри «selected», «content», «value», «title» тощо. JsonFilterHelper дає змогу отримувати відповідні JSON-фільтри, здійснювати пошук вкладених елементів за ключовими параметрами, перевіряти відповідність стану інтерфейсу значенням, що повертаються API, та виявляти невідповідності між даними UI й бекенд-сервісів. Документація Rest-Assured детально описує механізми роботи з JSONPath, що дозволяють швидко обробляти складні структури, а М. Фаулер у роботах, присвячених тестуванню API, наголошує на цінності подібних валідаторів для прискорення тестових циклів [9; 10].

OrientationHelper забезпечує стабільність сценаріїв, пов'язаних зі зміною орієнтації екрана. Нативні мобільні операційні системи реагують на зміну орієнтації як на подію конфігураційної зміни, що може спричинити перезавантаження активності, адаптацію layout-структури або зміну координат елементів. Документація Android зазначає, що подібні події здатні суттєво впливати на поведінку застосунку, а Appium рекомендує застосовувати спеціальні механізми керування орієнтацією та дотримуватися коректного очікування після ротації [11]. OrientationHelper уніфікує ці дії, визначає поточний режим орієнтації, здійснює зміну через AppiumDriver, контролює стабілізацію інтерфейсу та перевіряє готовність елементів до взаємодії.

Виділення таких утиліт у окремий Utility Layer відповідає загальноприйнятим рекомендаціям щодо структурування тестових фреймворків, відповідно до яких Page Object повинен містити лише опис UI, Helper Layer має реалізовувати бізнес-процеси, а Utility Layer — забезпечувати низькорівневу технічну взаємодію, що не належить ні до одного з попередніх рівнів. Такий підхід є практичним застосуванням принципу «separation of concerns» [12] і дає

змогу підвищити гнучкість, масштабованість і стійкість тестового фреймворку при зміні функціональності застосунку або зростанні кількості тестових сценаріїв.

Висновки до розділу 1

У першому розділі було здійснено теоретичне обґрунтування сутності та ролі автоматизації тестування в умовах Agile-процесів, що дало змогу сформувати концептуальну базу для подальших аналітичних і практичних досліджень. На основі аналізу положень «Маніфесту гнучкої розробки» та поширених Agile-фреймворків (Scrum, Kanban, XP, SAFe, LeSS) показано, що гнучкі методології орієнтовані на ітеративність, інкрементність, постійний зворотний зв’язок і тісну співпрацю з замовником. Було обґрунтовано, що такі характеристики безпосередньо впливають на організацію тестування, оскільки за умов коротких спринтів і частих релізів контроль якості має бути інтегрованим у процес розробки, здійснюватися безперервно та забезпечувати швидке виявлення дефектів.

У роботі детально розглянуто особливості розподілу відповідальності за якість у Agile-командах, де тестувальник перестає бути «окремою ланкою» і виступає повноцінним учасником команди розробки. Показано, що участь QA-фахівців у плануванні спринтів, уточненні вимог, формуванні критеріїв приймання та визначенні Definition of Done сприяє трансформації тестування з фінальної стадії життєвого циклу в постійний супровід розробки. На прикладі мобільного застосунку AUTO.RIA продемонстровано, що часті зміни вимог у поєднанні зі значною функціональною складністю продукту призводили до істотного зростання трудомісткості ручних перевірок, що, у свою чергу, обґрунтувало необхідність системного впровадження автоматизації тестування.

Окремим результатом розділу є розкриття ролі автоматизації тестування саме в Agile-контексті. Було показано, що автоматизовані тести дозволяють суттєво скоротити час проведення регресійних перевірок, забезпечити оперативний зворотний зв’язок у межах процесів безперервної інтеграції та постачання (CI/CD), знизити вплив людського фактору та підвищити

повторюваність результатів. Наголошено, що регулярний запуск Unit-, API- та UI-тестів після кожного коміту дає змогу виявляти дефекти на ранніх етапах, зменшуючи вартість їхнього усунення та мінімізуючи ризики регресій, а стабільність виконання автотестів на різних пристроях, версіях операційних систем і середовищах є критичною для мобільних застосунків.

У межах підрозділу 1.3 було узагальнено основні методики тестування (unit-, інтеграційне та end-to-end), які відіграють ключову роль у побудові цілісної системи контролю якості. Показано, що в мобільних продуктах unit-тести забезпечують коректність бізнес-логіки, інтеграційні тести — узгодженість взаємодії компонентів та API, а UI-тести — цілісність критичних користувацьких сценаріїв на кшталт авторизації, пошуку, роботи з фільтрами та створення оголошень. Застосування моделі Agile Testing Quadrants дозволило структурувати типи тестів за їхнім призначенням, рівнем технічної абстракції та взаємодією з користувачем, що дало можливість обґрунтувати оптимальний розподіл зон відповідальності між автоматизованими перевірками (квадранти Q1–Q2) та ручним тестуванням (квадранти Q3–Q4.

Важливим підсумком розділу є також опис архітектурних підходів до побудови тестового фреймворку, які забезпечують підтримуваність і масштабованість автоматизації. Зокрема, було проаналізовано застосування шаблону Page Object Model, що забезпечує інкапсуляцію UI-елементів у спеціалізованих класах і зменшує залежність тестів від змін інтерфейсу; Helper Layer, який реалізує бізнес-логіку сценаріїв та підвищує читабельність тестів; а також Watcher Utility Layer, що включає утиліти для роботи з динамічними списками, API-фільтрами та орієнтацією екрана. Показано, що таке багаторівневе структурування відповідає принципу розділення відповідальностей і дозволяє створювати стійкі до змін, гнучкі та придатні до розвитку рішення з автоматизації тестування.

РОЗДІЛ 2
СУЧАСНІ ПІДХОДИ ТА ІНСТРУМЕНТИ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ

2.1. Аналіз сучасних технологій та інструментів автоматизації тестування

У цьому розділі розглянуто сучасні технології автоматизації тестування, їх вплив на ефективність Agile-процесів, а також особливості інтеграції автоматизованих тестів у CI/CD. Представлено кейс-стаді, що демонструють практичне застосування інструментів автоматизації в реальних проєктах, включаючи досвід компанії RIA.com Marketplaces, отриманий під час практики.

ТОВ «RIA.com Marketplaces» - одна з провідних IT-компаній України, що спеціалізується на створенні високонавантажених онлайн-платформ. Найбільш відомими продуктами є AUTO.RIA, DOM.RIA, Ria.com, Ria Travel. Основна діяльність компанії полягає у розробці, підтримці та масштабуванні цифрових маркетплейсів, які об’єднують мільйони користувачів і продавців.

Компанія активно розвиває напрям мобільних застосунків, що забезпечують швидкий доступ до сервісів у будь-який час. Особливу роль у портфелі компанії відіграє застосунок AUTO.RIA, який входить до топ-3 найвідвідуваніших автомобільних ресурсів в Україні. Щодня через додаток здійснюється понад 1,5 млн пошуків і понад 50 000 контактів між покупцями та продавцями.

Організаційна структура компанії (табл 2.1) побудована за Agile-принципами. Кожен продукт підтримується окремою Scrum-командою, до складу якої входять: Product Owner, Scrum Master, 4–5 розробників, Manual QA Testers, Automation QA Engineer.

Таблиця 2.1 – Організаційна структура команди проєкту

Посада	Кількість	Основні обов’язки
Product Owner	1	Формує backlog, визначає пріоритети завдань, приймає результати спринтів

Посада	Кількість	Основні обов'язки
Scrum Master	1	Координує процес Scrum, усуває перешкоди в команді
Automation QA Engineer	1	Розробка фреймворку, створення та підтримка автотестів, інтеграція з CI/CD
Manual QA Testers	4	Виконання мануальних сценаріїв, досліджувальне тестування, перевірка нових функцій
Android Developers	2	Реалізація клієнтського функціоналу для Android
iOS Developers	2	Реалізація клієнтського функціоналу для iOS
DevOps Engineer	1	Налаштування Jenkins, Device Farm, інтеграції Allure та GitLab CI

Таке розподілення забезпечує незалежність команд, короткі цикли релізів і можливість безперервного вдосконалення продукту.

Автоматизація тестування є однією з ключових інженерних практик у сучасних IT-компаніях. Вона дозволяє скоротити обсяг ручних повторюваних перевірок, підвищити стабільність тестового процесу та забезпечити швидке отримання зворотного зв'язку у продукті, що постійно оновлюється. Розвиток фреймворків автоматизації створив широкий спектр інструментів, які адаптовані під різні типи застосунків: веб, мобільні, API, десктопні системи.

