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

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Магістр
(назва освітнього ступеня)

на тему: Методи автоматизованого тестування мобільних додатків при їх проектуванні з використанням технології Agile

Виконав: студент(ка) 6 курсу, групи СІм-61
спеціальності 123 «Комп'ютерна інженерія»
(шифр і назва спеціальності)

(підпис) Зарічний Н.Р. (прізвище та ініціали)

Керівник (підпис) Тиш Є.В. (прізвище та ініціали)

Нормоконтроль (підпис) Луцик Н.С. (прізвище та ініціали)

Завідувач кафедри (підпис) Осухівська Г.М. (прізвище та ініціали)

Рецензент (підпис) Литвиненко Я. В. (прізвище та ініціали)

Тернопіль
2022

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

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

Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Осухівська Г.М.*
(підпис) (прізвище та ініціали)
« » грудня 2022 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *магістр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студенту *Зарічному Нестору Романовичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Методи автоматизованого тестування мобільних додатків при їх проектуванні з використанням технології Agile*

Керівник роботи *Тиш Євгенія Володимирівна, к.т.н.*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*06*» *грудня 2022* року № *4/7-986*

2. Термін подання студентом завершеної роботи *20.12.2022 р.*

3. Вихідні дані до роботи *Методика тестування МД, архітектура МД, Методи автоматизованого тестування*

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Огляд теоретичних основ тестування мобільних додатків
2. Аналіз методик автоматизації тестування мобільних додатків на Agile проектах
3. Розробка і апробація методики автоматизованого тестування мобільних додатів на Agile - проекті
4. Охорона праці та безпека в надзвичайних ситуаціях
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Актуальність і мета дослідження.
2. Задачі дослідження, об'єкт і предмет, наукова новизна і практична цінність дослідження.
3. Технології тестування мобільних додатків
4. Методики автоматизації тестування мобільних додатків
5. Методика автоматизації тестування МД на Agile проекті
6. Алгоритм проведення тестування із застосуванням розробленої методики
7. Результати апробації методик для автоматизації тестування.
8. Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека в надзвичайних ситуаціях			
охорона праці			

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	*Аналіз сучасних методів і технологій автоматизованого тестування мобільних додатків*	*14.11.2022-20.11.2022*	*виконано*
2	*Аналіз методик автоматизації тестування мобільних додатків*	*20.11.2022 – 27.11.2022*	*виконано*
3	*Вибір інструментів тестування мобільних додатків*	*27.11.2022 – 04.12.2022*	*виконано*
4	*Розробка і апробація методики автоматизованого тестування мобільних додатків на Agile -проекті*	*04.12.2022– 08.12.2022*	*виконано*
5	*Охорона праці та безпека в надзвичайних ситуаціях*	*08.12.2022-12.12.2022*	*виконано*
6	*Оформлення пояснювальної записки і графічного матеріалу*	*12.12.2022-14.12.2022*	*виконано*
7	*Попередній захист кваліфікаційної роботи магістра*	*14.12.2022*	*виконано*
8	*Захист кваліфікаційної роботи магістра*	*21.12.2022*	

Студент _____ *Зарічний Н.Р.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Тиш Є.В.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Методи автоматизованого тестування мобільних додатків при їх проектуванні з використанням технології Agile // Кваліфікаційна робота // Зарічний Нестор Романович // ТНТУ, комп'ютерна інженерія, група СІм-61 // Тернопіль, 2022 // с. – 79, рис. – 22, табл. – 4, аркушів А1 – 8, додат. – 6 , бібліогр. – 26.

Ключові слова: Мобільний додаток, тестування, автоматизація, Agile.

Кваліфікаційна робота складається з вступу, чотирьох розділів, висновків, списку використаної літератури та додатку.

У першому розділі описано специфіку мобільних додатків, розглянуто методи та види тестування. Також описано специфіку тестування мобільних додатків на Agile - проектах.

У другому розділі описано сферу застосування автоматизованого тестування, методики та інструменти автоматизації тестування мобільних додатків. Розглянуто методику Scripting, що застосовується для підготовки автотестів на Agile -проектах.

Третій розділ присвячено розробці методики автоматизації тестування мобільних додатків, що проектуються за технологією Agile. У цьому ж розділі представлені результати апробації розробленої методики

У висновку містяться підсумки виконаної роботи.

Додатки містять тестові сценарії.

ABSTRACT

Methods of automated testing of mobile applications when they are designed using Agile technology // Master thesis // Zarichny Nestor Romanovych // TNTU, computer engineering, group СІм-61 // Ternopil, 2022 // p. – 79, fig. - 22, tab. - 4, sheets A1 - 8, add. – 6, bibliography. - 26.

Keywords: mobile application, testing, automation, Agile.

The qualification work consists of an introduction, four chapters, conclusions, a list of used literature and an appendix.

The first chapter describes the specifics of mobile applications, examines methods and types of testing. The specifics of testing mobile applications on Agile projects are also described.

The second chapter describes the scope of automated testing, methods and tools for automating testing of mobile applications. The scripting technique used for preparing self-tests on Agile projects is considered.

The third chapter is devoted to the development of a methodology for automating the testing of mobile applications designed using Agile technology. In the same section, the results of the approbation of the developed methodology are presented

The conclusion contains the results of the work performed.

Appendices contain test scenarios.

ЗМІСТ

ПЕРЕЛІК ОСНОВНИХ УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ І СКОРОЧЕНЬ

API	англ. Application Programming Interface прикладний програмний інтерфейс
BDD	англ. Behavior-driven development, розробка через поведінку
GUI	англ. graphical user interface графічний інтерфейс користувача
QA	англ. Quality Assurance забезпечення якості
ЖЦ	життєвий цикл
ІС	інформаційна система
ІТ	інформаційні технології
МД	мобільний додаток
ОС	операційна система
ПЗ	програмне забезпечення

ВСТУП

Актуальність теми. В даний час технологія Agile набирає все більшої популярності у сфері розробки програмного забезпечення, у тому числі - для мобільних пристроїв.

Для автоматизації проектування додатків із застосуванням гнучких методологій, найчастіше використовується методика Scripting. Вона підходить для автоматизації тестування API, але є неоптимальним рішенням для підготовки автотестів GUI. В результаті розробники змушені проводити тестування графічного інтерфейсу вручну.

Під час коротких ітерацій на Agile проекті можливо провести тільки базові перевірки GUI. Такий підхід допустимий для веб- та десктоп-додатків, які розраховані на роботу з обмеженою кількістю браузерів та платформ. На відміну від них, мобільні програми розробляються під різні мобільні платформи, версії операційних систем і конфігурації пристроїв. Через те, що тестування обмежується базовими перевірками, багато дефектів GUI потрапляють у фінальну версію програми та виявляються лиш кінцевими користувачами. Для мобільних програм така ситуація може призвести до отримання негативних відгуків від користувачів і, як наслідок, до комерційного провалу.

Таким чином, актуальність дослідження обумовлена необхідністю розробки методики, яка дозволить автоматизувати тестування API та графічного інтерфейсу мобільного додатка.

Метою кваліфікаційної роботи є дослідження та розробка методики автоматизації тестування мобільних додатків, що проектуються за технологією Agile.

Задачі кваліфікаційної роботи:

1. Проаналізувати існуючі методи та види тестування програмного забезпечення та оцінити можливість їх застосування для мобільних додатків.
2. Проаналізувати існуючі методики та інструменти автоматизації тестування та оцінити можливість їх застосування для мобільних додатків.

3. Розробити методику автоматизації тестування МД за технологією Agile.

4. Виконати апробацію та обґрунтувати застосування розробленої методики для підвищення ефективності тестування мобільних додатків.

Відповідно до цілей та завдань дисертаційної роботи визначено її об'єкт та предмет.

Об’єкт дослідження: методи автоматизованого тестування мобільних додатків, що проектуються за технологією Agile.

Предмет дослідження: розробка методики автоматизованого тестування мобільного додатку, розробленого за технологією Agile.

Методи дослідження: системний аналіз, методи тестування програмного забезпечення, експеримент. Метод візуалізації даних, що дозволяє наочно представляти отримані результати дослідження.

Наукова новизна дослідження полягає у розробці методики автоматизації, яка ґрунтується на комбінації двох існуючих методик для підготовки різних типів тестів, що дозволить підвищити ефективність процесу тестування мобільних додатків, що проектуються за технологією Agile.

Теоретична значущість полягає у розробці вдосконаленого способу проведення автоматизованого тестування мобільних додатків, розробленого за технологією Agile.

Практичне значення результатів кваліфікаційної роботи полягає у можливості застосування розробленої методики автоматизації для тестування мобільних додатків.

Публікації. Результати дослідження апробовано на X науково-технічній конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (7-8 грудня 2022 року) у вигляді тез конференцій.

1. Зарічний Н., Тиш Є. Автоматизація тестування мобільних додатків за технологією agile. Матеріали X науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі системи та технології» (7-8 грудня 2022 року). Тернопіль: ТНТУ. 2022. С. 27.

Структура роботи. До складу кваліфікаційної роботи магістра входить розрахунково-пояснювальна записка та графічний матеріал. Розрахунково-пояснювальна записка містить вступ, 4 розділи, загальні висновки, список використаної літератури і додатки. Обсяг роботи: розрахунково-пояснювальна записка – 79 арк. формату А4, графічна частина – 8 аркушів формату А1.

РОЗДІЛ 1
ОГЛЯД ТЕОРЕТИЧНИХ ОСНОВ ТЕСТУВАННЯ МОБІЛЬНИХ ДОДАТКІВ

1.1. Особливості проектування мобільних додатків

Мобільний додаток (МД) – програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях, розроблене для конкретної платформи [18].

Існує кілька типів МД [18]:

- нативні;
- браузерні (мобільні веб-додатки);
- гібридні.

Нативні МД створюються до роботи з конкретної платформою. Їхній код пишеться мовами програмування, які є «рідними» для мобільних платформ. Для Android це Java, для iOS – Swift або Objective-C. Такі МД фізично встановлюються на мобільний пристрій та поширюються через магазини мобільних додатків.

Браузерні МД - це оптимізовані версії веб-застосунків, спочатку розроблених для ПК. МД цього типу є мультиплатформенними – запускаються через браузер на мобільному пристрої з будь-якою операційною системою (ОС) та не використовують його програмне забезпечення (ПЗ).

Гібридні МД поєднують у собі риси перших двох типів - вони використовують веб-технології, але вимагають установки на пристрій та мають доступ до його функцій. Гібридні МД використовують вбудовану оболонку програми, яка містить веб-подання (web-view) для запуску веб-програми всередині програми для конкретної платформи.

Незалежно від типу, МД має ряд особливостей, які відрізняють його від веб- та настільних додатків. Далі будуть розглянуті ключові моменти [20, 25]:

- взаємодія з основними функціями пристрою-комунікатора. Смартфон - це, в першу чергу, телефон і жодні програми не повинні блокувати можливість прийняти/здійснити дзвінок, отримати/надіслати SMS повідомлення;

- робота з додатком у різних умовах (наприклад: зміна рівня зовнішнього освітлення, нестабільний зв'язок з Інтернетом, перемикання між Wi-Fi і 3/4G, витрата заряду батареї пристрою, переведення програми у фоновий режим);

- короткий цикл розробки. Це викликано необхідністю часто випускати оновлення для забезпечення сумісності з новішими моделями мобільних пристроїв та мобільних операційних систем;

- велика різноманітність мобільних пристроїв, розмірів та діагоналей екранів, версій ОС, на яких може бути використана програма, та графічних оболонок (для Android);

- сильна конкуренція серед виробників мобільного програмного забезпечення, через яку у користувачів формуються високі очікування.

Виходячи з цього, користувачі оцінюють якість мобільного додатка за такими критеріями як:

- коректне виконання заявлених у назві та описі завдань,

- інтуїтивна зрозумілість та хороша швидкість відгуку всіх елементів управління,

- безперебійна робота за будь-яких умов.

Незручність використання, несумісність з популярними моделями пристроїв, функціональні помилки, проблеми графічного інтерфейсу та інші недоліки призводять до негативних відгуків та різкого зниження рейтингу програми [19].

Тому якість МД – необхідна умова його затребуваності та конкурентоспроможності. Важливо виділити саме ті тести, які є найбільш критичними для конкретного додатка з метою скорочення витрат компанії та зниження ризику появи помилок.

Складність МД не дозволить провести всі можливі тести, тому слід використовувати пріоритети зон тестування, серед яких можна виділити найбільш критичні [26]:

- інтерфейс - необхідно переконатися, що всі елементи мають зручний розмір, в додатку немає порожніх екранів, МД підтримує стандартні жести;

- апаратні ресурси - потрібно ретельно перевіряти обробку проблемних

ситуацій (наприклад: установка програми за нестачі пам'яті, недостатній обсяг пам'яті для роботи програми в активному або фоновому режимі);

- перевірка різних версій ОС та дозволів екрану (наприклад: коректність відображення елементів МД на AMOLED- та retina дисплеї, у ландшафтній та портретній орієнтації, неможливість встановлення програми на пристрій з непідтримуваною версією ОС);

- реакція на зовнішні переривання - в першу чергу це вхідні дзвінки та СМС, перехід пристрою в сплячий режим, push -сповіщення інших програм, підключення додаткових пристроїв відключення та включення Wi-Fi та мобільного інтернету;

- зворотний зв'язок з користувачем - відгук елементів на дії користувача повинен бути зрозумілим та своєчасним, реакція кнопок на натискання повинна відповідати їхньому стану (активна, натиснена, заблокована), при спробі видалити дані повинні з'являтися запобіжні повідомлення з можливістю скасувати дії;

- платний контент - вартість повинна відповідати функціоналу, що надається, покупки не повинні губитися при оновленні додатка і так далі;

- локалізація - сюди відносяться максимальна кількість символів, які можна ввести в поля, що заповнюються, коректність перекладу, формат відображення дат і специфічних для конкретної мови символів;

- оновлення - основні перевірки - збереження даних користувача, функціонування урізаних версій програми, створених для роботи з більш старими версіями ОС;

- відповідність МД правилам та угодам конкретної ОС – необхідно перевіряти коректність назви та опису, формат інсталяційного файлу, підтримку вимог різних магазинів додатків (для Android) [18];

- обробка випадкових і непередбачуваних подій - мобільні пристрої часто виявляються в умовах, в яких отримують випадкову інформацію або випадкові дії (наприклад, коли відбувається розблокування пристрою, що лежить у кишені), тому МД має адекватно обробляти випадкове введення [18];

- імітація реальних умов використання - необхідно перевіряти роботу мобільного додатка за нестабільного зв'язку з інтернетом.

Перелік критичних зон може скорочуватися чи розширюватись залежно від специфіки конкретного МД. Наприклад, для програми, яка розробляється для використання в конкретній країні, не потрібно проводити тестування локалізації.

1.2. Методи та види тестування мобільних додатків

Базові принципи тестування сформульовані у класичних книгах із тестування [11, 13, 16]. Автори виділяють два підходи до тестування програмних продуктів - метод чорної скриньки та скляної (білої) скриньки.

Тестування методом білої скриньки спрямоване на перевірку внутрішніх аспектів роботи програми. При цьому метою є виявлення не синтаксичних помилок (для їхнього пошуку зазвичай використовується компілятор), а складніших для локалізації логічних дефектів.

Подібні перевірки дозволяють абстрагуватися від зовнішнього прояву помилок і виявити їхню причину, спростити пошук та діагностику прихованих проблем. Це необхідний етап тестування, але його недостатньо для оцінки якості МД.