У цьому підрозділі було здійснено порівняльний аналіз найпоширеніших інструментів автоматизації веб-тестування (табл 2.2), зокрема Selenium, Cypress та Playwright, із акцентом на їх архітектурних особливостях, перевагах і обмеженнях. Selenium розглядається як один із найбільш відомих та широко використовуваних інструментів для автоматизації веб-застосунків; він працює на основі WebDriver API та реалізує віддалене керування браузером таким чином, що дії тесту максимально наближені до дій реального користувача. WebDriver взаємодіє з браузером нативно, локально або через віддалений Selenium Server, що означає якісний крок уперед у порівнянні з попередніми підходами до автоматизації браузерів, а сам термін «Selenium WebDriver» охоплює як мовні обгортки, так і реалізації драйверів керування окремими браузерами [13]. Серед

ключових переваг Selenium відзначено кросбраузерність, підтримку широкого спектра мов програмування (Java, Python, C#, JavaScript), велику спільноту користувачів і гнучкість налаштувань, тоді як до недоліків належать нижча стабільність у складних інтерфейсах, необхідність написання значної кількості допоміжних обгорток навколо WebDriver та певні труднощі з автоматизацією сучасних односторінкових застосунків (SPA).

Cypress позиціонується як сучасний фреймворк для тестування фронтенд-застосунків, принциповою відмінністю якого від Selenium є виконання тестів у тому ж середовищі, що й сам веб-додаток, що сприяє зменшенню флейковості тестів [14]. Його архітектура та синтаксис орієнтовані на розробників: наявність зрозумілої мови опису сценаріїв на JavaScript, автоматичне очікування появи елементів та подій, тісна інтеграція з DOM і розширені можливості відлагодження через вбудовані інструменти значно спрощують створення та підтримку тестів. Крім того, Cypress підтримує збереження відео- та скріншот-логів виконання тестів, що полегшує аналіз збоїв, та без особливих ускладнень інтегрується з системами безперервної інтеграції. Водночас його обмеженнями є прив’язка виключно до стеку JavaScript/TypeScript і фокус лише на веб-застосунках, без нативної підтримки мобільних платформ.

Playwright розглядається як представник нового покоління інструментів браузерної автоматизації, розроблений компанією Microsoft, який має низку суттєвих переваг порівняно із Selenium [15]. Він забезпечує високу швидкість та стабільність виконання тестів, підтримує всі ключові рушії браузерів (Chromium, WebKit, Firefox) та є кросплатформеним, що дає змогу запускати тести в операційних системах Windows, Linux, macOS, а також емулювати мобільні браузери Android і iOS, включно з рідною мобільною емуляцією Google Chrome та Safari. Додатковою перевагою є наявність офіційних API для різних мов програмування (TypeScript, JavaScript, Python, .NET, Java), можливість перехоплення та аналізу мережевих запитів, а також підтримка Android WebView. Сукупність зазначених характеристик дозволяє розглядати Playwright як потужний універсальний інструмент для автоматизації браузерних сценаріїв у сучасних високодинамічних веб-застосунках.

Таблиця 2.2 – Порівняння інструментів Playwright, Selenium WebDriver та Cypress

Критерій	**Playwright**	**Selenium WebDriver**	**Cypress**
Тип застосунку	Веб	Веб	Веб (SPA)
Мова програмування	JS/TS, Python, C#, Java	Java, Python, JS, C#, Ruby	JS/TS
Швидкість виконання тестів	Дуже висока	Середня	Висока
Стабільність	Висока	Середня (флейки при складних UI)	Висока
Автоматичні очікування	Так	Ні	Так
Підтримка мобільних тестів	Частково (WebKit)	Ні	Ні
Підтримка реальних пристроїв	Частково	Через сторонні сервіси	Ні
Паралельний запуск	Так	Так (через Selenium Grid)	Так
Простота налаштування	Висока	Середня	Висока
Підтримка локаторів	CSS, текстові локатори	CSS, XPath	CSS
Прозорість дебагінгу	Висока	Середня	Дуже висока
Використання у CI/CD	Дуже широка	Широка	Дуже широка
Основні переваги	Швидкість, стабільність	Кросбраузерність, велика спільнота	Стабільність, простота, автоматичні wait-и
Основні недоліки	Немає мобільної автоматизації	Флейки, старий протокол	Тільки JS/TS, тільки веб

Далі проведемо огляд і порівняння інструментів, що застосовуються для автоматизованого тестування мобільних застосунків (табл 2.3):

Appium розглядається як де-факто стандарт у цій сфері, оскільки підтримує платформи Android та iOS, базується на WebDriver-протоколі та не потребує внесення змін до вихідного коду продукту. Як інструмент із відкритим вихідним кодом, Appium дає можливість виконувати ті самі тестові сценарії як на

емуляторах і симуляторах, так і на фізичних пристроях, включно з використанням хмарних середовищ. Фреймворк працює через нативні засоби автоматизації, надані постачальниками платформ, тому не вимагає компіляції спеціалізованих версій застосунків або сторонніх бібліотек. Крім того, Appium розширює стандартний WebDriver-протокол додатковими API-методами, що оптимізують автоматизацію мобільних сценаріїв [16].

У проєкті AUTO.RIA цей інструмент було обрано основним через необхідність підтримки двох мобільних платформ одночасно, велику кількість складних E2E-сценаріїв, важливість запуску тестів на реальних пристроях та можливість інтеграції з інфраструктурою Appium Device Farm. До ключових переваг Appium належать кросплатформеність, доступ до нативних локаторів Android (UIAutomator2) та iOS (XCUITest), сумісність із поширеними інструментами екосистеми Java (TestNG, Maven, Allure) та підтримка паралельного виконання тестів. Водночас фреймворк має й певні обмеження: складність конфігурації XCUITest та нижчу швидкість порівняно з нативними середовищами автоматизації, такими як Espresso та XCTest.

Нативні інструменти мобільної автоматизації вирізняються вищою швидкістю та стабільністю, але позбавлені кросплатформенності.

Espresso є офіційним фреймворком для тестування інтерфейсу Android-додатків, інтегрованим у платформу Android та орієнтованим на написання швидких, точних і надійних UI-тестів із використанням простого API. Його застосування передбачає доступ до вихідного коду проєкту та тісну взаємодію з внутрішніми механізмами Android, що забезпечує мінімальну кількість флейкових сценаріїв і високу стабільність виконання тестів [17].

Аналогічним нативним рішенням для екосистеми iOS є XCTest - фреймворк, що входить до складу Xcode та підтримує створення юніт-, UI- та перформанс-тестів для застосунків, розроблених на Swift чи Objective-C. XCTest надає базові засоби для структурування тестових класів через XCTestCase, а також механізми ініціалізації та очищення середовища перед і після виконання тестів, що сприяє підвищенню точності та повторюваності результатів [18].

Попри очевидні переваги нативних фреймворків у швидкості та надійності, вони не є універсальними, оскільки обмежені однією платформою. Саме тому Appium, що забезпечує кросплатформеність і дозволяє використовувати єдину кодову базу тестів для Android та iOS, залишається оптимальним вибором для мультиплатформних мобільних застосунків, таких як AUTO.RIA.

Таблиця 2.3 – Порівняння інструментів Playwright, Selenium WebDriver та Cypress

Критерій	**Appium**	**Espresso**	**XCTest**
Тип застосунку	Мобільні Android/iOS	Android	iOS
Мова програмування	Java, Python, JS, C#, Ruby	Java/Kotlin	Swift/Objective-C
Швидкість виконання тестів	Низька–середня	Дуже висока	Дуже висока
Стабільність	Середня	Дуже висока	Дуже висока
Автоматичні очікування	Частково	Так	Так
Підтримка мобільних тестів	Так	Так	Так
Підтримка реальних пристроїв	Так	Так	Так
Паралельний запуск	Так	Так	Так
Простота налаштування	Низька	Середня	Середня
Підтримка локаторів	ID, XPath, iOS predicates	ID	Accessibility ID
Прозорість дебагінгу	Середня	Висока	Висока
Використання у CI/CD	Широка	Середня	Середня
Основні переваги	Мобільність, кросплатформеність	Стабільність, швидкість	Стабільність, інтеграція з iOS
Основні недоліки	Низька швидкість	Тільки Android	Тільки iOS

У сучасних мобільних застосунках значна частина функціональності забезпечується через взаємодію з бекенд-сервісами, що генерують сотні API-запитів, тому тестування API є невід'ємною складовою забезпечення якості.

Для таких перевірок використовуються різноманітні інструменти, серед яких особливе місце посідають Postman у поєднанні з Newman, Rest-Assured та Karate (табл 2.4).

Postman є зручним графічним середовищем для створення, організації та ручного виконання API-запитів, тоді як Newman виконує роль інструменту командного рядка, який дає змогу автоматизувати запуск створених у Postman колекцій та інтегрувати їх у процеси безперервної інтеграції. Така зв’язка дозволяє поєднати простоту створення тестів у візуальному інтерфейсі з можливістю масштабованого та «headless» виконання на серверних середовищах [19].

Rest-Assured є Java-бібліотекою для тестування REST API, яка значно спрощує процес формування HTTP-запитів і перевірки відповідей. Завдяки синтаксису, наближеному до BDD-підходу, цей інструмент забезпечує лаконічний і виразний спосіб опису очікуваних результатів, підтримує валідацію статус-кодів, заголовків та структурованих даних у форматах JSON та XML і надає готові методи для взаємодії з REST-сервісами. Така функціональність робить Rest-Assured одним із найпоширеніших рішень у Java-екосистемі для автоматизації API-перевірок [20].

Karate є сучасним фреймворком з відкритим кодом, який поєднує можливості API-тестування з простотою написання сценаріїв, притаманною Gherkin-подібним DSL-мовам. На відміну від традиційних Java-бібліотек, Karate дозволяє створювати тести без необхідності програмування мовою Java, оскільки логіка запитів та валідацій описується у формі поведінкових сценаріїв. Такий підхід робить інструмент доступним навіть для тестувальників без глибокого досвіду програмування[21].

У межах проєкту AUTO.RIA, як свідчить проведений аналіз, API-перевірки активно застосовуються для тестування JSON-фільтрів, що дає можливість

оперативно виявляти логічні розбіжності між серверною та клієнтською частинами та підвищує загальну ефективність тестового процесу.

Для такого роду перевірок, на проєкті AUTO.RIA використовується інструмент Rest-Assured.

Таблиця 2.4 – Порівняння інструментів Postman + Newman, Rest-Assured (Java) та Karate.

Критерій	**Postman + Newman**	**Rest-Assured (Java)**	**Karate**
Тип інструменту	GUI/API колекції + CLI runner	Java фреймворк	DSL фреймворк
Сценарії тестування	Функціональні, smoke, sanity	Функціональні, інтеграційні	Функціональні, інтеграційні
Мова / підхід	JSON + візуальні колекції	Java	Gherkin-like DSL
Поріг входу	Дуже низький	Середній	Середній
Швидкість виконання	Середня	Висока	Висока
Підтримка CI/CD	Так (через Newman)	Відмінна	Відмінна
Можливість рефакторингу	Обмежена	Висока	Висока
Робота з JSON/XML	Легка, але обмежена логіка	Дуже гнучка (JsonPath, Hamcrest)	Вбудована
Підтримка performance-тестів	Ні	Ні	Частково
Паралельність	Обмежена	Так	Так
Основні переваги	Простота, популярність, візуальність	Потужність, гнучкість, Java	Мінімум коду, BDD-формат
Основні недоліки	Не придатний для великих фреймворків	Потрібно знати Java	Обмежена нішевість

Сучасні мобільні застосунки взаємодіють із великою кількістю бекенд-сервісів і обробляють сотні API-запитів, що зумовлює необхідність формування

збалансованої стратегії тестування. Відповідно до актуальних рекомендацій WQR 2023, оптимальне тестове покриття передбачає переважання unit- та API-тестів, частка яких має становити приблизно 60–70%, тоді як на UI та end-to-end перевірки відводиться 30–40% [22]. У межах проєкту AUTO.RIA основний акцент автоматизації був спрямований саме на UI-тестування, оскільки інтерфейс мобільного застосунку характеризується високою складністю взаємодії, значна частина бізнес-логіки реалізована безпосередньо через UI, а більшість критичних дефектів виникали саме на цьому рівні. Такий підхід дозволив зосередити зусилля на тих модулях, що мають найбільший ризик впливу на кінцевий користувацький досвід, забезпечивши підвищення стабільності та передбачуваності роботи продукту.

2.2. Аналіз зовнішнього та внутрішнього середовища та аналіз зацікавлених сторін (Stakeholder Analysis)

Зовнішнє середовище організації включає економічні, технологічні, соціальні та конкурентні фактори, що впливають на ефективність реалізації проєкту автоматизації тестування (табл 2.5).

Таблиця 2.5 – Аналіз факторів зовнішнього середовища (PEST-аналіз)

Категорія	Фактори	Вплив на проєкт	Оцінка впливу (0-5)
Політичні	Законодавча стабільність, підтримка IT-галузі	Сприятливі умови для експорту IT-послуг, зростання довіри до аутсорсингу	4
Економічні	Курс валют, ціни на енергоресурси, інфляція	Збільшення операційних витрат, потреба в оптимізації ресурсів	3
Соціальні	Попит на онлайн-послуги, рівень цифровізації населення	Підвищений попит на мобільні застосунки, активне використання AUTO.RIA	5
Технологічні	Розвиток мобільних платформ, хмарних сервісів, CI/CD	Сприяє ефективній реалізації автоматизації тестування	5

Узагальнюючи результати аналізу, можна стверджувати, що зовнішнє середовище є сприятливим для реалізації проєкту, оскільки поєднання технологічного прогресу, зростання ринку мобільних сервісів і гнучких методів управління створює оптимальні умови для впровадження автоматизації тестування.

«Agile-тестування - це не окремий етап, а безперервна діяльність, інтегрована у життєвий цикл розробки програмного забезпечення»[23].

Для визначення сильних і слабких сторін організації застосовано метод SWOT-аналізу (табл 2.6), який дозволяє поєднати внутрішні можливості компанії з зовнішніми факторами впливу.

Таблиця 2.6 – SWOT-аналіз AUTO.RIA у контексті впровадження автоматизації тестування

Сильні сторони (S)	Слабкі сторони (W)
- високий рівень технічної експертизи команди.	- високе навантаження на QA-команду через часті релізи.
- використання сучасних фреймворків і CI/CD інструментів.	- обмежене покриття автотестами (до початку проєкту — 0%).
- Agile-культура, швидке ухвалення рішень.	- залежність від людського фактору при регресії.
- підтримка керівництва у розвитку автоматизації.	- недостатня документація для старих модулів.
Можливості (O)	Загрози (T)
- масштабування автотестів на всі мобільні платформи.	- часті зміни в API та UI інтерфейсах, що потребують оновлень тестів.
- інтеграція автотестів у CI/CD і DevOps-процеси.	- конкуренція за фахівців на ринку праці.
- зниження витрат на ручне тестування.	- ризик технічної заборгованості через швидкий темп релізів.

Згідно з отриманими результатами, компанія має сильний технічний потенціал для успішного впровадження автоматизації тестування. Основні ризики пов’язані із частими змінами у продукті та потребою підтримувати актуальність тестових сценаріїв, що компенсується Agile-гнучкістю команди.

Реалізація проєкту автоматизації впливає на декілька ключових груп стейкголдерів (табл 2.7): команду тестування, розробників, менеджмент і кінцевих користувачів. Оцінка їхніх очікувань і рівня впливу наведена нижче.

Таблиця 2.7 – Аналіз стейкголдерів проєкту

№	Стейкголдер	Роль у проєкті	Очікування	Рівень впливу
1	Product Owner	Визначає пріоритети фіч і релізів	Підвищення швидкості релізу, стабільність продукту	Високий
2	QA-команда (1 Automation + 4 Manual)	Реалізація автотестів і мануальних перевірок	Зменшення навантаження, повторюваність сценаріїв	Високий
3	Dev-команда	Інтеграція автотестів у пайплайн	Менше збоїв після деплою	Середній
4	Керівництво компанії	Забезпечує ресурси та підтримку	Підвищення ROI, зниження витрат QA	Високий
5	Користувачі AUTO.RIA	Отримувачі кінцевого продукту	Стабільна робота додатку, відсутність критичних багів	Непрямий, але важливий

2.3. Кейс-стаді: впровадження автоматизації тестування в Agile-командах

Кейс-стаді дозволяють на практичних прикладах продемонструвати, яким чином застосування інструментів автоматизації впливає на ефективність процесів у сучасних продуктових командах.

Одним із найбільш досліджених прикладів є підхід компанії Spotify, де, за даними публікації «Quality Engineering Productivity at Spotify» (QE Unit), модель організації роботи базується на squad-структурі, у межах якої кожна команда несе повну відповідальність за якість створюваного продукту, а QA-функція інтегрована в команду розробки. Така модель дає можливість швидко реагувати на зміни вимог і забезпечувати стабільність процесів навіть за умов високої інтенсивності релізів [24]. Додаткові матеріали, що стосуються підходу Spotify до тестування у масштабі («Testing at Scale»), підкреслюють необхідність

побудови потужної та гнучкої тестової інфраструктури, здатної підтримувати часті оновлення, складні інтеграційні сценарії й різнорівневі тестові активності. У зовнішніх джерелах зазначається, що Spotify застосовує комбіновану стратегію тестування, яка включає контрактні та API-тести, UI та E2E-перевірки критичних сценаріїв, а також регулярні smoke- і regression-цикли, що в сукупності мінімізує ризики функціональних деградацій при частих релізах [25]. Переваги такого підходу полягають у поєднанні відповідальності команд за повний життєвий цикл продукту, у багаторівневій стратегії тестування та в здатності інфраструктури масштабуватися разом зі зростанням навантаження та обсягів коду.