При тестуванні методом білого ящика додаток досліджується в синтетичних умовах, без урахування впливу реального середовища виконання та у відриві від сценаріїв користувача. Для користувачів МД велике значення мають простота та зручність графічного інтерфейсу, дизайн екранів та інші зовнішні аспекти, які неможливо перевірити на рівні коду.

Крім того, при тестуванні методом білої скриньки фокус робиться на реалізованій функціональності, в результаті підвищується ймовірність пропустити нереалізовані вимоги.

Під час тестування чорною скринькою програма розглядається як об'єкт з невідомою внутрішньою структурою. Таким тестуванням займаються QA-фахівці. При цьому вони фокусуються не на коді, а на тому, як програма обробляє різні вхідні дані.

Підхід чорної скриньки застосовується лише за наявності відкритих

інтерфейсів МД - користувальницького та (або) програмного (API). Поведінка програми порівнюється з тим, що описано у вимогах до неї.

Метод чорної скриньки спрямований на пошук помилок, які мають зовнішні прояви. Це проблеми, пов'язані з функціоналом ПЗ, результатами обчислень, допустимими діапазонами даних, які можуть бути оброблені додатком. Робота внутрішніх компонентів системи перевіряється неявно за допомогою аналізу зовнішніх проявів дефектів .

Основний плюс такого підходу – взаємодія з програмою з позиції кінцевого користувача. З одного боку, це дозволяє зосередитися на перевірці функціоналу програми та виявити найбільш помітні та критичні проблеми у його роботі. З іншого - застосування чорної скриньки для тестування у чистому вигляді призводить до одностороннього бачення програмного продукту.

Методи чорного і білого ящика не є взаємовиключними - вони гармонійно доповнюють один одного і таким чином компенсують їх взаємні недоліки.

Тому в більш сучасних книгах з тестування [15, 23] пропонується використовувати комбінований метод – тестування сірою скринькою. Це підхід, що поєднує елементи перших двох методів. З одного боку, тестувальник використовує патерни поведінки кінцевого користувача, з іншого - частково знає, як влаштований бек-енд програми і активно застосовує цю інформацію.

Метод сірої скриньки широко застосовується для тестування МД. Він передбачає однакову увагу як до зовнішньої (графічний інтерфейс користувача), так і до внутрішньої (взаємодія з сервером) частин програми.

Тестування методом сірої скриньки дозволяє локалізувати дефекти одночасно і за принципом зв’язку з певними рядками коду, і за послідовністю дій користувача в результаті яких відтворюється проблема. Ці відомості допомагають швидше і точніше оцінювати дефекти за серйозністю та пріоритетом, визначати масштаб їх прояву та прогнозувати наслідки їх виправлення.

У межах цього методу перевірки роботи МД зазвичай застосовуються різні види тестування. Залежно від мети їх можна поділити на три групи [7]:

- функціональні,

- нефункціональні,
- пов’язані зі змінами.

Функціональне тестування передбачає перевірку того, що ті чи інші функції реалізовані в даній версії програми та працюють відповідно до вимог.

Нефункціональне тестування спрямоване на перевірку нефункціональних особливостей МД. Нижче розглянуто основні види нефункціонального тестування.

Інсталяційне тестування призначене для виявлення проблем, що впливають на встановлення МД. Включає перевірку таких ситуацій як установка в новому середовищі (нова модель пристрою або версія ОС), оновлення поточної версії, зміна встановленої версії на старішу, повторне встановлення після невдалої спроби, видалення програми.

Тестування зручності використання (usability) призначено для визначення, наскільки функціонал програмного продукту зрозумілий користувача і подобається йому. МД можна вважати зручним для використання, якщо користувач робить те, що йому здається найбільш правильним і при цьому у нього не виникає жодних питань та сумнівів у своїх діях [5, 15]. При роботі з МД не повинно виникати «тупикових ситуацій», коли користувач не повністю розуміє, що відбувається у додатку та (або) не може контролювати те, що відбувається.

Тестування інтерфейсу передбачає перевірку того, наскільки коректно працює інтерфейс користувача і його компоненти [12].

Тестування безпеки перевіряє, чи здатний МД протистояти зловмисним діям – спробам доступу до конфіденційної інформації чи інтеграції шкідливого коду.

Тестування сумісності дозволяє визначити здатність програми працювати в конкретному оточенні (модель і конфігурація пристрою, версія ОС, обладнання, що підключається - зовнішня клавіатура, гарнітура). Сюди ж можна віднести перевірку коректного функціонування програми у різних орієнтаціях конкретного пристрою, перевірку модулів програми, їх дизайну щодо відповідності певній ОС [22].

Тестування продуктивності зводиться до вивчення того, з якою швидкістю додаток реагує на навантаження різного характеру та інтенсивності - звичайну, зростаючу та стабільно високу (стресову).

Тестування локалізації проводиться для перевірки якості та коректності адаптації МД до роботи певною мовою, а також з урахуванням культурних особливостей (наприклад формат дати, специфічні для мови символи, одиниці ваги, відстані).

За ступенем автоматизації тестування поділяється на ручне та автоматизоване. У першому випадку всі перевірки проводяться вручну, у другому – тест-кейси частково чи повністю виконує спеціалізований інструментальний засіб. При цьому тестувальник не виключається із процесу повністю – він займається розробкою тестових сценаріїв, підготовкою даних, аналізом та оцінкою результатів виконання, підготовкою звітів про знайдені помилки.

Також тестування мобільних додатків може класифікуватися за суб'єктом виконання. Коли продукт уже зібраний воєдино, але ще надто «сирий» для демонстрації зовнішнім користувачам, всередині компанії-розробника проводиться альфа-тестування.

До виконання залучаються як професійні випробувачі, так і співробітники інших відділів організації. Альфа-тестування застосовується для перевірки життєздатності ідеї проекту та відстеження найбільш критичних помилок у коді МД [7].

Бета-тестування має на увазі активне залучення замовника та (або) кінцевих користувачів. Продукт передається бета-тестерами, коли він вже досить стабільний, але може містити дефекти, виявити які можливі тільки при використанні МД в реальних умовах. Бета-тестування може бути організовано в закритому або відкритому форматі.

При закритому бета-тесті доступ до програми отримує обмежену кількість учасників, у відкритому може взяти участь будь-хто. У першому випадку простіше контролювати коло осіб, у яких є доступ до додатку. У другому - з'являється можливість охопити більш широку аудиторію, отримати більший обсяг зворотного зв'язку та забезпечити наближене до реального навантаження на серверну частину МД.

Тестування мобільної програми проводиться після кожного внесення змін -

виправлення дефектів, додавання або виключення функціональності.

Димне тестування є невеликим набором перевірок, що дозволяє переконатися, що після складання нового або виправленого коду програму можна встановити, запустити і використовувати за призначенням.

Регресійне тестування проводиться після того, як у додатку або середовищі виконання були зроблені зміни для підтвердження того, що реалізована раніше функціональність працює коректно.

Тестування збірки дозволяє переконатися, що випущена версія відповідає критеріям якості, необхідним для початку тестування.

Санітарне тестування є підвидом регресійного тестування та спрямоване на перевірку того, що конкретна функція працює згідно з вимогами, заявленими у специфікації.

Залежно від глибини перевірок і функціональне, і нефункціональне тестування можна розділити на два підвиди - тестування критичного шляху і розширене.

Тестування критичного шляху націлене дослідження тієї частини функціональності, яка «використовується типовими користувачами у повсякденній діяльності» [15].

У випадку з тестуванням МД сюди можна віднести встановлення та запуск програми, авторизацію, переходи між основними екранами та події, пов'язані з виконанням базової функції конкретного МД – збереження та обробка певного виду даних, відправка певних запитів тощо.

На рис. 1.1 показано суть тестування критичного шляху [15].

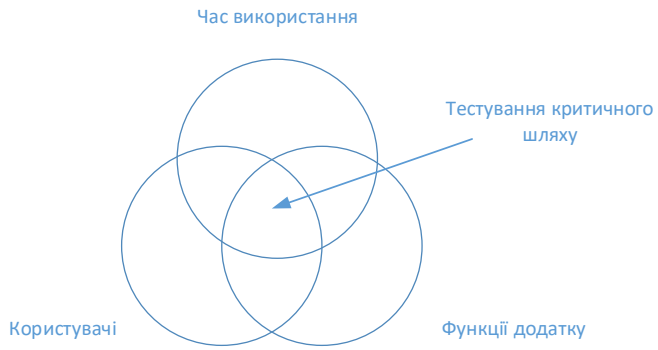

Рис. 1.1. Суть тестування критичного шляху

Розширене тестування, навпаки, спрямоване на перевірку всієї описаної у вимогах функціональності. При виконанні тестів враховується пріоритет функціональності – спочатку перевіряється важливіша, потім менш важлива. За наявності достатньої кількості часу та людських ресурсів тестуються навіть випадки з найнижчими пріоритетами.

Також у рамках розширеного тестування можуть перевірятися нетипові та малоймовірні тест-кейси та сценарії використання властивостей та функцій програми, більш поверхово проведених під час тестування критичного шляху.

Залежно від характеру використовуваних кейсів, перевірка всіх описаних вище видів може відноситися як до позитивного, так і до негативного тестування.

Перше передбачає дослідження роботи МД відповідно до інструкції, тобто, з виконанням коректних дій та введенням валідних даних. Друге, навпаки, спрямоване на перевірку того, як програма обробляє помилкові дії користувача. Наприклад, надсилання невалідних даних або пропуск обов'язкових кроків.

Виходячи зі сказаного вище, метод сірої скриньки відповідає завданням мобільного тестування - його застосування дозволяє приділити однакову увагу фронт-енд та бек-енд частинам програмного продукту, його інтерфейсу та продуктивності. Але в той же час цей метод передбачає проведення великого обсягу перевірок і, як наслідок, потребує суттєвих витрат часу.

1.3. Специфіка тестування мобільних додатків в Agile проектах

Більшість мобільних додатків створюються в умовах, що постійно змінюються, коли потрібно регулярно доопрацьовувати функціонал, оновлювати дизайн, переписувати існуючий код для сумісності з новими версіями мобільних ОС, а також відповідності зростаючим вимогам користувачів.

Тому в розробці мобільних додатків все більшої популярності набирає технологія Agile, яка передбачає швидку реалізацію функціоналу, його часте оновлення і тісне співробітництво із замовником.

Agile або гнучка методологія розробки (agile software development) - група

методологій розробки програмного забезпечення, заснованих на ітеративній поетапній розробці, де вимоги та рішення розвиваються через співпрацю між командами що розробляють програмний продукт [17].

Ключові принципи методології описані в Agile Manifesto [19] – програмному документі спільноти «Agile Alliance», розробленому у лютому 2001 року. В основі Agile лежать 12 принципів. Суть даних принципів полягає в тому що під час роботи над проектом важливими є люди і їх співпраця, пріоритетом є вимоги замовника, передбачена готовність до нового, незважаючи на попередньо затверджений план і саме головне це швидкість виходу продукту на ринок.

На рис. 1.2 [14] показана різниця в організації процесу тестування ПЗ із застосуванням каскадної методології та Agile.

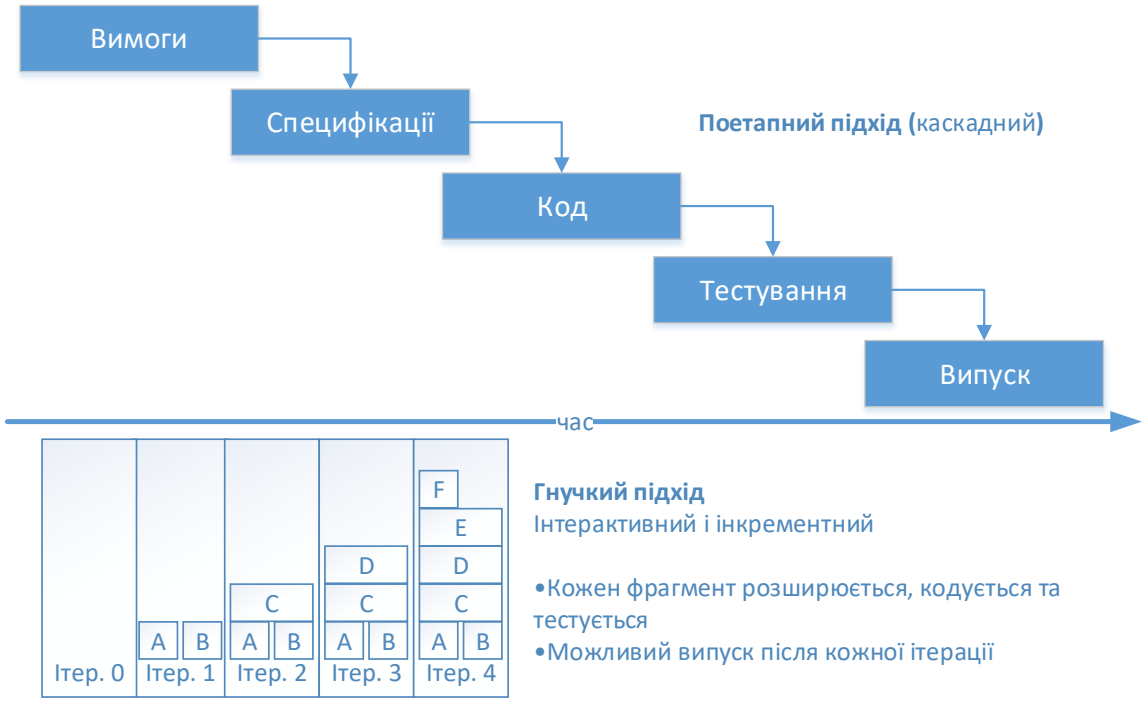

Рис. 1.2. Відмінності в організації процесу тестування при застосуванні традиційної та гнучкої методологій

При застосуванні каскадної методології програма випускається одночасно з повним набором функціоналу. Якщо розробка ведеться з використанням гнучкої методології – функціонал програми розробляється та тестується поетапно, у кожній

ітерації до існуючого функціоналу додається новий фрагмент.

Agile -підхід до тестування має на увазі наступні зміни [14] в роботі QA - команди:

- тестування перестає бути ізольованою фазою у створенні ПЗ та активно застосовується на всіх стадіях життєвого циклу продукту – починаючи з планування. Таким чином, QA - фахівці можуть своєчасно виявляти найпростіші у виправленні помилки (неточності, неоднозначні деталі та інші проблеми документації), скласти уявлення про програмний продукт задовго до написання коду та виявити його потенційно слабкі місця;
- обсяг тестової документації скорочується до мінімуму: на зміну докладним тест-кейсам приходять більш високорівневі та універсальні тест-плани та чек-листи. Формат чек-листа дозволяє не розписувати перевірки досконально, за рахунок цього документацію простіше підтримувати в актуальному стані, а у роботі тестувальника збільшується частка дослідницького тестування;
- на всіх стадіях розробки підтримується зворотний зв'язок між фахівцями з тестування та рештою членів команди (розробники, бізнес-аналітики, дизайнери, проектний менеджер). Завдяки цьому у кожного з учасників проекту формується повніше і всебічно бачення продукту;
- швидка віддача від тестування - знайдені баги підлягають оперативному виправленню, що дозволяє підтримувати «чистоту коду» та уникнути накопичення застарілого та складно підтримуваного коду;
- тестування – невід'ємна частина критерію готовності: ступінь готовності ПЗ визначається з урахуванням кількості, пріоритету та серйозності виявлених проблем. Наприклад, критерієм готовності МД до випуску може бути відсутність у продукті дефектів з пріоритетом вище «незначного» (Minor) або «середнього» (Normal).