Іншим показовим прикладом є досвід компанії Uber. У блозі компанії, зокрема у матеріалі «Shifting E2E Testing Left at Uber», описано, як Uber інтегрував E2E-тестування в ранні етапи CI/CD-процесу, забезпечивши автоматичну перевірку кожної зміни коду або конфігурації у великій мікросервісній архітектурі [26]. Додаткові публікації на тему «Mobile Engineering at Uber» висвітлюють створення спеціалізованої мобільної тестової інфраструктури, здатної моделювати реальні мережеві умови, сценарії навігації та інші критичні фактори, що впливають на поведінку мобільних застосунків [27]. Окремої уваги заслуговує ініціатива DragonCrawl - інтелектуальна система автоматизації мобільного тестування, розроблена Uber з метою зменшення нестабільності тестів, підвищення охоплення та прискорення циклу зворотного зв’язку, що підтверджує прагнення компанії до інновацій у сфері тестування [28].

Сукупність описаних підходів Uber включає раннє виконання E2E-тестів (shift-left), широке охоплення backend-, API- та мобільної логіки, застосування масштабованої тестової інфраструктури з підтримкою паралельних прогонів та використання інноваційних технологій, таких як AI-підсилені алгоритми аналізу та генерації тестів. Ці принципи дозволяють компанії підтримувати стабільність великого глобального продукту, що функціонує на багатьох платформах та обслуговує мільйони користувачів.

Порівняння наведених кейсів зі станом тестування в AUTO.RIA показує подібну динаміку розвитку: до впровадження автоматизації регресійне

тестування тривало 6–8 днів, релізні цикли уповільнювалися, навантаження на команди Manual QA було надмірним, а кількість критичних дефектів у продакшені досягала восьми на спринт. Після запровадження системної автоматизації кількість регресійних днів скоротилася до 2–3, стабільність CI/CD досягла 97%, тестове покриття зросло до 65% ключових модулів, а кількість дефектів зменшилася на 75%. Економічний ефект також був суттєвим, що підтверджується розрахунками NPV, які досягли більш ніж +246 тис. грн. Сукупність цих показників демонструє, що автоматизація є дієвим механізмом підвищення продуктивності та якості в умовах швидких Agile-процесів і підтверджує успішність подібних практик у порівнянні зі світовими лідерами галузі.

Висновки до розділу 2

У другому розділі здійснено системний аналіз сучасних технологій автоматизації тестування та умов їх упровадження в Agile-командах на прикладі ТОВ «RIA.com Marketplaces». Дослідження організаційної структури, побудованої за Scrum-принципами, показало, що компанія має зрілу продуктову культуру, розвинену CI/CD-інфраструктуру та окремий напрям автоматизації, що формує сприятливі передумови для масштабного розвитку QA-процесів. Для високонавантаженого мобільного маркетплейсу рівня AUTO.RIA автоматизоване тестування є критично необхідним елементом забезпечення стабільності релізів і якості продукту.

У межах розділу виконано порівняльний аналіз інструментів автоматизації для веб-, мобільного та API-рівнів, у результаті якого обґрунтовано доцільність їх використання залежно від технологічного контексту. Показано, що сучасні браузерні фреймворки орієнтовані на швидкість і стабільність фронтенд-тестування, тоді як у сфері мобільної автоматизації Appium виступає оптимальним кросплатформенним рішенням для Android та iOS. Для перевірки серверної логіки в проєкті AUTO.RIA обґрунтовано вибір Rest-Assured як ефективного інструмента Java-екосистеми для автоматизації API-тестів і роботи з JSON-даними.

Окрему увагу приділено формуванню стратегії тестового покриття. На основі сучасних рекомендацій доведено доцільність домінування unit- та API-тестів, однак для мобільного застосунку AUTO.RIA акцент автоматизації обґрунтовано зміщено в бік UI-сценаріїв через високу залежність бізнес-логіки від інтерфейсу та значний вплив UI-дефектів на користувацький досвід. Такий підхід свідчить про адаптацію теоретичних моделей тестування до реальних потреб продукту.

У підрозділах, присвячених аналізу середовища, встановлено, що зовнішні умови є загалом сприятливими для розвитку автоматизації, а внутрішній потенціал компанії дозволяє ефективно реалізувати відповідні ініціативи. SWOT- та stakeholder-аналізи показали, що ключовими чинниками успіху є технічна експертиза команд, підтримка керівництва та інтеграція автоматизації в DevOps-процеси, тоді як основні ризики пов’язані з динамічними змінами продукту та потребою постійної підтримки тестів.

Аналіз кейсів провідних міжнародних компаній і порівняння з практикою AUTO.RIA підтвердили спільний вектор розвитку: перехід від тривалих ручних регресій до безперервного автоматизованого тестування, зростання тестового покриття, підвищення стабільності CI/CD та суттєве зменшення кількості дефектів у продакшені. Отримані результати засвідчують, що вибір інструментів і підходів до автоматизації в AUTO.RIA є технічно та економічно обґрунтованим і відповідає сучасним світовим тенденціям, створюючи надійне підґрунтя для практичної реалізації фреймворку автоматизації тестування, що розглядається у третьому розділі роботи.

РОЗДІЛ 3
МОДЕЛЬ ПРОЄКТУ ТА ЇЇ ФУНКЦІОНАЛЬНА ВЗАЄМОДІЯ

3.1. Опис проєкту та його функціональних вимог

Метою цього розділу є розроблення практичної моделі автоматизації тестування для мобільного застосунку AUTO.RIA, визначення організаційних, процесних і технічних передумов її впровадження, а також опис ролей учасників проєкту та їх взаємодії в межах Agile-середовища. Особлива увага приділяється формуванню тестової архітектури, інтеграції автоматизованих перевірок у CI/CD-процеси та забезпеченню відповідності тестування реальним бізнес-потребам продукту.

Реалізація проєкту здійснюється відповідно до принципів Scrum, який передбачає ітераційний підхід до розробки з використанням коротких спринтів тривалістю два тижні. У межах кожного спринту команда проходить повний цикл робіт — від уточнення вимог і планування до реалізації функціональності, тестування та демонстрації результатів. Щоденні стендап-зустрічі забезпечують синхронізацію між учасниками команди, дозволяють оперативно виявляти блокери та коригувати план виконання завдань. Планування спринтів, демо та ретроспективи виступають ключовими інструментами безперервного вдосконалення процесів і підвищення якості продукту. Для централізованого управління завданнями, моніторингу прогресу та фіксації результатів використовується система Jira, яка є основним інструментом підтримки Scrum-процесів у команді.

Мобільний застосунок AUTO.RIA є ключовим елементом цифрової екосистеми однойменного маркетплейсу та одним із найбільш відвідуваних автомобільних сервісів в Україні. Щодня застосунок обслуговує понад 1,5 млн користувачів, що зумовлює високі вимоги до його стабільності, продуктивності та якості користувацького досвіду. Функціональні можливості застосунку охоплюють широкий спектр бізнес-сценаріїв, зокрема пошук автомобілів із використанням багатокомпонентних фільтрів, перегляд і взаємодію з оголошеннями, роботу зі списком обраного, перевірку VIN-історії транспортних

засобів, створення та редагування оголошень, застосування пакетів публікацій, а також керування різними типами користувацьких акаунтів — приватними, корпоративними та акаунтами співробітників компаній. Така функціональна насиченість призводить до високої концентрації бізнес-логіки в інтерфейсі застосунку та значної кількості сценаріїв взаємодії, що суттєво ускладнює процес тестування і робить автоматизацію необхідною умовою стабільного розвитку продукту.

Основною метою впровадження автоматизації тестування в проєкті AUTO.RIA є підвищення стабільності релізів та передбачуваності якості продукту в умовах частих оновлень. Скорочення часу, необхідного для виконання регресійних перевірок, розглядається як критичний чинник, що безпосередньо впливає на швидкість випуску нових версій застосунку. Окрему увагу приділено зменшенню кількості критичних дефектів, які потрапляють у промислову експлуатацію, оскільки такі дефекти мають безпосередній вплив на довіру користувачів та репутацію сервісу. Важливою складовою мети також є забезпечення можливості безперервного тестування шляхом інтеграції автоматизованих перевірок у CI/CD-процеси, що дозволяє виявляти помилки на ранніх етапах розробки та мінімізувати вартість їх усунення.

На основі аналізу проблем, характерних для проєкту, та особливостей мобільного застосунку було сформульовано комплекс ключових вимог до системи автоматизації тестування. Однією з базових вимог є одночасна підтримка платформ Android та iOS, що зумовлено необхідністю збереження функціональної паритетності між мобільними версіями застосунку. Важливою умовою є використання стабільних локаторів для динамічних UI-компонентів, оскільки інтерфейс застосунку часто змінюється під впливом бізнес-вимог і дизайнерських оновлень. Передбачено можливість паралельного запуску тестів на кількох фізичних пристроях, що дозволяє суттєво скоротити тривалість регресійних циклів. Архітектура автоматизації повинна бути модульною, із чітким розділенням відповідальностей між Page Object, допоміжними шарами (Helpers) та тестовими сценаріями, що підвищує підтримуваність і масштабованість фреймворку. Окремо наголошено на необхідності гнучкої

роботи з API та JSON-фільтрами, а також на повній інтеграції з CI/CD-процесами Jenkins і GitLab CI з використанням єдиного звітного середовища на базі Allure.

Ефективна реалізація автоматизації тестування неможлива без правильно сформованої кросфункціональної команди (табл 3.1), у якій кожен учасник виконує чітко визначену роль. Структура команди в проєкті AUTO.RIA відповідає сучасним вимогам Agile-підходів і забезпечує баланс між розробкою, тестуванням та управлінням продуктом.

Таблиця 3.1 – Організаційна структура команди проєкту

Посада	Кількість	Основні обов'язки
Product Owner	1	Формує backlog, визначає пріоритети завдань, приймає результати спринтів
Scrum Master	1	Координує процес Scrum, усуває перешкоди в команді
Automation QA Engineer	1	Розробка фреймворку, створення та підтримка автотестів, інтеграція з CI/CD
Manual QA Testers	4	Виконання мануальних сценаріїв, досліджувальне тестування, перевірка нових функцій
Android Developers	2	Реалізація клієнтського функціоналу для Android
iOS Developers	2	Реалізація клієнтського функціоналу для iOS
DevOps Engineer	1	Налаштування Jenkins, Device Farm, інтеграції Allure та GitLab CI

Product Owner відповідає за формування бачення продукту, управління беклогом і визначення пріоритетів завдань. Саме ця роль забезпечує узгодженість між бізнес-цілями та технічною реалізацією, приймає результати спринтів і визначає критерії успішності функціональних змін. Scrum Master виконує роль фасилітатора процесу Scrum, слідкує за дотриманням принципів гнучкої розробки, допомагає команді усувати перешкоди та сприяє безперервному вдосконаленню процесів.

Automation QA Engineer є ключовою фігурою у впровадженні автоматизації тестування. До його обов'язків належить проєктування архітектури тестового фреймворку, розробка та підтримка автоматизованих тестових сценаріїв,

інтеграція тестів у CI/CD, а також аналіз стабільності та ефективності автотестів. Фахівці з ручного тестування зосереджуються на виконанні мануальних сценаріїв, дослідницькому тестуванні та перевірці нових функцій, забезпечуючи додатковий рівень контролю якості там, де автоматизація є недоцільною або економічно невиправданою.

Мобільні розробники Android та iOS відповідають за реалізацію клієнтського функціоналу відповідно до вимог Product Owner і результатів тестування, тоді як back-end розробники забезпечують стабільність серверної логіки та API, з якими безпосередньо взаємодіє мобільний застосунок. DevOps Engineer відіграє ключову роль у налаштуванні інфраструктури автоматизації, включно з Jenkins, Device Farm, інтеграцією Allure та GitLab CI, що дозволяє забезпечити безперервний цикл тестування та розгортання.

У процесі формування тестової архітектури було визначено низку ключових критеріїв якості, які характеризують очікуваний рівень надійності та ефективності автоматизованого тестування. Одним із базових показників є стабільність виконання UI-тестів на рівні не менше 95%, що є критичним для мобільних застосунків із динамічним інтерфейсом. Середній час виконання одного UI-сценарію обмежується 25 секундами, оскільки тривалість прогонів безпосередньо впливає на швидкість отримання зворотного зв’язку та можливість інтеграції тестів у CI/CD. Досягнення покриття ключових модулів на рівні щонайменше 60% розглядається як достатній компроміс між глибиною контролю та витратами на підтримку тестів. Кількість критичних дефектів у релізі повинна бути мінімальною і не перевищувати двох, що відповідає сучасним підходам до управління якістю у високонавантажених цифрових продуктах. Важливим критерієм також є повна відповідність реалізованих функцій визначеним Acceptance Criteria, що гарантує узгодженість між очікуваннями бізнесу та фактичними результатами розробки.

3.2. Архітектура системи та модель взаємодії функціоналу

Архітектура розробленого фреймворку автоматизації тестування ґрунтується на поєднанні патернів Page Object і Helper Layer, що забезпечує

модульність, повторне використання компонентів і можливість подальшого розширення тестового середовища. Подібний підхід відповідає рекомендаціям, наведеним у роботі Meszaros (2007), де зазначено, що якісно організовані тести повинні бути незалежними, добре читаними та передбачати чітку логіку очікуваних результатів [29]. Фреймворк містить низку ключових модулів, серед яких PageObject Layer, що відповідає за опис елементів інтерфейсу; Helper Layer, який інкапсулює бізнес-логіку сценаріїв і взаємодію між модулями; Driver Factory, який створює екземпляри WebDriver або AppiumDriver; а також спеціалізовані утилітарні модулі для роботи з JSON-структурами, API та файловими ресурсами. Конфігураційний шар забезпечує адаптивність фреймворку до різних пристроїв, середовищ та версій операційних систем, а Test Layer містить сценарії різного рівня – від smoke- і regression-тестів до nightly-прогонів. Фреймворк інтегрований з CI/CD-середовищами Jenkins і GitLab, що забезпечує можливість автоматизованого запуску тестів та формування звітності (схема 3.1).

```
src/
├── main/java/
│   ├── pageobject/
│   │   ├── MainPage.java
│   │   ├── SearchPage.java
│   │   └── NotepadPage.java
│   ├── helpers/
│   │   ├── SearchHelper.java
│   │   ├── NotepadHelper.java
│   │   └── OrientationHelper.java
│   ├── watchers/
│   │   ├── ScrollWatcherMapHelper.java
│   │   └── JsonFilterHelper.java
│   └── utils/
│       └── GmailReader.java
├── test/java/
│   ├── tests/ui/android/
│   └── tests/ui/ios/
└── resources/
    └── configs/
```

Схема 3.1 – Структура фреймворку автоматизації AUTO.RIA

Загальна модель взаємодії між модулями формується за принципом PageObject → Helper → TestNG, де PageObject надає доступ до елементів UI, Helper реалізує високорівневу логіку взаємодії з ними, а TestNG виступає основним інструментом запуску тестів та передачі параметрів. Робота з реальними пристроями забезпечується через Device Farm, що дозволяє розподіляти тести між декількома воркерами, вибирати необхідні версії операційних систем і виконувати паралельні прогони. Особливої уваги потребувала розробка стабільних локаторів для iOS та Android, оскільки структури accessibility tree та наявність динамічних індексів ускладнювали пошук елементів. Для усунення цих проблем було застосовано iOS Predicate, прив’язку до статичних компонентів, а також відносні XPath-конструкції та спеціалізовані утиліти локалізації.

При розробці моделі перевірки було визначено стратегію тестування, яка включає регулярне виконання smoke-сценаріїв перед злиттям коду, проведення повних регресій перед релізами, автоматизовані nightly-прогони, а також застосування API-тестів для валідації бізнес-логіки та обробки фільтрів. Значну частину сценаріїв складають UI-тести, орієнтовані на критичні бізнес-функції, зокрема авторизацію, пошук автомобілів, застосування понад сорока фільтрів, керування обраним, роботу з пакетами публікацій, створення оголошень та перегляд VIN-інформації. Окрему категорію становлять мультиакаунтні сценарії, що охоплюють взаємодію з профілями компаній, співробітників та приватних користувачів.