Застосування тестування на всіх стадіях ЖЦ додатку створює умови для реалізації методології BDD (скор. від англ. Behavior-driven development, дослівно «розробка через поведінку»), яка заснована на поєднанні у процесі розробки суто технічних інтересів та бізнесу [21].

Відповідно до цієї методології, для спілкування членів проектної команди використовується предметно-орієнтована мова. Основу якої представляють конструкції з природної мови, зрозумілі нефахівцеві. Мовні конструкції описують поведінку програмного продукту та очікувані результати.

Під очікуваним результатом розуміється поведінка ПЗ, що становить цінність для бізнесу. Для його опису використовується специфікація поведінки (англ. behavioral specification), яка має наступну структуру:

1) заголовок - опис бізнес-мети у умовному способі;

2) короткий опис користувальницької історії із зазначенням ролі користувача;

3) один або кілька сценаріїв, кожен з яких розкриває ситуацію користувача поведінки.

Такий підхід дозволяє формувати єдине розуміння продукту усіма учасниками процесу розробки. За рахунок цього BDD дозволяє сформулювати вимоги до продукту, які зроблять його технічно реалізованим, корисним з погляду бізнесу та зручним для кінцевого користувача.

Взаємодія за принципами BDD показано рис.1.3 [14].

Рис.1.3. Застосування методології BDD розробки ПЗ

Методологія BDD протиставляється традиційному підходу до організації взаємодії всередині проектної команди, у якому джерелом помилок у роботі продукту часто стають різні трактування вимог бізнес-аналітиками, розробниками та тестувальниками.

Традиційний підхід показано рис.1.4 [14].

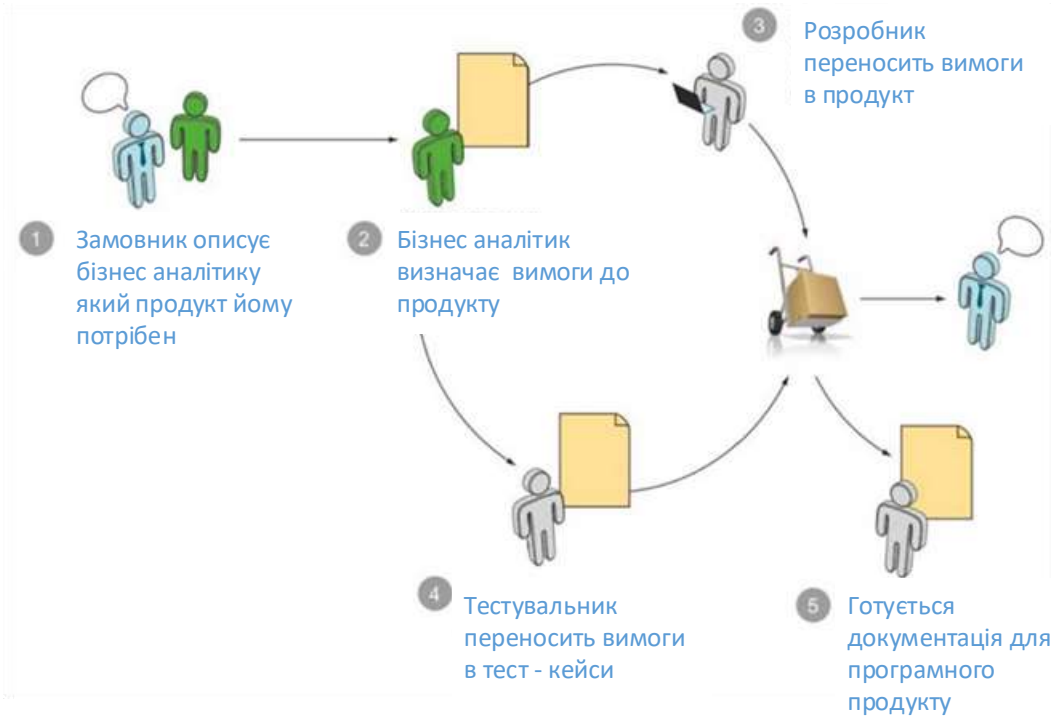

Рис. 1.4. Традиційний підхід до організації взаємодії усередині проектної команди

Перелічені принципи та зміни включають в усі стадії розробки програми:

- проектування. Відмова від вичерпного опису всіх аспектів програмного забезпечення дозволяє швидко вносити зміни до документації без шкоди для цілісності загальної ідеї програми. Основні завдання QA - фахівця на етапі проектування - тісне спілкування з бізнес-аналітиками, вивчення проектної документації та підготовка тестових сценаріїв;

- технологія. За рахунок частого випуску нових версій, МД постійно еволюціонує та не втрачає актуальності коли виходять нові моделі смартфонів та планшетів, мобільні ОС та «оболонки». Тестувальники виконують ручну перевірку,

що спрямована на пошук суттєвих (блокуючих, критичних та важливих) помилок. Паралельно створюються автоматизовані функціональні тести;

- завершення ітерації. Активне спілкування із замовником дозволяє своєчасно отримувати зворотний зв'язок та, як наслідок, оперативно усувати помилки, забезпечувати узгодженість роботи всіх модулів додатка. На цій стадії робота QA - фахівців зосереджена на відтворенні та аналізі проблем, які знайдені та зафіксовані на етапах проектування та розробки.

Отже, застосування гнучкої методології дозволяє побудувати процес розробки МД з урахуванням специфіки додатків цього.

Висновки до розділу 1:

1. Мобільні програми мають ряд особливостей, які визначають специфіку їх тестування.

2. При тестуванні МД використовується метод сірої скриньки, тому що він передбачає однакову увагу як до зовнішньої (графічний інтерфейс користувача), так і до внутрішньої (взаємодія з сервером) частин програми.

3. Застосування методу сірої скриньки у тестуванні МД передбачає проведення великого обсягу перевірок протягом короткого циклу розробки.

4. Розробка МД із застосуванням гнучкої методології дозволяє враховувати специфіку додатків на всіх етапах розробки.

5. Застосування тестування всіх стадіях ЖЦ програмного продукту на Agile - проекті створює умови реалізації методології BDD, що дозволяє формувати єдине розуміння продукту усіма учасниками процесу розробки ПЗ.

РОЗДІЛ 2
АНАЛІЗ МЕТОДИК АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ МОБІЛЬНИХ ДОДАТКІВ НА AGILE ПРОЕКТАХ

2.1. Автоматизоване тестування та сфера його застосування

Всупереч назві автоматизоване тестування не є повністю автономним процесом і має на увазі активну участь людини. Крім запуску автотестів, життєвий цикл автоматизованого тестування включає оцінку вигоди застосування автотестів, пошук інструментальних засобів, написання скриптів та підготовку тестових даних, аналіз результатів виконання автотестів.

Виходячи зі сказаного вище, ефективність роботи команди тестування багато в чому залежить від того, які саме завдання було вирішено автоматизувати та як ця автоматизація була проведена [9, 9, 10, 12, 24].

Будь-який додаток (у тому числі - мобільний) можна умовно розділити на три рівні:

- рівень компонентів (Unit layer) включає код МД (наприклад, змінні, функції, методи, бібліотеки);
- рівень функціональності (Functional layer) - це бізнес-логіка програми, тобто практичний результат роботи, для отримання якого він створений;
- рівень графічного інтерфейсу користувача (GUI layer) включає компоненти МД, видимі кінцевому користувачеві (наприклад, екрани, кнопки списки).

Автоматизація приносить максимальний ефект процесу тестування, якщо вона зачіпає всі рівні МД, що тестується. На рис. 2.1 показано, які типи автотестів відповідають усім рівням програми.

Автоматизація починається на рівні компонентів.

Автоматизовані юніт-тести (Unit Tests) створюються для кожної нової можливості доданої в додаток. Вони дозволяють швидко виявляти помилки у коді.

Наступний ступінь – рівень функціональності. Йому відповідають сервісні автотести (Service Tests), спрямовані на тестування класів, що утворюють

компонент у складі нового функціоналу. Такі тести запускаються лише після успішного завершення юніт-тестування.

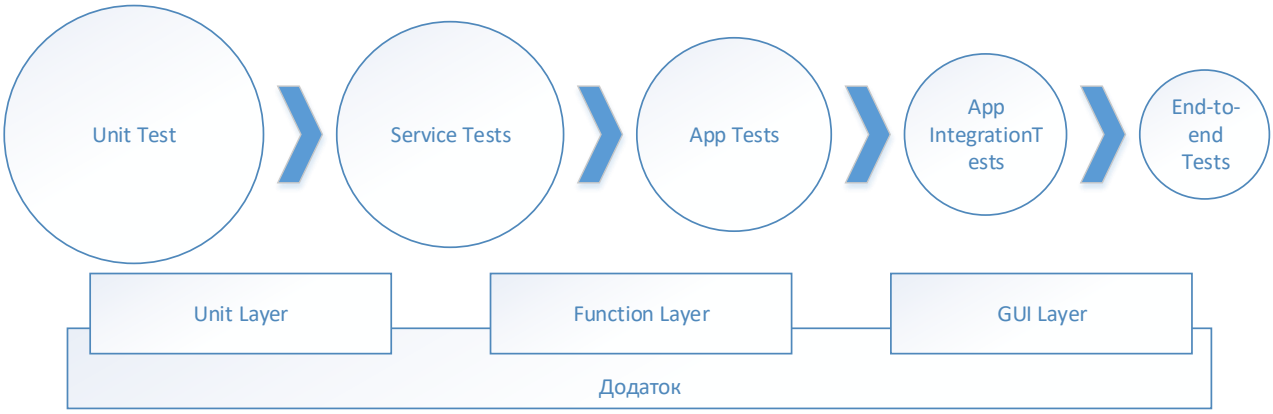

Рис. 2.1. Типи автотестів для різних рівнів програми

Це тести, призначені для перевірки функціональності "у чистому вигляді". Зазвичай вони запускаються лише на рівні API без використання графічного інтерфейсу. Якщо потрібно протестувати взаємодію із зовнішніми сервісами, які не можуть гарантувати надання даних або з якихось причин недоступні, використовуються емулятори зовнішніх сервісів.

На рівні інтерфейсу виконуються автотести для перевірки програми в цілому (App Tests), інтеграції програм (App Integration Tests) та повні сценарні тести (End-to-End Test).

Автотести для перевірки програми в цілому відрізняються глибиною опрацювання та великим обсягом. Їхня мета - переконатися в коректності роботи всього додатку. Якщо програма має об'ємний функціонал, для тестування вона може бути розбита на кілька окремих програм, що надають користувачеві різні можливості.

В цьому випадку також застосовуються автотести інтеграції додатків, призначені для перевірки взаємодії «додатків всередині додатка» і коректності перемикання між ними.

Повні сценарні тести являють собою автоматизовані GUI -тести, які запускаються для всієї системи, відтворюють типові шляхи користувача або повні сценарії взаємодії.

2.2. Етапи автоматизованого тестування

У загальному вигляді процес автоматизації тестування виглядає так:

Крок 1: визначення цілі автоматизації;

Крок 2: вибір існуючої методики автоматизації чи розробка нової;

Крок 3: вибір інструментів автоматизації;

Крок 4: підготовка тестової інфраструктури (наприклад бібліотек коду, систем звітності, баз даних);

Крок 5: написання та налагодження набору автотестів для основної архітектури МД (зазвичай це перевірки для проведення приймального тестування);

Крок 6: підготовка більш детальних автотестів для тестування критичної функціональності, регресійного тестування;

Крок 7: одноразовий або багаторазовий (залежно від цілей розробки конкретного набору автотестів) прогін автотестів;

Крок 8: аналіз автоматично сформованих звітів, оцінка якості МД, що тестується;

Крок 9: підтримка автотестів – перевірка адекватності логіки нових тестів, оновлення параметрів у раніше створених тестах, адаптація тестової інфраструктури для перевірки змінених версій МД або іншої програми;

Крок 10: за потреби - передача автотестів замовнику.

Застосування описаного сценарію дозволяє здійснити автоматизацію тестування з урахуванням особливостей конкретного МД.

2.3. Методики автоматизованого тестування мобільного додатку

Перш ніж вибрати інструментальний засіб, необхідно визначитися з методикою автоматизованого тестування.

В даний час застосовуються чотири методики:

- запис та відтворення скриптів,
- написання сценарію,

- параметризоване тестування,
- тестування на основі ключових слів.

Запис та відтворення скриптів (Record and Play) - передбачає використання утиліт для запису дій користувача у додатку. Програма перетворює запис на код і генерує автотести, які згодом виконуються без участі людини. Основні переваги - простота застосування, не потрібні знання у галузі програмування. Головний недолік - необхідність створення нових автотестів після внесення будь-яких змін до інтерфейсу МД. Зазвичай ця методика активно застосовується при димовому тестуванні і для одноразового прогону однотипних тестів у різних середовищах.

Написання сценарію (Scripting) – методика полягає у використанні тестових сценаріїв, написаних мовами, спеціально розробленими для автоматизації тестування ПЗ [20]. Основна відмінність від методики «запис та відтворення скриптів» – код тестів створюється людьми, а не програмою.

Це потребує серйозних часових та фінансових витрат, оскільки розробкою займаються програмісти чи тестувальники з високою кваліфікацією. З іншого боку – такі тести простіше підтримувати та масштабувати, оскільки при написанні коду вручну автор враховує можливі зміни у назвах структурних елементів МД, а також може погодити ці зміни з рештою учасників проектної команди.

Параметризоване тестування (Data-driven testing). Це методологія створення скриптів та їх верифікації на основі даних, що містяться у базі даних чи сховищі. Використовується у випадках, коли потрібно реалізовувати однотипні перевірки різних комбінацій вхідних даних.

Тестування на основі ключових слів (Keyword-based testing) - методика написання автоматизованих тестових сценаріїв, що використовує файли, що подаються на вхід, не тільки для зберігання тестових даних і очікуваних результатів, але і ключових слів, що належать до тестованого додатку. Ключові слова інтерпретуються спеціальними процедурами, що викликаються з керуючого сценарію даного тесту [17]. Тести, підготовлені у межах цього підходу, є не програмний код, а послідовність дій зі своїми параметрами. Як і перший підхід дозволяє створювати автотести тестувальникам, які не мають навичок

програмування.

При цьому автотести під на основі ключових слів є стабільнішими і легшими в підтримці, ніж тести типу Record and Play. Для опису Keyword based тестів використовуються ключові слова, для реалізації застосовуються фреймворки.

При виборі методики автоматизації тестування для конкретного продукту потрібно враховувати такі фактори: особливості предметної галузі та тип МД, а також методологію розробки програми.

2.4. Інструменти автоматизації тестування мобільних додатків

Автоматизація тестування призводить до ускладнення схеми цього процесу. Схема ручного тестування включає три компоненти - тести, додаток і тестувальник, який виступає як посередник між ними. Він перетворює кроки тест-кейсів у дії з тим чи іншим інтерфейсом програми (API, GUI, Net та інші).

Тому, щоб автоматизувати тести, недостатньо використовувати єдиний інструмент, функції якого обмежуються запуском. Також необхідні інструментальні засоби, які формуватимуть тест-кейси, взаємодіятимуть з інтерфейсами програми, що тестується, та іншими інструментами автоматизації [8, 21].