Таблиця 3.2 – Функціональні блоки мобільного застосунку які будуть покриті автотестами в ході роботи

№	Модуль	Приклади автоматизованих сценаріїв
1	Головний екран	Відображення основних елементів на головному екрані, та перевірка карток авто
2	Пошук авто	Перевірка фільтрів “Марка”, “Модель”, “Ціна”, “Пробіг”, “Перевірений VIN”

№	Модуль	Приклади автоматизованих сценаріїв
3	Обране (Notepad)	Додавання/видалення оголошень, перевірка синхронізації з профілем
4	Авторизація	Вхід через Google, OTP, Apple ID
5	Відеоповідомлення	Відтворення, перегляд кількості повідомлень, прокрутка списку
6	Додавання | редагування оголошення	Сетап фільтрів з характеристиками авто, та створення оголошення
7	Пакети публікацій	Перевірка умов, вигоди, бонусів для B2C/SMB користувачів

Інтеграція фреймворку у CI/CD забезпечується послідовністю (рис 3.1) Developer → GitLab Trigger → Jenkins Pipeline → Device Farm → Allure Reports → Slack Notification. Така модель дозволяє автоматично виконувати тести після кожної зміни коду, формувати централізовані звіти й оперативно повідомляти команду про результати прогону. Взаємодія Jenkins із Device Farm дає можливість масштабувати тестове навантаження та забезпечувати відтворюваність перевірок у різних середовищах.

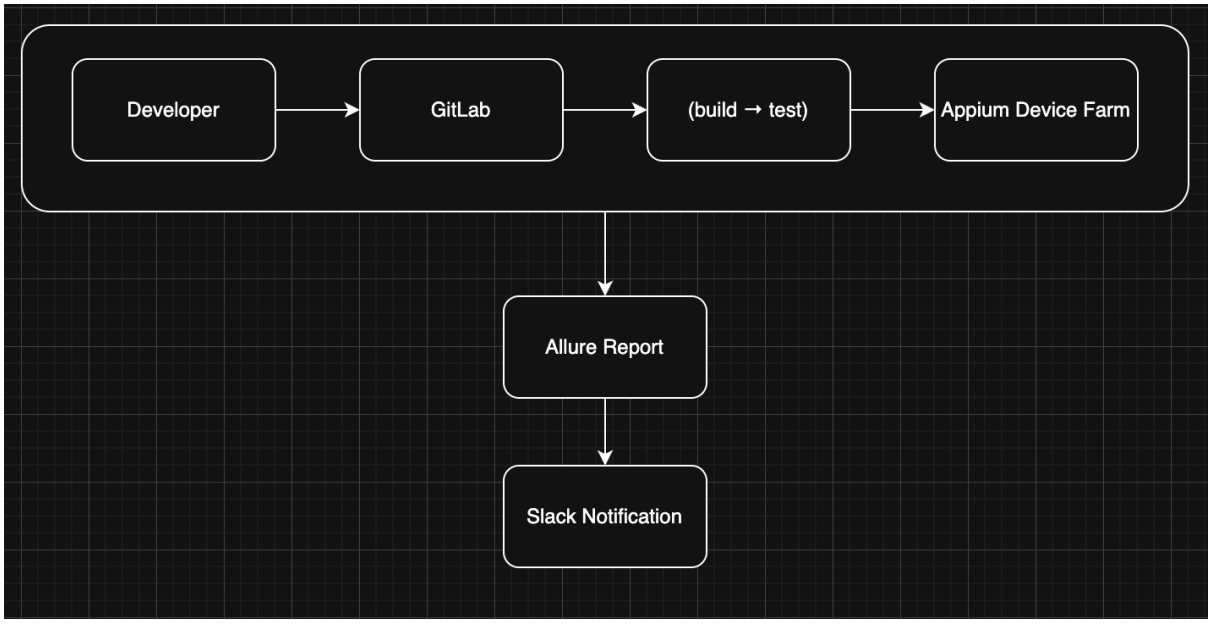

Рисунок 3.1 – Схема інтеграції Jenkins + Device Farm + Allure

Джерело: розроблено автором.

Етапи реалізації проєкту охоплювали шість спринтів по два тижні (табл 3.3., рис 3.2), під час яких здійснювали аналіз існуючого QA-процесу, проєктування архітектури фреймворку, розроблення базових модулів, інтеграцію з CI/CD, масштабування тестового покриття та оптимізацію продуктивності. Підсумковий етап включав збір метрик, розрахунок ROI та формування рекомендацій для подальшого розвитку автоматизації.

Таблиця 3.3 – Етапи проєкту та основні роботи

№ етапу	Назва етапу	Основні завдання	Результати
1	Аналіз поточного стану QA-процесів	Збір статистики тестувань, визначення вузьких місць, аудит сценаріїв	Документ “Current QA Workflow”
2	Проєктування архітектури фреймворку	Вибір стеку (Java, Appium, TestNG, Maven), створення Page Object-структури	Структура проєкту, базові класи Page/Helper
3	Реалізація базових модулів тестування	Реалізація core-методів, обробка локаторів Android/iOS, інтеграція Device Farm	Мінімальний життєздатний фреймворк (MVP)
4	Інтеграція в CI/CD	Налаштування Jenkins-пайплайну, Allure-репортів, GitLab webhooks	Автоматичний запуск автотестів при коміті
5	Масштабування тестів і покриття регресії	Створення тестів для ключових сценаріїв (пошук, фільтри, обране, підписки)	Покриття 60–70% критичних сценаріїв
6	Аналіз результатів і оптимізація	Оптимізація часу виконання, рефакторинг, збір метрик ROI	Фінальний звіт, рекомендації для масштабування

Для кожного спринту були визначені окремі ключові результати: від аналізу наявних сценаріїв TestRail і створення базових класів PageObject до розроблення тестів для критичних модулів і впровадження паралельного запуску на Appium Device Farm. Протягом роботи над проєктом здійснювалася щільна

інтеграція автоматизації з Agile-процесами команди. Автотести включалися до Definition of Done для кожної функції, а результати прогонів систематично фіксувалися в Jira через вкладені звіти Allure. Під час проведення ретроспектив оцінювалися стабільність тестів, тривалість прогонів і причини появи нестабільних сценаріїв.

Рисунок 3.2 – Діаграма Ганта проєкту автоматизації тестування в Agile-процесах.

Джерело: розроблено автором

За результатами реалізації етапів очікувалося отримання повноцінного кросплатформенного фреймворку автоматизації з інтеграцією у CI/CD-процеси, підвищення продуктивності QA-команди на 50–60%, а також суттєве зниження ручного навантаження під час регресійних циклів. Оцінка економічної ефективності була визначена як необхідний компонент управлінського рішення щодо інвестицій у розвиток фреймворку. Її виконання дозволило уточнити доцільність впровадження автоматизації, визначити економічну вигоду від скорочення тривалості тестових циклів і мінімізації дефектів, а також сформувати прогностичну модель ефективності проєкту в динаміці.

3.3. Економічна ефективність проєкту автоматизації QA

Оцінювання економічної ефективності інформаційних проєктів у сфері розробки програмного забезпечення є ключовим етапом обґрунтування управлінських рішень. У контексті сучасних вимог до якості IT-продуктів, мінімізації витрат та оптимізації ресурсів особливої актуальності набуває питання автоматизації тестування. Регресійне тестування, яке виконується багаторазово та має високу трудомісткість, є одним із найбільш витратних процесів у діяльності QA-команди. Тому перехід від виключно ручного тестування до змішаної моделі з використанням автотестів здатен суттєво вплинути на фінансові результати підприємства.

Метою цього розділу є всебічне дослідження економічної доцільності впровадження автоматизованого тестування в команді з п’яти QA-спеціалістів шляхом аналізу структури витрат, розрахунку чистої теперішньої вартості (NPV), дисконтування грошових потоків, визначення коефіцієнта співвідношення вигід і витрат (BCR), а також формування висновків щодо доцільності реалізації відповідного управлінського рішення.

Методологія оцінювання базується на класичних принципах інвестиційного аналізу та враховує:

- прогнозовані грошові потоки за проєктом;
- часову вартість грошей;
- рівень ризику;
- альтернативну вартість капіталу;
- вплив інфляції;
- участь людських ресурсів у процесах тестування.

Далі перерахуємо формули які будемо використовувати для оцінки економічної ефективності:

Чиста теперішня вартість (NPV):

$\text{NPV} = \sum_{t=0}^{n} \frac{CFt}{(1+k)t}$

де CF_t— грошовий потік у році t,

k— ставка дисконту.

Коефіцієнт дисконтування:

$$\text{DF(t)} = \frac{1}{(1+k)^t}$$

Ставка дисконту з урахуванням ризику та інфляції:

$$\text{k} = r_{\text{альт}} + r_{\text{ризик}} + \pi$$

Коефіцієнт співвідношення вигід і витрат (BCR):

$$\text{BCR} = \frac{\sum PV(\text{вигоди})}{\sum PV(\text{витрати})}$$

QA-команда складається з п’яти тестувальників, кожен із яких отримує 20 000 грн на місяць. Вартість одного робочого дня визначається як:

$$C_{\text{day}} = \frac{20000}{22} = 909{,}09 \text{ грн}$$

Регресійне тестування здійснюється двічі на місяць, а його повний цикл займає від 3 до 6 робочих днів, відповідно сукупні витрати на одного спеціаліста становлять, на початку вказується цифра :

$$C_{\text{reg}} = C_{\text{day}} \cdot [6 \text{ або } 12] = 5454{,}54 \text{ або } 10909{,}08 \text{ грн}$$

Для чотирьох тестувальників загальні витрати за пів року становлять:

$$C_{\text{6міс}} = [32724 \text{ або } 65454] \cdot 4 = \min: 130896, \max: 261816$$

Для зниження навантаження прийнято стратегічне рішення впровадити автоматизацію тестування. Один тестувальник проходить навчання вартістю 10 000 грн, після чого його заробітна плата збільшується до 30 000 грн. Компанія відводить на розробку автотестів шість місяців. Інвестиційні витрати становлять:

$$I_0 = 30000 \times 6 + 10000 = 190000 \text{ грн}=$$

Після впровадження автоматизації 60 % регресії виконується автотестами, решта 40 % - вручну, що зменшує трудовитрати до 2–6 днів на місяць.

Місячна економія: $E_{\text{міс}} = min: 14543{,}33,\ max: 21818{,}16$ грн

З урахуванням підвищеної заробітної плати автоматизатора: $E_{\text{чиста}} = E_{\text{міс}} - 10000 = min: 4543{,}33, max: 11818{,}16$

Річна економія:

$$E_{\text{рік}} = [\text{m}in: 4543{,}33, max: 11818{,}16] \times 12 = \min: 54519{,}96, \max: 141817{,}92$$

Ставка дисконту з урахуванням:

– альтернативної рентабельності: $r_{\text{альт}} = 0{,}10$

- ризику: $r_{\text{ризик}} = 0{,}03$
- інфляції: $\pi = 0{,}132$

Маємо:

$$k = 0{,}10 + 0{,}03 + 0{,}132 = 0{,}262$$

Визначено коефіцієнти дисконтування за формулою (табл 3.4):

DF(t) = $\frac{1}{(1+k)^t}$ для, t = 0, 1, 2, 3, 4, 5.

Таблиця 3.4 – Коефіцієнт дисконту (k=26.2%)

Коефіцієнт дисконту (k=26.2%)	
Рік t	Коефіцієнт дисконту 1/(1+k)^t (k=26.2%)
0	1.0
1	0.79239
2	0.62789
3	0.49753
4	0.39424
5	0.31239

Визначимо чисту теперішню вартість (NVP) для обох сценаріїв (табл 3.5):

NPV = $\sum_{t=0}^{5} * PV_t$

Мін-сценарій:

NPV_{min} = (−70 000) + 21 600,62 + 34 232,36 + 27 125,48 + 21 494,04 + 17 031,73 = 51 484,23 грн.

Макс-сценарій:

NPV_{max} = (−70000) + 56187,77 + 89045,59 + 70559,10 + 55910,54 + 44303,12 = 246 006,12 грн.

Таблиця 3.5 – Порівняльна таблиця зі значеннями PV у мінімальному та максимальному-сценаріях в гривнях

Рік (t)	PV (мін-сценарій), грн	PV (макс-сценарій), грн
0	−70 000,00	−70 000,00
1	21 600,62	56 187,77
2	34 232,36	89 045,59
3	27 125,48	70 559,10
4	21 494,04	55 910,54
5	17 031,73	44 303,12
∑ PV (NPV)	51 484,23 грн	246 006,12 грн

$$\text{BCR} = \frac{\sum PV(\text{вигоди})}{\sum PV(\text{витрати})}$$

У нашій інкрементальній моделі:

PV(витрати) = тільки PV0 за моделюванням (початкові інкрементальні витрати): ∑PV(витрати) = 70 000.

PV(вигоди) = сума всіх додатних PV (t=1..5).

Мін-сценарій:

∑PV(вигоди) = 21 600,62 + 34 232,36 + 27 125,48 + 21 494,04 + 17 031,73 = 121 484,23.

$BCR_{min} = \frac{121484,23}{70\ 000} = 1,735.$

Макс-сценарій:

∑PV(вигоди) = 56 187,77 + 89 045,59 + 70 559,10 + 55 910,54 + 44 303,12 = 316 006,12.

$BCR_{max} = \frac{316006,12}{70\ 000} = 4,514.$

Таблиця 3.6 –Порівняльна таблиця зі значеннями BCR у мінімальному та максимальному-сценаріях в гривнях

Показник	Мін-сценарій	Макс-сценарій
Сума приведених вигід, ∑PV(вигоди), грн	121 484,23	316 006,12
Приведені витрати, PV(витрат), грн	70 000,00	70 000,00
BCR = ∑PV(вигоди) / ∑PV(витрат)	1,735	4,514

Отже, у всіх сценаріях значення BCR значно перевищує 1, що свідчить про те, що вигоди перевищують витрати як у короткостроковому, так і в довгостроковому періодах. Враховуючи рівень інфляції в Україні (13,2 % р/р), базову інфляцію (11,4 %), місячну динаміку (+0,5 %) та прогноз Національного банку

України на кінець року (8,7–9,7 %), можна зробити висновок, що економічний ефект від автоматизації є стійким до макроекономічних коливань. Впровадження автотестів не лише зменшує прямі витрати на регресійне тестування у 2–3 рази, а й підвищує продуктивність команди, скорочує цикл

розробки, мінімізує людський фактор і створює значні довгострокові економічні переваги. Таким чином, автоматизація QA є обґрунтованим управлінським рішенням, що приносить фінансову вигоду та сприяє покращенню операційної ефективності підприємства.

Таблиця 3.7 – Підсумки (NPV, BCR, PV, Річна чиста вигода)

Підсумки (NPV, BCR)		
№	Показник	Значення
1	NPV (мін)	51 484.23
2	NPV (макс)	246 001.84
3	BCR (мін)	1.735
4	BCR (макс)	4.514
5	PV вигод (мін)	121 484.23
6	PV витрат (мін)	70 000.0
7	PV вигод (макс)	31 6001.84
8	PV витрат (макс)	70 000.0

Висновки до розділу 3

У третьому розділі дипломної роботи здійснено практичну реалізацію моделі автоматизації тестування для мобільного застосунку AUTO.RIA, що охоплює аналіз вимог, проєктування архітектури тестового фреймворку, впровадження автоматизованих сценаріїв, інтеграцію тестів у CI/CD-процеси та оцінку економічної ефективності запропонованого рішення. Отримані результати підтверджують можливість практичного застосування автоматизації як ключового інструмента підвищення якості та стабільності продукту в умовах Agile-розробки.

На основі аналізу функціональних можливостей і бізнес-процесів застосунку AUTO.RIA було сформовано вимоги до автоматизації та визначено критерії якості, що включають стабільність виконання тестів, достатній рівень покриття критичних сценаріїв, повторюваність перевірок і тісну інтеграцію з

CI/CD. Це дозволило обґрунтовано визначити межі автоматизації та розподіл відповідальності між автоматизованим і ручним тестуванням.

У межах розділу розроблено багаторівневу архітектуру тестового фреймворку на основі підходу Page Object та Helper Layer, що забезпечує модульність, розширюваність і зручність супроводу тестів. Особливу увагу приділено роботі з динамічними елементами інтерфейсу, стабілізації локаторів для iOS та Android, а також створенню допоміжних утиліт, які підвищують надійність і повторюваність тестових сценаріїв. Запропонована модель взаємодії компонентів дозволила мінімізувати дублювання коду та знизити витрати на підтримку автоматизації.

Також у розділі сформовано стратегію тестування, що поєднує UI- та API-перевірки, різні типи тестових прогонів (smoke, regression, nightly) і підтримує процеси планування в Agile-команді через Definition of Done. Інтеграція автоматизованих тестів у CI/CD із використанням Jenkins, GitLab та Device Farm забезпечила перехід до безперервного тестування, підвищила прозорість контролю якості та дозволила масштабувати перевірки на декілька пристроїв і платформ.

Важливою складовою третього розділу стала економічна оцінка ефективності автоматизації тестування. Проведені розрахунки показали, що впровадження фреймворку має позитивний фінансовий результат, характеризується високими показниками NPV, ROI та BCR і є економічно доцільним у середньостроковій перспективі. Це підтверджує, що автоматизація не лише покращує технічну якість продукту, а й забезпечує відчутний економічний ефект.