Тому схема автоматизованого тестування включає комплекс інструментів. Залежно від функціональних можливостей та механізму роботи їх можна розділити на кілька груп [21]:

- драйвери,
- надбудови,
- фреймворки запуску,
- комбайни.

Вибір конкретних інструментів залежить від типу МД, тестування якого потрібно автоматизувати. Існують крос-платформні засоби автоматизації та спеціалізовані програми, призначені для роботи з конкретною платформою. Перші застосовуються для автоматизації МД будь-яких типів, другі – для автоматизації нативних та гібридних МД.

На рис. 2.2 [21] показано зміну схеми тестування за його автоматизації, а також позначено місце кожного з перерахованих вище типів інструментів у процесі автоматизованого тестування.

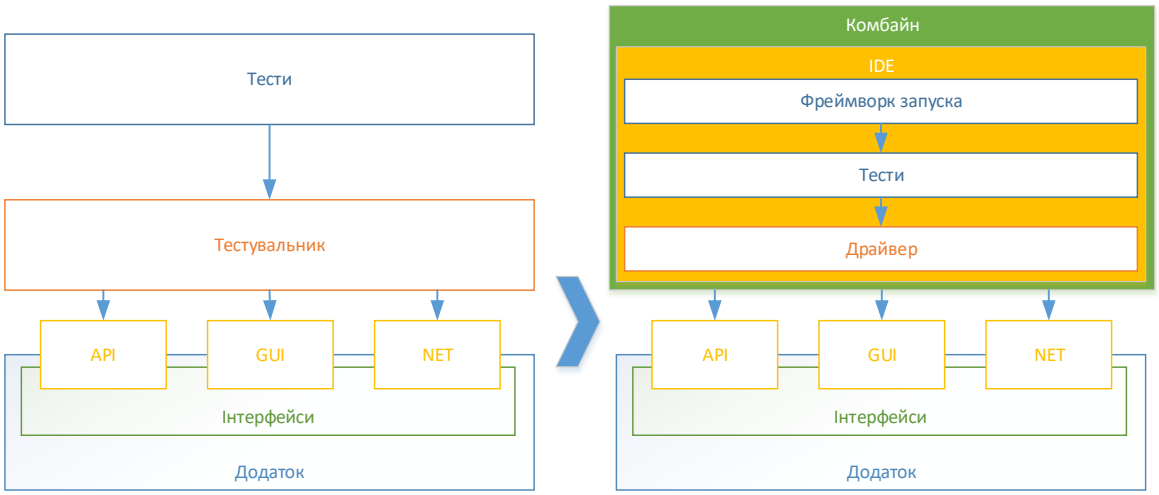

Рис 2.2. Зміна схеми тестування при його автоматизації

У цьому контексті драйвер - це «програма, що видає себе за апаратне забезпечення та забезпечує зв'язок між програмами за стандартним інтерфейсом. Іншими словами, цей інструмент надає API для одного з інтерфейсів програми» [21].

Наприклад, драйвер для графічного інтерфейсу сприймає команди через API і відправляє їх в МД, що тестується, де команди перетворюються на відповідні дії з графічними елементами.

При тестуванні нативних та гібридних МД використовуються GUI- драйвери XCTest для iOS, UIAutomator та Espresso для Android. Для створення Автотести графічного інтерфейсу для браузерних МД застосовується Selenium WebDriver.

XCTest підтримує версії iOS від 9.0 та вище. Для створення автотестів потрібен доступ до вихідного коду МД, що тестується. Тести для цього драйвера пишуться тими ж мовами, що і iOS -додатки - Swift та Objective-C.

У базовій комплектації XCTest тести можна запустити лише на симуляторі, але за допомогою сторонніх утиліт це можна зробити і на реальних пристроях. За допомогою рекордера, вбудованого в інтерфейс XGode, XCTest дозволяє записувати GUI -тести, знаходити графічні елементи та їх властивості.

UIAutomator дозволяє працювати з версіями Android , починаючи з Android 4.3 (API level 18) і не вимагає впровадження свого коду в проект. Цей інструмент підтримує такі можливості Android як поворот екрана, зняття скріншотів та натискання на кнопку Home. Завдяки цьому UIAutomator широко застосовується для функціонального End-to End тестування.

Утиліта UIAutomator Viewer взаємодіє з програмами, запущеними на емуляторі або на реальному пристрої, отримує дані про GUI -елементи і показує їх локатори.

Espresso підтримує версії Android починаючи з Android 2.3.3 (API level 10) і призначений для тестування методом білої скриньки. До того ж утиліта не може самостійно працювати з системою Android та іншими додатками. Для запуску драйвера потрібен доступ до вихідного коду МД. Espresso можна використовувати разом із UIAutomator, поєднуючи в одному тесті команди обох інструментів.

Надбудовою називається програма, яка взаємодіє з додатком через один або кілька драйверів, підвищує зручність їх використання або розширює можливості. Надбудова може мати такі функції, як:

- модифікація поведінки драйвера без зміни API (наприклад: валідація даних, очікування виконання дії протягом заданого часу);
- підвищення рівня абстракції API шляхом спрощення складних команд, реалізації альтернативних стилів програмування тощо;
- уніфікація драйверів через надання єдиного інтерфейсу для них, наприклад, для використання одного і того ж коду тестів для використання з програмою на iOS та Android.

Найбільш відомими надбудовами є Appium та Calabash.

Appium підходить для автоматизованого тестування МД незалежно від платформи, типу програми та версії системи. Програма підтримує описані раніше драйвера XCTest, UIAutomator та Espresso, надає можливості:

- писати автотести будь-якою популярною мовою програмування, у тому числі - «нерідними» для МД мовами;
- створювати та запускати тести для будь-яких типів МД (нативні, гібридні,

веб);

- працювати з будь-яким тестовим фреймворком;
- тестувати програми без доступу до коду.

Надбудова Calabash представлена у вигляді двох інструментів – Calabash iOS та Calabash Android. Обидві підтримують мови Ruby та JRuby. Calabash Android працює без підступу до коду МД, а утиліта для iOS потребує підключення Calabash framework.

Фреймворк запуску (далі - фреймворк), на відміну від драйверів та налаштувань, не є прошарком між тестами та МД. Це програма, яка служить для формування та запуску набору тестів, а також збору результатів виконання.

Крім цього, до завдань фреймворку входить угруповання, упорядкування та розпаралелювання виконуваних тестів, формування звітів про їх виконання. Найбільш популярні фреймворки – xUnit та Cucumber.

Комбайн - це утиліта, що поєднує в собі драйвери, фреймворки та можливості розробки. В автоматизованому тестуванні найчастіше використовуються комбайни Xamarin, UITest, Squish та Ranorex.

Xamarin є сервісом для розробки (в основному мовою C#) та тестування МД. Цей комбайн має інструменти автоматизації тестування, а також власні ферми мобільних пристроїв.

Ranorex дозволяє тестувати МД на емуляторах та реальних пристроях. Тести для нього пишуться мовами C# та VB.NET. Доступний лише для Windows, має рекордер для тестів.

Squish має власний рекордер і IDE. Мови для написання тестів: Ruby, Perl, Python, JavaScript.

2.5. Методика автоматизації тестування в Agile проектах та оцінка її застосовності для мобільних додатків

Технологія Agile передбачає частий випуск нових версій програми, це визначають специфіку тестування на Agile проектах. На підготовку тестової

документації, виконання тестів та аналіз результатів виділяється набагато менше часу, ніж на проектах із традиційним підходом до розробки.

Тому більшість проведених тестувальниками перевірок пов'язані з реалізацією нових функціональних можливостей, розробка і тестування виконуються паралельно. Це дозволяє від початку ітерації закласти забезпечення якості та знизити ризик того, що нові можливості порушать роботу існуючого функціоналу.

Одним із способів оптимізації процесу тестування стає застосування автотестів. При цьому основною метою автоматизації стає можливість швидко оцінити стан програми та оперативно передати цю інформацію розробникам.

На Agile -проект автоматизоване тестування, як тестування в цілому, є не ізольованим завданням або етапом у роботі над додатком, а безперервним процесом, який вписаний у всі стадії життєвого циклу ПЗ.

Для забезпечення якісного зворотного зв'язку автотести повинні виконуватися часто та швидко, а їх результати – бути достовірними та досить деталізованими [26].

Мінімальним критерієм готовності нової версії програми до випуску вважається відсутність регресійних дефектів. Тому одним із ключових моментів автоматизації тестування на Agile -проектах є частий запуск регресійних тестів [3].

Як правило, автотести для регресії включають кілька наборів тест-кейсів, які відрізняються за кількістю та ступенем деталізації тест-кейсів:

- пакет «димових» автотестів для перевірки того, що програма успішно завантажується та запускається, а також перевірки кількох ключових сценаріїв роботи з програмою. "Димовий" набір запускається при кожному розгортанні програми;

- пакет функціональних автотестів застосовується для детальнішої перевірки роботи програми, зазвичай включає кілька тестових наборів для різних цілей. Такі набори при необхідності запускаються в різних оточеннях і перевіряють стабільність роботи програми. Подібні автотести запускаються кілька разів на день;

- пакет регресійних автотестів призначений для тестування програми як єдиного цілого. Така перевірка дозволяє переконатися, що різні частини програми, які звертаються до інших програм, різних баз даних, сторонніх бібліотек та

зовнішніх ресурсів, працюють коректно.

Цей набір призначений не для перевірки всіх можливостей програми (їх робота раніше перевірена функціональними автотестами), а для тестування переходів з одного стану в інший, а також найбільш популярних сценаріїв користувача.

Для підготовки перерахованих автотестів використовують методику Scripting. Технічно вона дозволяє створювати автотести для графічного інтерфейсу. Але останні мають ряд особливостей, через які написання таких тестів з використанням Scripting стає неоптимальним рішенням.

Прийнято говорити про такі недоліки автотестів GUI:

- крихкість - для визначення графічних елементів та взаємодії з ними у тестах прописуються локатори, тому після зміни існуючих елементів або заміни їх новими тести перестають працювати;

- обмеженість тестування - графічний інтерфейс який завжди дозволяє тестувальнику повністю перевірити функціональність, оскільки він надає недостатньо деталей, необхідні верифікації;

- відносно низька швидкість виконання (проти юніт і API тестами) - оскільки тести проводяться через GUI, час завантаження графічних елементів помітно збільшує загальний час виконання тестів, у результаті зростає час отримання зворотний зв'язок;

- найменша окупність - через перераховані вище проблеми, автотести графічного інтерфейсу стають невигідними з фінансової точки зору.

В умовах Agile автотести GUI можуть втратити актуальність ще до першого запуску. Ручне створення автотестів нових графічних елементів займає багато часу, а виконання відкладається до наступної ітерації. На той момент графічний інтерфейс програми знову може змінитися і написані раніше автотести доведеться переробляти.

Тому на Agile проектах часто відбувається відмова від GUI тестів на користь тестування на рівні API [3, 14], де автотести можна запустити відразу після юніт-тестів.

Такий підхід до автоматизації допустимий для веб- та десктоп-додатків, які розраховані на роботу з обмеженою кількістю браузерів та платформ.

Але МД проектуються під різні мобільні платформи, версії їх операційних систем та конфігурації пристроїв. Ручне тестування МД дозволяє перевірити лише базові сценарії та лише у найбільш популярних оточеннях.

Тому автоматизації тестування, яка не охоплює перевірки графічного інтерфейсу, не застосовується більшість мобільних додатків. Для МД, які розробляються за технологією Agile, відсутність автотестів GUI особливо критична.

Типова тривалість ітерації на Agile проекті становить два тижні. За цей час неможливо протестувати графічний інтерфейс МД вручну. В результаті багато дефектів GUI потрапляють у продакшен-версію МД і виявляються кінцевими користувачами. Така ситуація може призвести до отримання негативних відгуків від користувачів та, як наслідок, до комерційного провалу програми.

Для вирішення цієї проблеми необхідно розробити нову методику автоматизації тестування, яка дозволить автоматизувати тестування як API, так і графічного інтерфейсу МД.

Загальний принцип цієї методики можна сформулювати так: використання Scripting для створення тестів API, Record and Play – для автотестів GUI.

Методику «запис і відтворення» має сенс застосовувати для тестування нових або екранів МД, що часто змінюються. Використання програм-драйверів дозволяє автоматизувати тестування нових елементів GUI відразу після їхньої розробки, в рамках однієї ітерації. Також ця методика дозволяє швидко підготувати нові автотести для покриття позапланових доробок у дизайні або його кардинальній зміні.

Ручне створення автотестів нових екранів займає більше часу, ніж генерація коду з допомогою драйвера, і передбачає виконання автотестів лише у наступній ітерації. До цього доведеться тестувати нові екрани вручну і, як наслідок, обмежитись перевіркою основних сценаріїв.

У результаті виникає подвійний ризик. По-перше, після введення нової версії МД в експлуатацію, користувачі можуть виявити значні дефекти в сценаріях, не

покритих ручними тестами. По-друге, після запуску автотестів у наступній ітерації може з'ясуватись, що їх необхідно доопрацювати. У цьому випадку фактичне застосування автотестів відкладається ще на два тижні.

Тому методика «написання сценарію» більше підходить для автоматизованої перевірки API та функціоналу, які не змінюються або змінюються незначно.

Під час підготовки автотестів GUI необхідно переконатися, що вони мінімально перетинаються з функціональними автотестами. Останні повинні покривати більшу частину позитивних та негативних сценаріїв, спрямованих на перевірку взаємодії із сервером, базою даних та різними зовнішніми сервісами.

Автотести графічного інтерфейсу не повинні перетинатися із функціональними перевірками. Виняток – тести, пов'язані з формуванням коректних запитів при взаємодії користувача з різними контролами.

Паралельна розробка та запуск автотестів для API та GUI допомагає максимально швидко оцінити його якість та виявити найбільші дефекти всіх типів – структурні помилки в коді, проблеми функціоналу та графічного інтерфейсу.

Друга перевага використання комбінованої методики - можливість комплексно аналізувати та лагодити проблеми, пов'язані одночасно з функціоналом та компонентами GUI. Це дозволяє уникнути ситуації, коли дефекти на відповідних рівнях виявляються і виправляються асинхронно, через що ремонт на стороні бек-енду може призвести до появи нових проблем графічного інтерфейсу і навпаки.

Висновки до розділу 2

1. Автоматизоване тестування має на увазі активну участь людини, яка за часом можна порівняти з організацією та проведенням ручного тестування. Тому ефективність автоматизації залежить від того, які саме завдання було вирішено автоматизувати та як вони були виконані.

2. При виборі методики автоматизації тестування для конкретного МД слід враховувати такі чинники: особливості предметної області та тип докладання, і навіть методологію, якою воно розробляється.

3. При автоматизації тестування на Agile проектах найчастіше використовується методика Scripting, яка не охоплює перевірки графічного

інтерфейсу. Такий підхід є неприйнятним для тестування МД, особливо тих, що розробляються за гнучкою методологією. Протягом короткої Agile -ітерації неможливо провести вручну повноцінне тестування GUI.

4. Необхідно розробити нову методику автоматизації тестування, яка дозволить автоматизувати тестування як API, так і графічного інтерфейсу МД. Ключовий принцип цієї методики: використання Scripting для створення тестів API, Record and Play – для автотестів GUI.

РОЗДІЛ 3
РОЗРОБКА І АПРОБАЦІЯ МЕТОДИКИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ МОБІЛЬНИХ ДОДАТКІВ НА AGILE -ПРОЕКТІ

Аналіз методики Scripting показав, що її застосування не є оптимальним рішенням для автоматизації тестування МД на Agile- проектах.