Узагальнюючи результати, можна зробити висновок, що розроблений фреймворк автоматизації є ефективним практичним рішенням для мобільного застосунку AUTO.RIA, яке відповідає вимогам Agile-процесів, підвищує стабільність релізів, скорочує витрати на тестування та створює основу для подальшого масштабування автоматизації в межах компанії.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було здійснено комплексне дослідження процесів забезпечення якості в команді AUTO.RIA та визначено ключові проблеми, що впливали на ефективність розроблення мобільного застосунку. Аналіз існуючого підходу до тестування показав значне перевантаження ручних тестувальників, тривалі затримки регресійних циклів, дублювання перевірок та недостатній рівень тестового покриття, що призводило до збільшення кількості дефектів на етапі післярелізної експлуатації. Виявлені проблеми стали підґрунтям для розроблення цілісної системи автоматизації тестування, адаптованої до особливостей Agile-процесів та технічних вимог проєкту.

У межах роботи була створена архітектура масштабованого фреймворку автоматизації тестування для мобільних застосунків Android та iOS, побудована на основі технологій Java, Appium та TestNG. Розроблена архітектура забезпечує модульність, структурованість і придатність до подальшого розширення, що дозволяє ефективно підтримувати різні функціональні модулі застосунку та адаптувати тестові сценарії до змін бізнес-логіки. Фреймворк було інтегровано у наявну інфраструктуру команди, включно з Jira, Confluence, GitLab, Jenkins, Device Farm та Allure, що сформувало єдину автоматизовану екосистему тестування всередині Scrum-процесу.

Побудований CI/CD-пайплайн забезпечив автоматичний запуск тестів після внесення змін до коду, автоматизоване формування звітності та оперативне інформування команди про результати. Така інтеграція дала можливість скоротити час зворотного зв’язку та підвищити контроль якості кожного релізного циклу. Поступове нарощення кількості тестів дозволило досягти автоматичного покриття більш ніж 60 % критичних бізнес-сценаріїв мобільного застосунку, що суттєво зменшило залежність команди від ручного тестування.

Важливою частиною дослідження став економічний аналіз ефективності впровадження автоматизації. Проведені розрахунки засвідчили позитивний фінансовий ефект, який проявляється у скороченні витрат на регресійне

тестування, зменшенні тривалості тестових циклів, підвищенні стабільності релізів та зниженні кількості критичних помилок після деплою. Розраховані інтегральні показники, зокрема NPV, ROI та співвідношення вигід до витрат, підтвердили економічну доцільність проєкту та продемонстрували високу рентабельність інвестицій у автоматизацію.

Практичне впровадження фреймворку змінило як технічні, так і організаційні процеси роботи команди. Зменшення рутини дозволило ручним тестувальникам зосередитися на дослідницькому та аналітичному тестуванні, підвищилася прозорість релізного процесу, а результати тестувань стали доступними керівництву в режимі реального часу. Автоматизація сприяла формуванню більшої відповідальності за якість на рівні всієї команди та поступовому переходу до моделі DevTestOps, у якій тестування є невід’ємним елементом розробки.

Отримані результати підтверджують, що впровадження автоматизації тестування є стратегічно виправданим рішенням як з технічного, так і з економічного погляду. Розроблений фреймворк довів свою здатність стабілізувати процеси тестування, покращити якість продукту, скоротити час виходу релізів та забезпечити компанії відчутний економічний ефект. Проєкт продемонстрував, що навіть у межах невеликої Agile-команди можливо створити ефективну, масштабовану та гнучку систему автоматизації, яка інтегрується в наявні процеси та підтримує подальший розвиток тестування в екосистемі RIA.com Marketplaces.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Beck, K. та ін. Manifesto for Agile Software Development [Електронний ресурс] / K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, D. Thomas. – 2001. – URL: http://agilemanifesto.org.

2. Automated testing for software development [Електронний ресурс] – URL: https://www.testdevlab.com/blog/automated-testing-for-software-development.

3. RIA.com Marketplaces. Офіційний сайт компанії [Електронний ресурс]. URL: https://www.ria.com

4. Manifesto for Agile Software Development [Електронний ресурс] : презентація. – Agile Alliance, 2001. – URL: https://www.agilealliance.org/wp-content/uploads/2018/05/free-manifesto.pdf.

5. Crispin L. The Agile Testing Quadrants [Електронний ресурс]. – 2024. – URL: https://lisacrispin.com/2024/10/11/the-agile-testing-quadrants.

6. Page Object Model in Selenium [Електронний ресурс] // BrowserStack Guide. – URL: https://www.browserstack.com/guide/page-object-model-in-selenium.

7. Fowler M. Is Quality Worth the Cost? [Електронний ресурс]. – 2019. – URL: https://martinfowler.com/articles/is-quality-worth-cost.html.

8. Different Appium Scroll Strategies [Електронний ресурс] // GeeksforGeeks. – URL: https://www.geeksforgeeks.org/software-testing/different-appium-scroll-strategies/.

9. Rest-Assured Docs: JSONPath validation [Електронний ресурс]. – URL: https://rest-assured.io/#json-path.

10. Fowler M. Microservice Testing: API Layer [Електронний ресурс]. – 2017. – URL: https://martinfowler.com/articles/microservice-testing/#api-layer.

11. Google Android Developers. Handling Orientation Changes [Електронний ресурс]. – URL: https://developer.android.com/guide/topics/resources/runtime-changes.

12. Fowler M. Continuous Integration [Електронний ресурс]. – ThoughtWorks, 2006. – URL: https://martinfowler.com/articles/continuousIntegration.html.

13. Selenium WebDriver Documentation [Електронний ресурс]. – Selenium Project. – URL: https://www.selenium.dev/documentation/webdriver/.

14. Огляд фреймворку Cypress [Електронний ресурс] // IT Praktik. – 2023. – URL: https://itpraktik.com/oglyad-frejmvorku-cypress/.

15. Playwright Documentation [Електронний ресурс]. – Microsoft, 2024. – URL: https://playwright.dev/.

16. Appium Documentation. Appium Automation for Mobile Platforms [Електронний ресурс]. 2023. URL: https://appium.io/docs/en/latest/

17. XCTest: обзор инструмента тестирования [Електронний ресурс] // AppTractor. – 2020. – URL: https://apptractor.ru/info/articles/xctest.html.

18. Postman & Newman: запуск колекцій [Електронний ресурс] // IT Notes Wiki. – 2023. – URL: https://www.it-notes.wiki/tools/postman-newman-run-collections/.

19. Rest-Assured: початок роботи [Електронний ресурс] // iQA Engineer. – 2024. – Режим доступу: https://iqaengineer.com/ua/start_rest_assured/.

20. Karate: инструмент для автоматизации тестирования API [Електронний ресурс] // Software-Testing.ru. – URL: https://software-testing.ru/library/testing/testing-tools/3084--karate.

21. World Quality Report 2023-24 [Електронний ресурс] : презентація. – Capgemini, 2023. – 18 с. – URL: https://www.capgemini.com/wp-content/uploads/2023/11/WQR_2023_FINAL_WEB_CG.pdf.

22. Qameta Software. Allure Framework Documentation [Електронний ресурс]. 2023. URL: https://docs.qameta.io/allure/

23. Quality Engineering Productivity at Spotify [Електронний ресурс] // Medium : QE Unit. – 2021. – URL: https://medium.com/qe-unit/quality-engineering-productivity-at-spotify-d263858135ab.

24. Testing at Scale [Електронний ресурс] // Qubika Blog. – 2023. – URL: https://qubika.com/blog/testing-at-scale/.

25. Shifting E2E Testing Left [Електронний ресурс] // Uber Blog. – 2022. – URL: https://www.uber.com/en-UA/blog/shifting-e2e-testing-left/.

26. Mobile Engineering at Uber [Електронний ресурс] // Pragmatic Engineer Blog. – 2022. – URL: https://blog.pragmaticengineer.com/mobile-engineering-at-uber/.

27. Generative AI for High-Quality Mobile Testing [Електронний ресурс] // Uber Blog. – 2023. – URL: https://www.uber.com/en-UA/blog/generative-ai-for-high-quality-mobile-testing/.

28. Meszaros G. xUnit Test Patterns: Refactoring Test Code. Addison-Wesley, 2007. 944 с

29. IEEE Software Journal. Measuring the ROI of Test Automation. IEEE Press, 2019.

30. Свєчніков С.С. Аналіз сучасних Agile-технологій та інструментів для автоматизації тестування / С. С. Свєчніков, Д. О. Балдик // Сучасний менеджмент організації: витоки, реалії та перспективи розвитку 2025: тези доповідей V Міжнародної Наукової конференції (17 квітня 2025 року). - Київ: Університет "КРОК", 2025 - URL: https://conf.krok.edu.ua/MMO/MMO-2025/paper/view/2782