Для тестування МД необхідно використовувати автотести GUI. Це дозволить провести повне тестування графічного інтерфейсу МД протягом короткої Agile - ітерації. При цьому автотести GUI не є заміною, а доповненням для автотестів API. Тому автоматизація перевірок графічного інтерфейсу має скорочувати час створення автотестів API.

Також при постановці завдання на розробку методик автоматизації МД потрібно враховувати специфіку останніх:

- Мобільна розробка має на увазі необхідність часто змінювати або доопрацьовувати працюючий дизайн, переписувати код фронт-енду, щоб забезпечити коректну роботу програми з змінними параметрами мобільних пристроїв (розмір і роздільна здатність екрану, оновлення сенсорів і так далі);

- Більшість нативних і гібридних МД проектується для роботи на iOS і Android, тому код для клієнтської частини МД пишеться двома різними мовами. Відповідно, для автоматизації перевірок фронт-енду будуть потрібні окремі набори автотестів для кожної платформи.

Виходячи з вищесказаного, можна сформулювати такі вимоги до нової методики:

1. Можливість розробляти великий обсяг автоматизованих тестів GUI у стислий термін.

2. Можливість паралельно створювати та запускати автотести для API та графічного інтерфейсу мобільного додатка.

Застосування методики, розробленої з урахуванням даних вимог, дозволить підвищити ефективність процесу тестування МД.

3.1. Методика автоматизації тестування МД на Agile проекті.

На основі аналізу методики Scripting та з урахуванням специфіки тестування мобільних додатків розроблено методику автоматизації тестування МД, що проектуються за технологією Agile.

На рис. 3.1 представлена діаграма варіантів використання запропонованої методики.

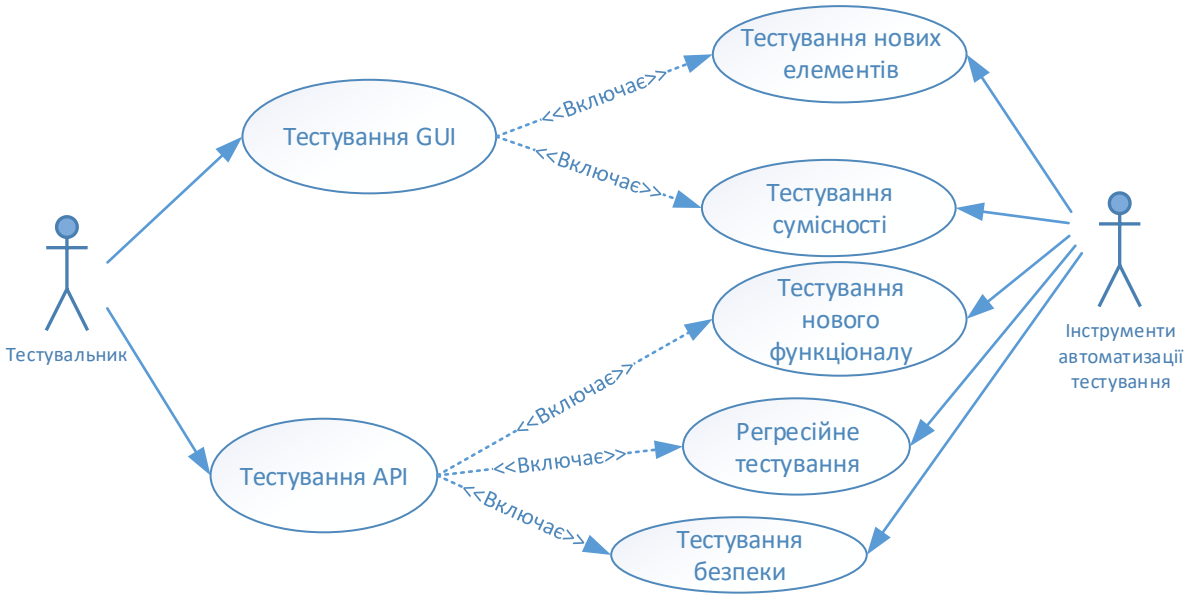

Рис. 3.1. Діаграма варіантів використання запропонованої методики

У запропонованій методиці для автоматизації тестування різних рівнях МД застосовуються дві різні методики:

- для створення автотестів GUI використовується методика Record and Play;
- Для створення автотестів API використовується методика Scripting.

Автоматизоване тестування паралельно виконується на GUI та API рівнях програми.

Тестування GUI стосовно МД включає такі перевірки:

- тестування працездатності нових елементів графічного інтерфейсу;
- перевірка функціонування елементів GUI на різних моделях мобільних пристроїв, версіях ОС та графічних оболонок, перерахованих у матриці тестових

пристроїв.

Така матриця складається на етапі проектування МД і включає пристрої з набором параметрів, які є найбільш популярними у потенційних користувачів. Приклад матриці тестових пристроїв представлений рис. 3.2.

Платф	Тип	Найменування	Модель	ОС	Ширин	Архітектура	Пк
Android	Smartphone	Google Pixel 3 XL	Pixel 3 XL	10	1440	2960 arm64-v8a	64
Android	Smartphone	Samsung Galaxy A50	SM-A505U	9	1080	2340 arm64-v8a	64
Android	Tablet	Samsung Galaxy Tab S4	SM-T830	8.1.0	1600	2560 arm64-v8a	64
Android	Tablet	Samsung Galaxy Tab S6 (Wifi)	SM-T860	9	1600	2560 arm64-v8a	128
IOS	Tablet	Apple iPad 2	A1395	9.3.5	768	1024 armv7f	32
IOS	Tablet	Apple iPad Mini 2	A1489	9.3.1	1536	2048 arm64	16
IOS	Smartphone	Apple iPhone 11	MWL72LL/A	13.1.3	828	1792 arm64e	64
Android	Tablet	ASUS Nexus 7 - 2nd Gen	ME571K	6	1200	1920 armeabi-v7a	32
Android	Smartphone	Galaxy S8 Unlocked	SM-G950U	7	1440	2960 arm64-v8a	64
Android	Smartphone	HTC One M9 (AT&T)	6735A	5.0.2	1080	1920 arm64-v8a	32
Android	Smartphone	LG Nexus 5	D820	5.0.1	1080	1920 armeabi-v7a	16
Android	Smartphone	LG Nexus 5	D820	6	1080	1920 armeabi-v7a	16
Android	Smartphone	Motorola Moto G - 3rd Gen	MotoG3	6	720	1280 armeabi-v7a	8
Android	Smartphone	Samsung Galaxy Note5 (AT&T)	SM-N9120	7	1440	2560 arm64-v8a	32
Android	Tablet	Samsung Galaxy Tab A 10.1"	SM-T580	7	1200	1920 armeabi-v7a	16
IOS	Tablet	Apple iPad Air	A1474	10.3.3	1536	2048 arm64	16
IOS	Tablet	Apple iPad Air 2	A1566	11.1	1536	2048 arm64	16
IOS	Smartphone	Apple iPhone 8	A1863	12	750	1334 arm64	64
IOS	Smartphone	Apple iPhone 8 Plus	A1864	11	1080	1920 arm64	64

Рис. 3.2. Матриця тестових пристроїв

Тестування API мобільного додатка включає два типи перевірок:

- функціональне тестування, яке включає тестування нової функціональності та регресійне тестування;
- тестування безпеки.

Для апробації розробленої методики було обрано такі інструментальні засоби:

1. Інтегровані середовища розробки (ICP) IntelleJ IDEA, Android Studio та Xcode. Перша використовувалася для створення автотестів для бекендної частини МД. Інші дві ICP застосовувалися для підготовки автотестів фронт-енду МД, представленого версіями під мобільні платформи iOS та Android;

2. Фреймворк Cucumber [24] для створення коду автотестів на основі сценаріїв, написаних предметно-орієнтованою мовою;

3. Фреймворк Allure для створення звітів про виконання автотестів.

Діаграма компонентів методики, що включає перелічені інструменти, представлена рис. 3.3

Після виконання всіх автотестів їх результати збираються до загального звіту, на основі якого проводиться аналіз ефективності процесу тестування.

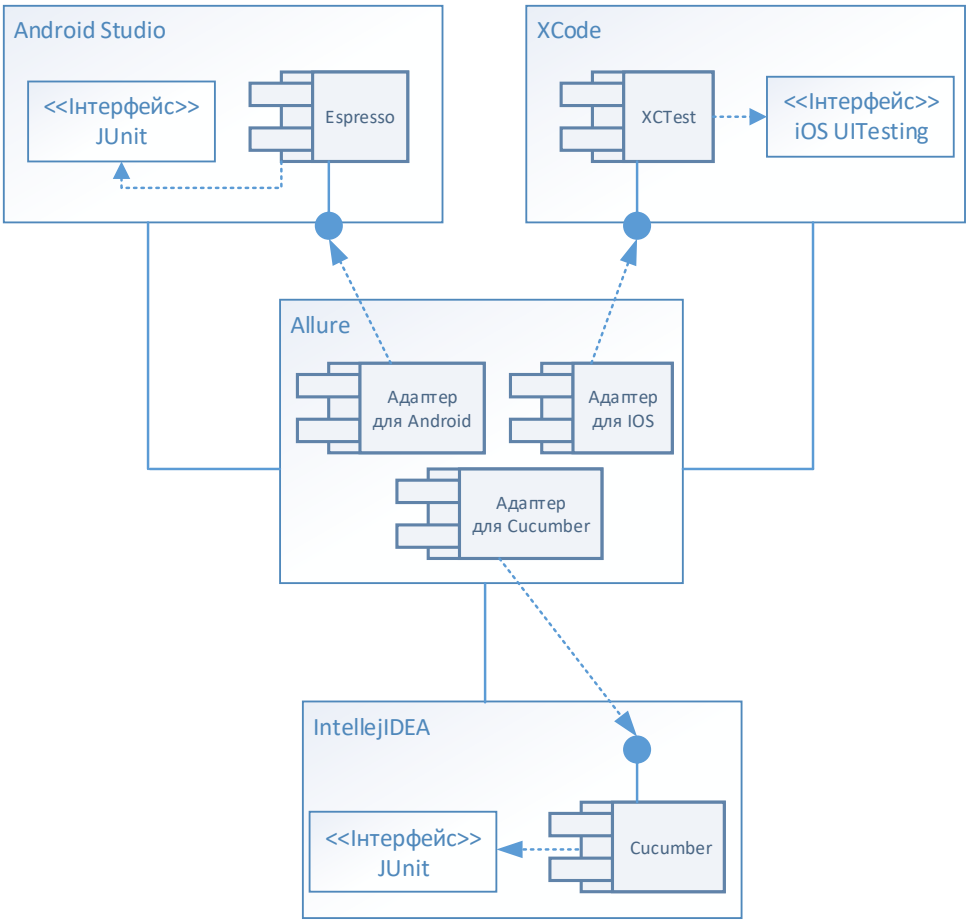

Рис. 3.3. Діаграма компонентів методики автоматизації МД

Для оцінки ефективності застосування методики використовуватимуться тестові метрики:

- Тестове покриття (Test Coverage) [33];

- кількість дефектів пріоритету (Bugs by Priority);

- кількість тест-кейсів, які не виконані в ході поточної ітерації (Not Run Test Cases).

Тестове покриття оцінюється як набір тестових пристроїв загалом, так і для кожного конкретного пристрою.

3.2. Апробація методики автоматизації тестування мобільних додатків

Апробацію розробленої методики автоматизації проведено на нативному МД для iOS та Android, яке розробляється за методологією Scrum із застосуванням BBD

-підходу.

У роботі використано таке визначення Scrum: «одна з гнучких технологій, що дозволяє жорстко фіксовані і невеликі за часом ітерації, звані спринтами (sprints), надавати кінцевому користувачеві працюючий продукт із новими бізнес-можливостями, котрим визначено найбільший пріоритет» [39].

МД призначене для адміністрування інформаційної системи (ІС) мережі книгарень. Це розрахована на багато користувачів додаток для отримання контрольованого доступу до ІС і виконання операцій, набір яких залежить від ролі користувача.

У додатку визначено такі ролі:

- власник мережі магазинів (Owner),
- головний адміністратор мережі магазинів (Main administrator),
- адміністратор філії (Department administrator),
- співробітник магазину (Employee).

Нижче наведено основні сутності, представлені в структурі даних МД:

- користувач (User),
- роль (Role),
- покупець (Customer),
- книга (Book),
- магазин (Department),
- видавництво (Publishing office),
- замовлення від покупця (Order),
- постачання від видавництва (Supply).

На рис. 3.4 показані варіанти використання МД для користувача з участю власника магазину.

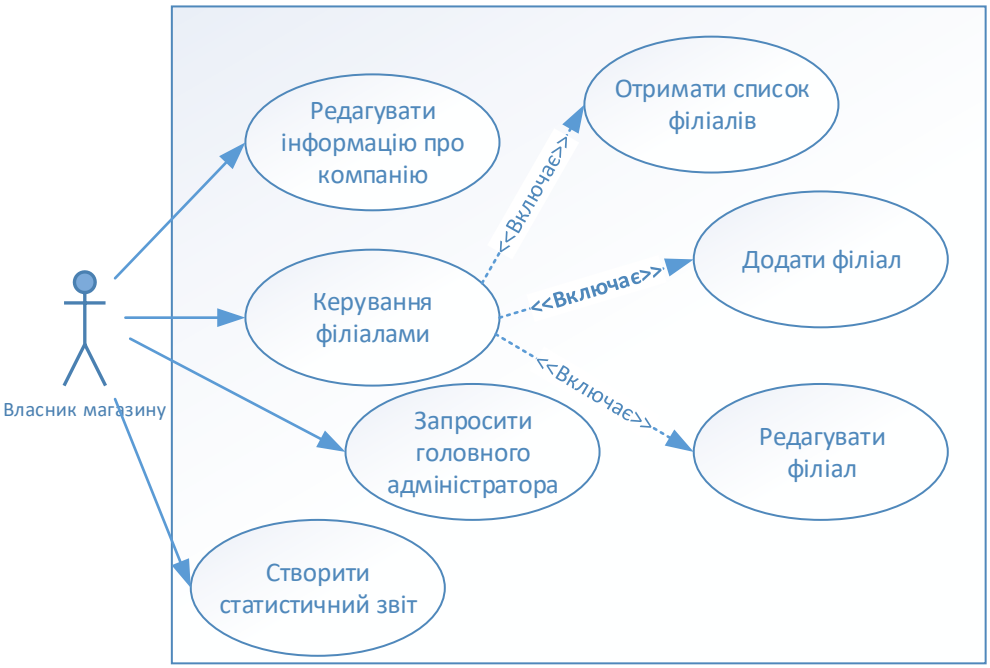

Рис. 3.4. Варіанти використання МД для власника магазину

На рис. 3.5 показані представлені варіанти використання користувача з участю головний адміністратор.

Рис. 3.5. Варіанти використання МД для головного адміністратора

На рис. 3.6 представлена діаграма варіантів використання користувача «адміністратор філії».

Рис. 3.6. Варіанти використання МД для адміністратора філії

На рис. 3.7 показано діаграму варіантів використання користувача «співробітник магазину».

Рис. 3.7. Варіанти використання МД для працівника магазину

Експеримент проводився після реалізації в МД нової функціональності, яка доступна всім користувачам МД - можливості перенесення книг із переліку постачання до бази книг магазину, який відображається на відповідному екрані. Для забезпечення цих функцій було додано новий екран «Доставлені книги» та декілька нових запитів:

- отримання переліку привезених книг,
- отримання конкретної книги зі переліку привезених книг,
- додавання книги зі переліку привезених книг до бази книг магазину.

Інші умови проведення експерименту описані у таблиці 3.1.

Таблиця 3.1

Умови проведення експерименту

Методика автоматизації	Record and Play	Scripting
Платформа	iOS, Android	iOS, Android
Метод тестування	Метод сірої скриньки	Метод сірої скриньки
Рівень МД	GUI	API
Тестована частина системи	Фронт-енд	Бек-енд
Вид тестування	Графічний інтерфейс, сумісність	Нової функціональності, регресійне, безпеки

Блок-схема алгоритму проведення автоматизованого тестування із застосуванням розробленої методики представлена рис. 3.8.

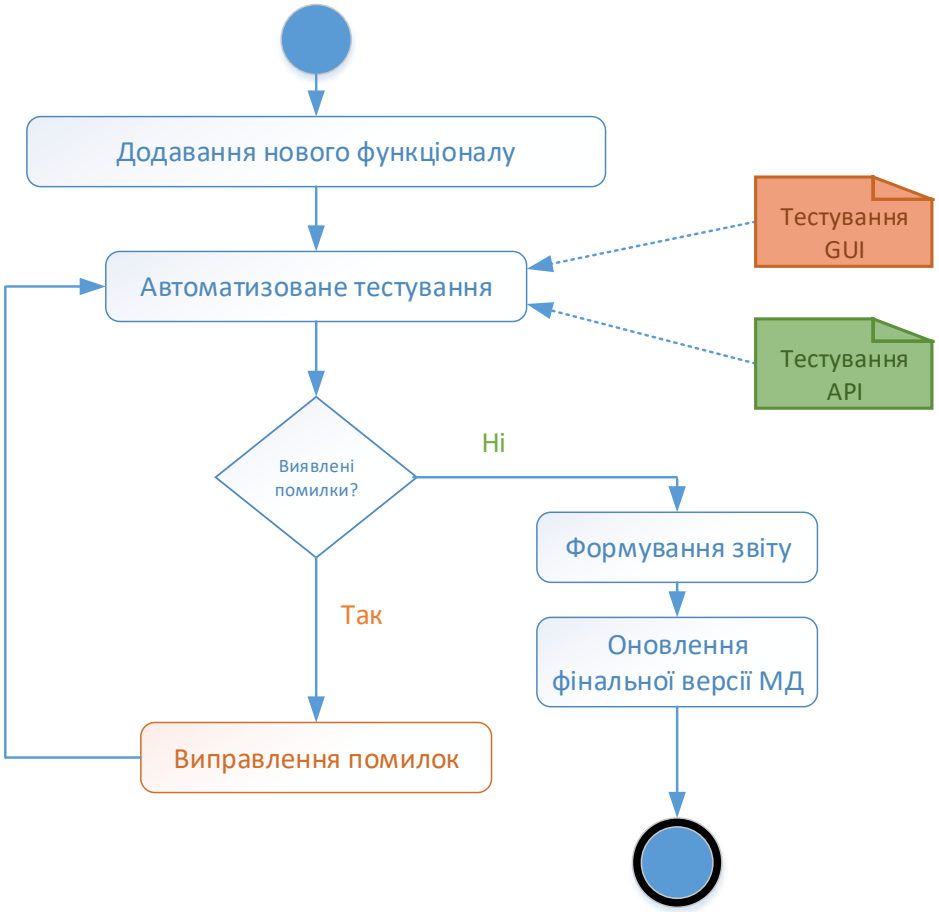

Рис. 3.8. Алгоритм проведення тестування із застосуванням розробленої методики

Далі описано апробацію розробленої методики для тестування GUI та API.

3.3. Апробація методик для автоматизації тестування GUI

Для запису автотестів із застосуванням методики Record and Play для МД використовують спеціальні драйвери, вбудовані в платформні засоби розробки. Для Android це драйвер Espresso у складі Android Studio, для iOS – драйвер XCTest, вбудований у Xcode.

Алгоритм створення таких автотестів однаковий для обох платформ:

Крок 1: у платформній ІСР вибрати реальний пристрій або емулятор для запуску програми, що тестується;

Крок 2: включити запис дій користувача, виконати вручну потрібний тест;

Крок 3: переконатися, що запису, створеної драйвером, немає зайвих кроків;

Крок 4: після завершення запису вибрати відповідну опцію, і згенерувати код автотесту;

Крок 5: запустити автоматичне виконання підготовленого тесту та переконатися, що він працює коректно;

Крок 6: за допомогою фреймворку Allure згенерувати звіт про результати виконання автотесту.

У таблиці 3.2 детальніше описано оточення, яке було створено для підготовки та запуску автотестів із застосуванням методики Record and Play для iOS та Android.

Таблиця 3.2

Опис оточення для роботи з автотестами Record and Play

Платформа	iOS	Android
Операційна система	macOS	Windows
додаток	.swift файли з вихідним кодом програми, що виконується ipa^nn	.java файли з вихідним кодом програми, що виконується apk^nn
Мова розробки програми	Swift	Java
Середовище розробки	Xcode 11.4.1	Android Studio 3.6.6
Інструмент	Надбудова Appium версія 1.15.1	Надбудова Appium 1.15.1
	Драйвер XCTest	Драйвери Espresso 3.2.0 та UIAutomator2

На рис. 3.9 приведені результати запуску автотестів в Android Studio.

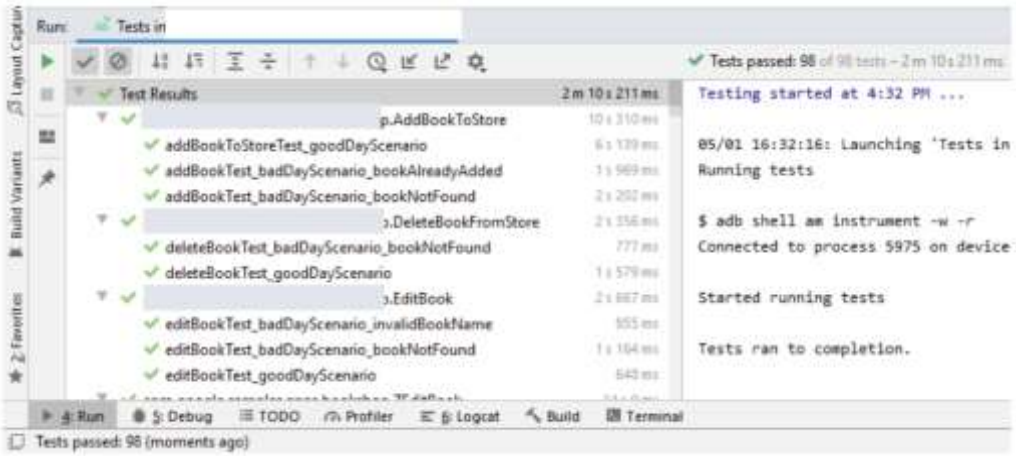

Рис. 3.9. Результати запуску автотестів Record and Play в Android Studio

На рис. 3.10 приведено фрагмент автотесту, записаного xCode.

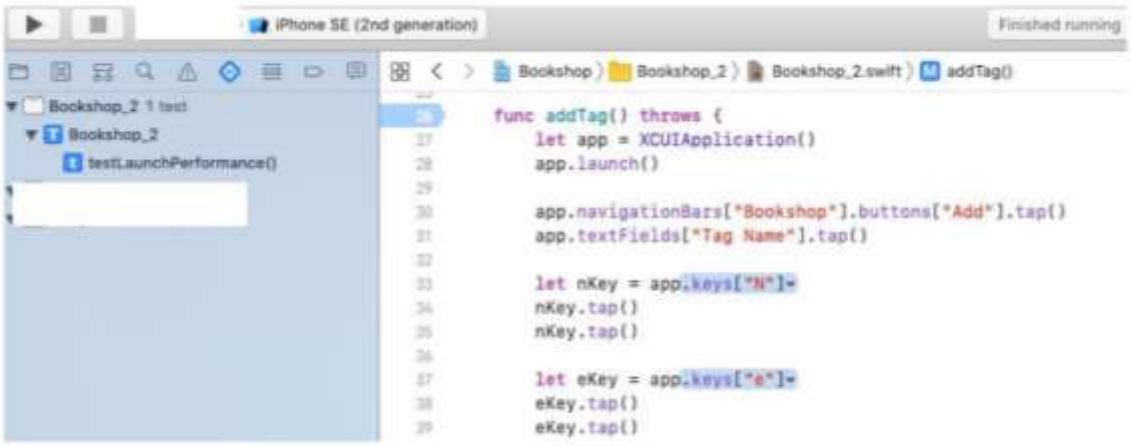

Рис. 3.10. Робота з автотестами Record and Play у xCode

В додатку Б наведено приклад автотесту, записаного за допомогою Espresso у Android Studio. Цей тест призначений для перевірки нових елементів GUI – екрана «Доставлені книги» та кнопки додавання книги до списку магазину.

В додатку В наведено приклад тесту, створеного xCode. Тест призначений для перевірки реалізованих раніше елементів GUI, пов'язаних з функціональністю додавання нового тега для книг.

При необхідності автотести, створені із застосуванням методики Record and Play, можна доопрацювати. Наприклад, додати до коду автотестів значення ідентифікаторів та імена локаторів елементів GUI, які не були розпізнані драйвером.

Для цього потрібно використовувати надбудову Appium та сумісні з нею драйвери. При роботі з МД використовувалися UIAutomator2 для Android і XCTest для iOS.

Алгоритм роботи з Appium виглядає так:

Крок 1: запустити сервер Appium;

Крок 2: в налаштуваннях Appium вказати можливості (Capabilities), які будуть використані при запуску сесії - платформу, версію ОС, ім'я реального пристрою або емулятора, назва потрібного драйвера та шлях до файлу, що виконується для тестованого МД;

Крок 3: розпочати сесію. Після підключення програми до пристрою та запуску МД перейти на потрібний екран та вибрати графічний елемент, для якого потрібно дізнатися локатор та (або) ідентифікатор.

Крок 4: скопіювати дані елемента раніше записані автотести, в яких задіяний цей елемент;

Крок 5: запустити оновлений тест та переконатися, що він працює коректно.

На рис. 3.11 показано конфігурацію Appium для запуску драйвера UIAutomator.

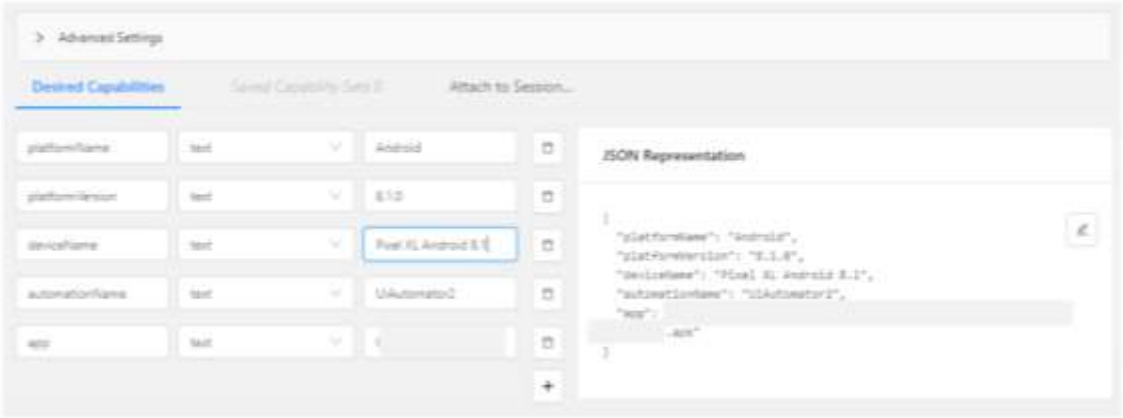

Рис. 3.11. Конфігурація Appium для запуску драйвера UIAutomator

На рис. 3.12 показано відображення локаторів графічних елементів в інтерфейсі UIAutomator.

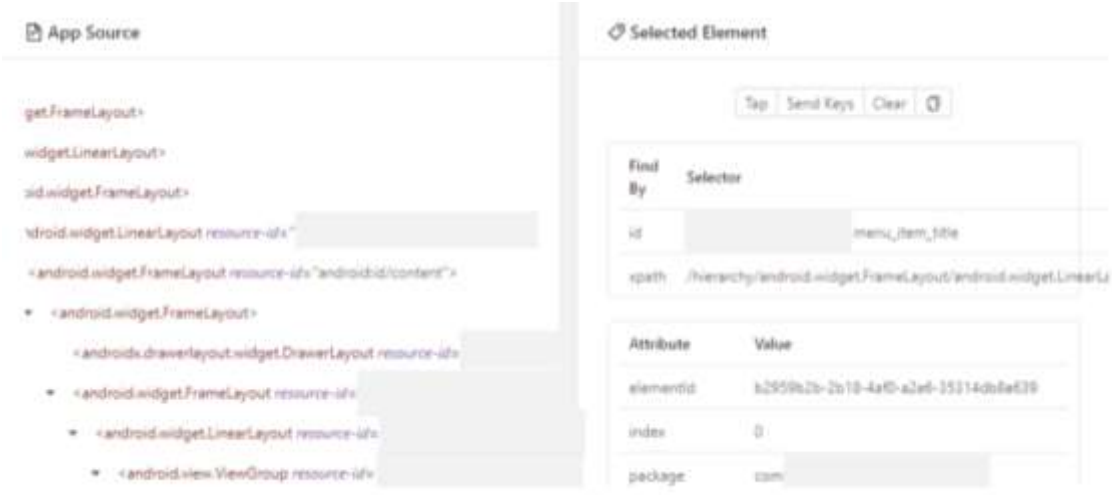

Рис. 3.12. Відображення локаторів елементів GUI в UIAutomator

Автотести, створені із застосуванням методики Record and Play, були виконані на всіх тестових пристроях із матриці пристроїв, підготовленої під час проектування МД. Це дозволило переконатися, що нові елементи графічного інтерфейсу коректно функціонують на всіх популярних оточеннях.

3.4. Апробація методик для автоматизації тестування API

Код бек-ендної частини МД не прив'язаний до конкретної мобільної платформи, тому автотести API є універсальними для iOS та Android. Для роботи з такими автотестами використовується середовище розробки, що підтримує мову, якою написано код програми.

Відповідно до принципів BDD, підготовка автотестів із застосуванням методики Scripting починається з написання тестового сценарію. Сутності та функціональності тестованого МД описуються за допомогою ключових слів предметно-орієнтованої мови, яка однаково зрозуміла розробникам, тестувальникам та бізнес-аналітикам [32].

Так як код бекенду МД, описаного в умовах проведення експерименту, реалізований на Java, при апробації методики використовувалося середовище Intellej IDEA. На рис. 3.13 показана підготовка тестового сценарію в Intellej IDEA

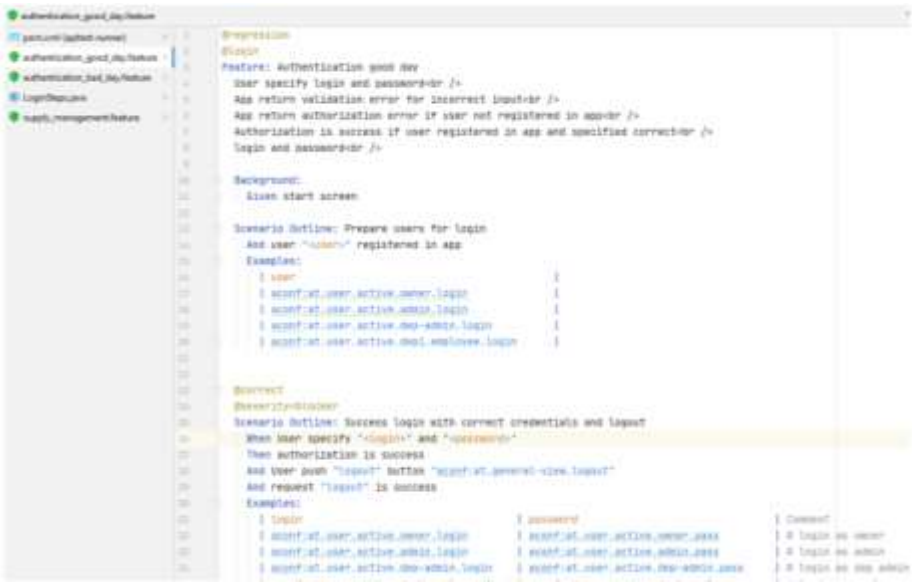

Рис. 3.13. Підготовка тестового сценарію IntelliJ IDEA

Алгоритм автоматизованого тестування із застосуванням методики Scripting складається з наступних кроків:

Крок 1: в ІСР створити сценарій тестування предметно орієнтованою мовою Gherkin, сумісному з фреймворком Cucumber [32];

Крок 2: написати код автотесту з урахуванням підготовленого сценарію;

Крок 3: за допомогою фреймворку складання Maven налаштувати конфігурацію запуску сценаріїв;

Крок 4: запустити отриманий автотест серед розробки;

Крок 5: за допомогою фреймворку Allure згенерувати звіт про виконання автотесту.

Опис оточення для підготовки та запуску автотестів Scripting наведено в таблиці 3.3.

Таблиця 3.3

Опис оточення для роботи з автотестами Scripting

Операційна система	Windows
Методологія	BDD
Середовище розробки	IntelleJ IDEA
Мова написання сценаріїв	Gherkin заснований
Мова написання коду автотестів	Java 8
Інструмент автоматизації	Фреймворк Cucumber 5.5.0, бібліотека cucumber Java 1.2.5
	Фреймворк автоматичного збирання Maven 3.6.1

В додатку Г наведено приклад тестового сценарію перевірки авторизації (Authentication) в МД з регресійного набору тестів. У сценарій включені як позитивні (correct), і негативні (fail) кейси.

Приклад більшого тестового сценарію для регресійного тестування наведено в додатку Д. В додатку Е представлений код тесту, написаний на основі сценарію Authentication на мові Java.

3.5. Результати застосування та оцінка ефективності розробленої методики

Результати виконання автоматизованого тестування МД, отримані під час апробації розробленої методики, представлені у таблиці 3.4.

Таблиця 3.4

Звіт про виконання автоматизованого тестування

Перевірка	Опис перевірки	Результат тестування
Тестування GUI	Перевірка наявності нових елементів графічного інтерфейсу та їх функціонування	Відповідає специфікації
Тестування GUI	Перевірка працездатності нових елементів GUI на різних пристроях	Виявлено незначні дефекти на деяких комбінаціях пристрій + платформа + версія ОС
Тестування API	Перевірка API - частини нової функціональності на відповідність специфікації	Відповідає специфікації
	Регресійна перевірка раніше створеного функціоналу, порушених при додаванні нової функціональності	Виявлено незначні дефекти
	Тестування безпеки передачі даних під час надсилання нових запитів	Виявлено незначні дефекти, пов'язані з порушенням рольової моделі

Застосування методики дозволило

- запустити автотести GUI на всіх тестових пристроях та завдяки цьому виявити дефекти, специфічні для конкретних пристроїв та версій ОС;

- виконати тестування API з використанням комбінаторних даних, що допомогло оперативно виявити проблеми під час проходження менш частотних сценаріїв, пов'язані з додаванням функціональності.

Для оцінки ефективності застосованої методики зроблено порівняння показників тестових метрик, отриманих до і після апробації розробленої методики. У результаті проведеного аналізу було виявлено такі зміни:

- тестове покриття збільшилося на 25% для всіх тестових пристроїв та на 60% для кожного конкретного пристрою;

- кількість не пройдених тест-кейсів зменшилася з 827 до 307 для всіх пристроїв та з 1938 до 704 для окремих пристроїв;

- кількість виявлених помилок збільшилася в 6 разів - з 12 до 72. Але зростання відбулося переважно за рахунок дефектів з незначним і тривіальним пріоритетом. Їхнє число склало 35 і 26 відповідно.

Така динаміка свідчить про те, що сценарії критичного шляху були вкриті до застосування автоматизації. Застосування розробленої методики дало змогу збільшити тестове покриття за рахунок запуску тестів на всіх тестових пристроях та виконання менш частотних сценаріїв, які при ручному тестуванні не перевірялися або перевірялися не повністю.

На рис. 3.14 представлена діаграма, що ілюструє стартові та підсумкові показники метрики тестове покриття для всього набору тестових пристроїв і для кожного окремого пристрою.

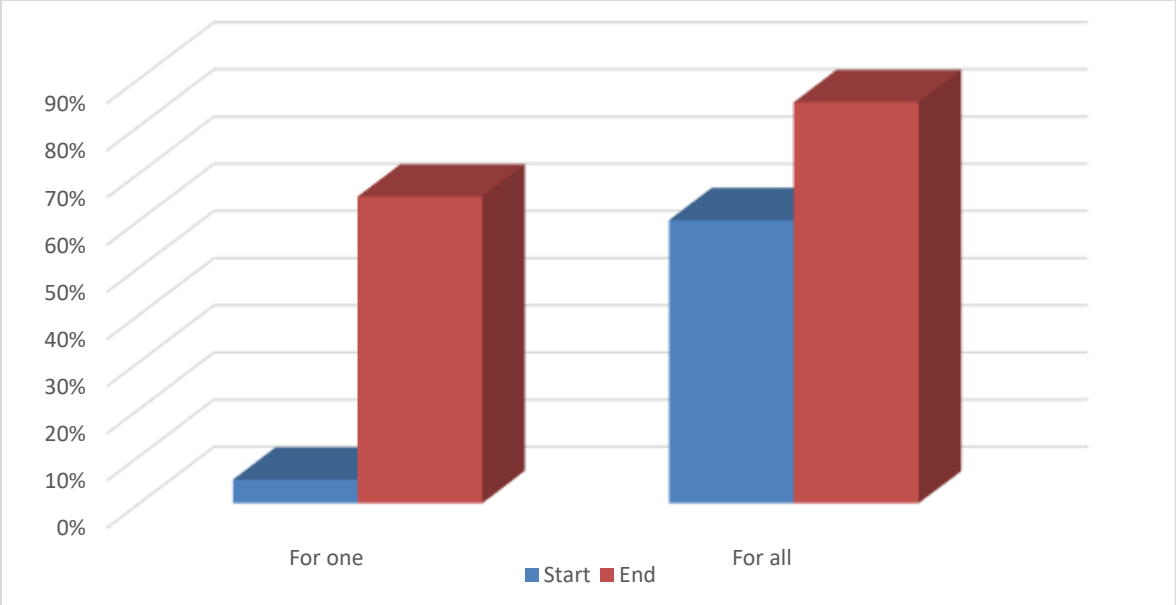

Рис. 3.14. Тестове покриття до та після застосування методики

На рис. 3.15 представлений фрагмент звіту про виконання всіх підготовлених наборів автотестів API після ремонту виявлених дефектів, згенерований за допомогою фреймворку Allure.

Рис. 3.15. Фрагмент звіту про виконання автотестів API

На рис. 3.16 показано фрагмент звіту про виконання конкретного автотесту.

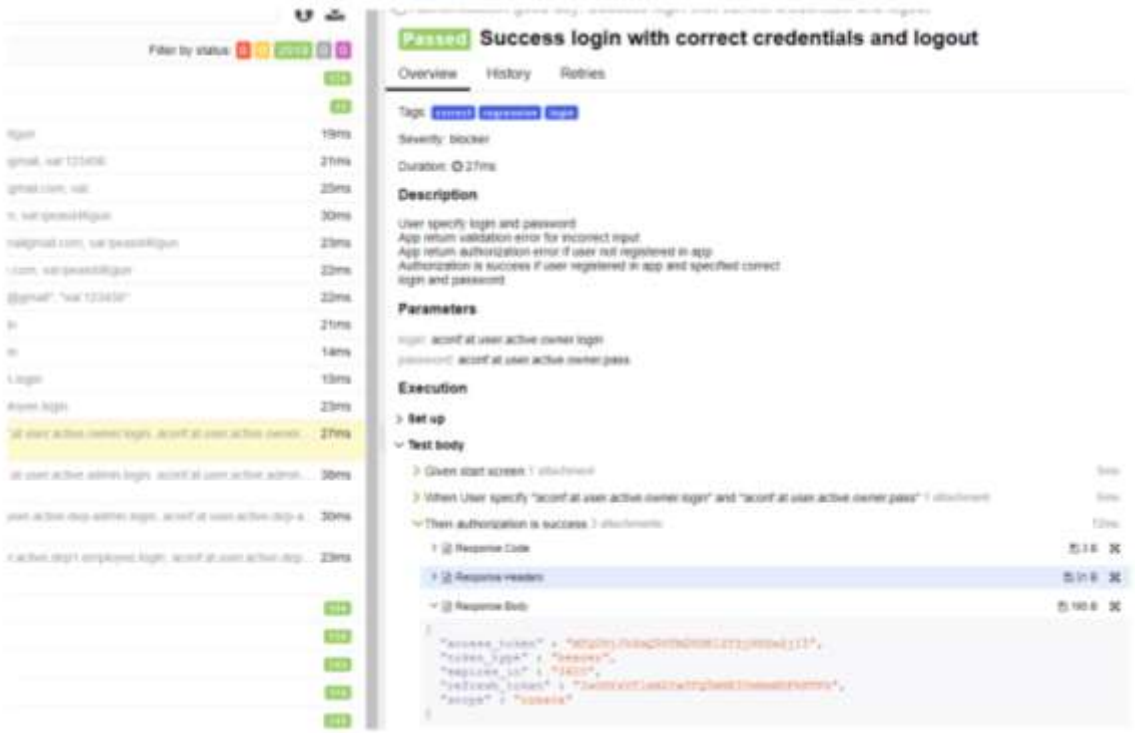

Рис. 3.16. Звіт про виконання автотесту «авторизація у МД»

Характер та якість змін доводять ефективність застосованої методики автоматизації тестування. Впровадження автотестів дозволило перевіряти більшість

сценарних розгалужень протягом двотижневої Scrum - ітерації. Таким чином, апробацію методики вважатимуться успішною.

Висновки до розділу 3

1. Для підвищення ефективності тестування МД на Agile проектах повинна використовуватися методика автоматизації, розроблена з урахуванням специфіки мобільних додатків.

2. Запропонована методика автоматизації тестування включає тестування GUI із застосуванням методики Record and Play і тестування API із застосуванням методики Scripting.

3. Для автоматизації процесу тестування МД використовують спеціалізовані засоби автоматизації: вбудовані в платформне середовище розробки драйвери для запису автотестів Record and Play, а також фреймворк, що реалізує підхід BDD.

4. Для проведення автоматизованого тестування API підготовлені тестові сценарії, написані предметно орієнтованою мовою Gherkin.

5. На основі запропонованої методики проведено тестування GUI та API мобільного додатка із застосуванням інструментів автоматизації, що підвищує ефективність процесу завдяки збільшенню тестового покриття.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Охорона праці

Усі дослідження проводились з дотриманням правил та норм охорони праці і вимог техніки безпеки. Одним із важливих питань є забезпечення вибухопожежобезпеки приміщень.

Всі приміщення поділяють на категорії. Категорія пожежної небезпеки приміщення (будівлі, споруди) це класифікаційна характеристика пожежної небезпеки об'єкта, що визначається кількістю і пожежонебезпечними властивостями речовин і матеріалів, які знаходяться (обертаються) в них з урахуванням особливостей технологічних процесів.

Відповідно до НАПБ Б.03.002-2007 приміщення за вибухопожежною та пожежною небезпекою поділяють на п'ять категорій (А, Б, В, Г, Д). Якісним критерієм вибухопожежної небезпеки приміщень (будівель) є наявність в них речовин з певними показниками вибухопожежної небезпеки. Кількісним критерієм визначання категорії є надмірний тиск (Р), який може розвинутись при вибуховому загорянні максимально можливого скупчення (навантаження) вибухонебезпечних речовин у приміщенні.

Під час аналізу пожежовибухонебезпеки необхідно також оцінювати можливість утворення вибухонебезпечного середовища необхідно враховувати небезпеку пожеж, що виникають внаслідок теплового прояву електричного струму. Однією з причин є загоряння кабелів і проводів:

1.Перегрів від короткого замикання між жилами кабелів, жилами кабелю та землею, який можливий внаслідок:

–пробою ізоляції підвищеною напругою, в тому числі від перевантаження, викликаного блискавкою;

–пробою ізоляції в місці механічного пошкодження в процесі експлуатації;

–пробою ізоляції при виникненні мікротріщин внаслідок заводського дефекту;

–пробою ізоляції від її старіння;

–пробою ізоляції в місці локального зовнішнього чи внутрішнього перегрівання;

–пробою ізоляції в місці локального підвищення вологості або агресивності середовища;

–випадкового або навмисного з’єднання струмопровідних жил кабелів та проводів між собою чи з’єднання струмопровідних жил із землею.

2.Перегрів від струмового перевантаження, який може статися у таких випадках:

– підключення споживача завищеної потужності;

– появи значного струму витоку між струмопровідними проводами, між струмопровідними проводами та землею;

– підвищення навколишньої температури на ділянці або в одному місці, погіршення тепловідводу чи вентиляції.

3.Перегрів у місцях перехідних опорів, який може виникнути при:

– послабленні контактного тиску в місці з’єднання двох або більше струмопровідних жил, що призводить до значного підвищення перехідного опору;

– окисненні в місцях з’єднання провідників електричного струму.

Значна кількість пожеж від теплового прояву електричного струму трапляється внаслідок використання саморобних електронагрівальних приладів, застосування жучків, недотримання безпечних відстаней, експлуатації несправного електроустаткування, неправильного вибору його виконання (ступеня захисту) залежно від класів зон.

Отже в даному підрозділі розглянуто особливості пожежної безпеки. Як висновок можна сказати, що робоче місце яке використовувалось для написання даного наукового дослідження відповідає вимогам з пожежної безпеки і забезпечено засобами пожежогасіння.

4.2. Вплив іонізуючого випромінювання на організм людини

Джерело іонізуючих випромінювань діє на організм при зовнішньому або внутрішньому опромінюванні (попаданні всередину організму з їжею, палінням і т. ін.). Під дією іонізуючих випромінювань в організмі людини відбувається іонізація молекул і атомів тканини, порушується хімічна структура сполук, утворюються сполуки, не властиві живій клітині, що в свою чергу призводить до її відмирання. Зміни фізичних і біологічних процесів в організмі залежно від дози опромінювання, тобто функції окремих органів і всього організму людини можуть відновлюватись повністю або вести до функціональних порушень організму і виникненню променевої хвороби.

Ураження може викликати гостру і хронічну форми променевої хвороби. Гостра форма хвороби виникає при дії великих доз опромінювання за короткий період часу, хронічна — розвивається в результаті тривалої дії малих доз при зовнішньому опромінюванні або при попаданні всередину організму під час приймання їжі, палінні, вдиханні невеликих кількостей радіоактивних речовин. При гострій променевій хворобі спостерігається анемія, слабкість і схильність організму до інфекційних захворювань.

На першій стадії хронічної променевої хвороби спостерігається порушення сну, погіршення апетиту, з’являється головний біль, слабкість і т. ін. На другій стадії ці симптоми загострюються ще більше, порушується обмін речовин, з’являються порушення в роботі серцево-судинної системи і органів травлення. На третій стадії порушується робота кровотворних органів, яка призводить до недокрів’я, лейкемії, відбувається крововилив в серцево-судинній системі, вражаються статеві органи, а також виникають зміни в генетичному апараті живого організму, якщо радіоактивне опромінювання діє на статеві органи і органи зародкового шляху. Спадкові зміни призводять до нежиттєздатності зародка як в першому, так і в наступних поколіннях. Шкідливі наслідки опромінення проявляються в стерильності потомства, в захворюваннях, які передаються в

спадщину від покоління до покоління і призводять до зменшення тривалості життя людини, зниження стійкості проти інфекційних захворювань.

Радіоактивні випромінювання викликають місцеві ураження: захворювання шкіри, злоякісні пухлини, катаракту, з'являється сухість шкіри, ламкість нігтів, випадає волосся. Небезпечність дії радіоактивних випромінювань обумовлюється ще й тим, що людина органами чуттів не відчуває їхньої дії доти, доки не з'явиться та або інша зміна в організмі.

Для попередження шкідливої дії іонізуючих випромінювань необхідно усувати всяку можливість опромінювання організму дозами, які перевищують гранично допустимі. Ступінь ураження радіоактивними речовинами організму людини залежить від ряду чинників: виду випромінювання (альфа-, бета-, гамма-промені і т. ін.); кількості ізотопу (активності); його властивостей (енергії частинок в період півроспаду та ін.); шляхів попадання в організм людини та його індивідуальної чутливості.

Впливу зовнішнього опромінювання організм зазнає тільки під час перебування людини у сфері впливу випромінювання. У випадку зникнення радіації припиняється і зовнішній вплив, а в організмі можуть розвинутись зміни – наслідки опромінювання.

Радіоактивні речовини можуть потрапляти до організму працюючих через легені або шлунково-кишковий тракт, а також через непошкоджену шкіру. Особливо небезпечні у цьому відношенні роботи, пов'язані з розробкою радіоактивних руд. Радіоактивне випромінювання не тільки спричинює іонізацію повітря, а й призводить до аналогічного процесу в тканинах організму, значно змінюючи їх. Потрапляючи до організму, радіоактивні речовини заносяться кров'ю у різні тканини та органи і стають джерелом внутрішнього опромінювання. Особливою загрозою для організму є ізотопи, які протягом усього життя потерпілого можуть бути джерелами іонізуючого випромінювання.

Впливу іонізуючого випромінювання можуть зазнати працюючі з рентгенівськими та γ-променями під час здійснення γ-дефектоскопії на промислових підприємствах, обслуговуючий персонал прискорювальних установок і ядерних

реакторів, а також зайняті розвідкою та добуванням корисних копалин та інше. У теперішній час вирішені основні питання радіаційної безпеки. Однак при порушеннях техніки безпеки або за певних обставин іонізуюче випромінювання може спричинити розвиток променевої хвороби (гострої та хронічної).

При початкових проявах захворювання показано тимчасове усунення від роботи, пов’язаної з впливом іонізуючої радіації, терміном до року. У разі наявності більш виражених проявів хвороби показано направлення хворого на лікарсько-експертну комісію для встановлення ступеня втрати професійної працездатності і трудових рекомендацій. Подальша трудова діяльність в контакті з цим фактором протипоказана.

Висновки.

Таким чином в даному підпункті розглянуто питання вплив іонізуючого випромінювання на організм людини. Необхідно дотримуватись організації праці і норм радіаційної безпеки. Всі види робіт повинні мати ефективну екранізацію. Велике значення має дозиметричний контроль, проведення попередніх і періодичних медичних оглядів, а також дотримування медичних протипоказань щодо осіб, які приймаються на роботу з радіоактивними речовинами.

ВИСНОВКИ

Під час проведення дослідження отримано такі основні висновки та результати:

1. Здійснено аналіз джерел з предмету дослідження, що підтвердило недостатність робіт, присвячених автоматизації тестування МД на Agile -проектах, що підтвердило актуальність теми дослідження

2. Зроблено аналіз методики Scripting, яка застосовується для автоматизації тестування на Agile -проектах, який показав, що її недостатньо для повноцінного тестування МД;

3. Розроблено та реалізовано методику автоматизації тестування МД на Agile -проектах, засновану на застосуванні двох різних методик для різних видів тестування: методики Record and Play для тестування графічного інтерфейсу та Scripting для тестування API;

Для застосування методики були використані спеціальні драйвери, вбудовані в платформні засоби розробки Android Studio і Xcode, а також фреймворк Cucumber, інтегрований в середу розробки IntelleJ IDEA.

4. Для оцінки ефективності розробленої методики зроблено порівняння показників тестових метрик, одержаних при тестуванні МД до та після застосування методики автоматизації.

5. Як показав аналіз отриманих результатів, впровадження методики дозволило збільшити тестове покриття МД на 25% для всіх тестових пристроїв та на 60% для кожного конкретного пристрою, а також збільшення кількості знайдених дефектів з 12 до 72 за рахунок більшої кількості дефектів із незначним пріоритетом.

Отримані зміни вказують підвищення ефективності процесу тестування МД при застосуванні розробленої методики. Таким чином, у роботі вирішено актуальну науково-практичну проблему розробки методики автоматизації тестування МД, що проектуються за технологією Agile.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Автоматизоване тестування. URL: https://uk.wikipedia.org/wiki Автоматизоване тестування (дата звернення: 1.12.2022).

2. Why Automation Testing is at the Centre of Agile. URL: https://www.browserstack.com/guide/automation-testing-in-agile. (дата звернення: 10.12.2022).

3. Automated Software Testing Isn't Automatic: Introduction to the Basics. URL: Режим доступу: URL https://www.smartsheet.com/automation-testing-software. (дата звернення: 10.12.2022).

4. The Ergonomics of a Mobile Application: Best Practices. URL: https://blog.ferpection.com/en/the-ergonomics-of-a-mobile-application-best-practices (дата звернення: 5.12.2022).

5. What is automated testing? URL: https://www.functionize.com/automated-testing (дата звернення: 5.12.2022).

6. A Comparative Analysis of Quality Assurance of Mobile Applications using Automated Testing Tools. URL: https://www.researchgate.net/publication/318340481_A_Comparative_Analysis_of_Quality_Assurance_of_Mobile_Applications_using_Automated_Testing_Tools (дата звернення: 5.12.2022).

7. Elfriede Dustin, Jeff Rashka , John Paul Automated Software Testing: Introduction, Management, and Performance: Introduction, Management, and Performance . Addison-Wesley Professional 1999, 608 p.

8. Kaner C. Testing Computer Software. Wiley; 2nd edition, 2019. 480 p.

9. Crispin, Lisa. Agile Testing A Practical Guide for Testers and Agile Teams. Addison-Wesley, 2008. 534 p.

10. Crispin L. Gregory J. Agile Testing: A Practical Guide for Testers and Agile Teams 1st Edition. AddisonWesley Professional, 2008. 576 p.

11. Чайковський А.В., Жаровський Р.О., Лещишин Ю.З Конспект лекцій з дисципліни «Дослідження і проєктування комп’ютерних систем та мереж» для студентів спеціальності 123 – Комп’ютерна інженерія. Тернопіль, 2021. 148 с.

12. Зарічний Н., Тиш Є. Автоматизація тестування мобільних додатків за технологією Agile. Матеріали X науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі системи та технології» (7-8 грудня 2022 року). Тернопіль: ТНТУ. 2022. С. 27.

13. Checklist for Testing iPhone and Android Software Products. URL: https://qatestlab.com/resources/knowledge-center/sample-deliverables/checklist-for-testing-iphone-and-android-software-products (дата звернення: 5.12.2022).

14. Complete Guide to Mobile App Testing and Tools in 2022. URL: http://thewebappmarket.com/complete-guide-to-mobile-app-testing-and-tools/ (дата звернення: 5.12.2022).

15. Agile Manifesto. URL: http://agilemanifesto.org/iso/ru/principles.html (дата звернення: 5.12.2022).

16. Bach, J. Lessons Learned in Software Testing / Cem Kaner, James Bach, Bret Pettichord. Wiley, 2001.

17. BDD. URL: https://wikipedia.org/wiki/BDD (дата звернення: 5.12.2022).

18. Bourque, P. SWEBOOK v 3.0 Guide to the Software Engineering Body of knowledge / Pierre Bourque, Richard E. (Dick) Fairley– IEEE Computer Society Products and Service, 2014.

19. Copeland, L. A Practitioner’s Guide to Software Test Design: Artech House Publishers, 2003.

20. Cucumber. URL: https://cucumber.netlify.app/docs/cucumber/ (дата звернення: 5.12.2022).

21. Davis, C. Agile Metrics in Action: Manning Publication, 2015. 272 p.

22. Gherkin Reference. URL: https://cucumber.netlify.app/docs/gherkin/reference/ (дата звернення: 5.12.2022).

23. Pashuk, Alesia Android app testing specifics. URL: https://www.scnsoft.com/blog/android-app-testing-specifics (дата звернення: 5.12.2022).

24. Pashuk, Alesia Mobile testing process: How to make it efficient. URL: https://www.scnsoft.com/blog/mobile-testingprocess-how-to-make-it-efficient - Назва з екрану.

25. Scrum. URL: https://wikipedia.org/wiki/SCRUM (дата звернення: 5.12.2022).

26. Smart, J.F. BDD in action Behavior-Driven Development for the whole software lifecycle: Manning Publication, 2015. 459 p.

Додаток А.

Тези конференцій

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

М А Т Е Р І А Л И

X НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

7–8 грудня 2022 року

ТЕРНОПІЛЬ
2022

УДК 004.416.2
Н. Зарічний, Є. Тиш
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ МОБІЛЬНИХ ДОДАТКІВ ЗА ТЕХНОЛОГІЄЮ AGILE

UDC 004.416.2
N. Zarichnyi, Ye. Tysh

AUTOMATION OF MOBILE APPLICATION TESTING USING AGILE TECHNOLOGY

В даний час технологія Agile набирає все більшої популярності у сфері розробки програмного забезпечення, у тому числі - для мобільних пристроїв.

Для автоматизації додатків, що проектуються із застосуванням гнучких методологій, найчастіше використовується методика Scripting. Він підходить для автоматизації тестування API, але є неоптимальним рішенням для підготовки автотестів GUI. В результаті, тестування графічного інтерфейсу проводиться вручну. Протягом коротких ітерацій на Agile проекті можливо провести тільки базові перевірки GUI. Такий підхід допустимий для веб- та десктоп-додатків, які розраховані на роботу з обмеженою кількістю браузерів та платформ. На відміну від них, мобільні програми розробляються під різні мобільні платформи, версії операційних систем і конфігурації пристроїв. Через те, що тестування обмежується базовими перевірками, багато дефектів GUI потрапляють у фінальну версію програми та виявляються кінцевими користувачами. Для мобільних програм така ситуація може призвести до отримання негативних відгуків від користувачів і, як наслідок, до комерційного провалу.

Agile – підхід до тестування має на увазі наступні зміни в роботі QA-команди:

- тестування перестає бути ізольованою фазою у створенні ПЗ та активно застосовується на всіх стадіях життєвого циклу продукту – починаючи з планування. Таким чином, QA -фахівці можуть своєчасно виявляти найпростіші у виправленні помилки (неточності, неоднозначні деталі та інші проблеми документації), скласти уявлення про програмний продукт задовго до написання коду та виявити його потенційно слабкі місця;
- обсяг тестової документації скорочується до мінімуму: на зміну докладним тест-кейсам приходять більш високорівневі та універсальні тест-плани та чек-листи. Формат чек-листа дозволяє не розписувати перевірки досконально, за рахунок цього документацію простіше підтримувати в актуальному стані, а у роботі тестувальника збільшується частка дослідницького тестування;
- на всіх стадіях розробки підтримується зворотний зв'язок між фахівцями з тестування та рештою членів команди (розробники, бізнес-аналітики, дизайнери, проектний менеджер). Завдяки цьому у кожного з учасників проекту формується повніше і всебічно бачення продукту;
- швидка віддача від тестування - знайдені баги підлягають оперативному виправленню, що дозволяє підтримувати «чистоту коду» та уникнути накопичення застарілого та складно підтримуваного коду;
- тестування – невід'ємна частина критерію готовності: ступінь готовності ПЗ визначається з урахуванням кількості, пріоритету та серйозності виявлених проблем. Наприклад, критерієм готовності МП до випуску може бути відсутність у продукті дефектів з пріоритетом вище «незначного» (Minor) або «середнього» (Normal).

Таким чином, актуальність дослідження обумовлена необхідністю розробки методики, яка дозволить автоматизувати тестування API та графічного інтерфейсу мобільного додатка.

Додаток Б.

Приклад автотесту, записаного за допомогою Espresso у Android Studio

```
@LargeTest
@RunWith(AndroidJUnit4.class) public class AddBookToStore {
public int POSITION = 3;
      @Rule
    public ActivityTestRule<BookshopActivity> mActivityTestRule = new
ActivityTestRule<>(BookshopActivity.class);
    @Test
    public void BookshopActivityTest3() {
        POSITION = new Random(new Date().getTime()).nextInt(12);
        ViewInteraction tabView = onView( aLLOf(withContentDescription("Book
        list"), chiLdAtPosition(
                            chiLdAtPosition( withId(R.id.tabs), 0), 1),
                    isDispLayed()));
        tabView.perform(click());
        ViewInteraction recyclerView = onView( aLLOf(withId(R.id.plant_list),
        chiLdAtPosition(
                            withCLassName(is("android.widget.FrameLayout")),
                            0)));
        recyclerView.perform(actionOnItemAtPosition(POSITION, click()));
        ViewInteraction floatingActionButton = onView( aLLOf(withId(R.id.fab),
        chiLdAtPosition(
                            chiLdAtPosition( withId(R.id.nav_host), 0),2),
                                isDispLayed()));
        floatingActionButton.perform(cLick());
        ViewInteraction appCompatImageButton = onView( aLLOf(chiLdAtPosition(
                    aLLOf(withId(R.id.toolbar), chiLdAtPosition(
                                withId(R.id.toolbar_layout), 1)),0),
                                isDispLayed()));
        appCompatImageButton.perform(cLick());
        ViewInteraction tabView2 = onView(
        aLLOf(withContentDescription("Store"), chiLdAtPosition(
```

```
                        chiLdAtPosition( withId(R.id.tabs), 0), 0),
                    isDispLayed()));
        tabView2.perform(cLick());
        ViewInteraction recyclerView2 = onView( aLLOf(withId(R.id.Book_list),
        chiLdAtPosition(
                        withCLassName(is("android.widget.FrameLayout")),
                        0)));
        recyclerView2.perform(actionOnItemAtPosition(0, cLick()));
}
    private staticMatcher<View> childAtPosition(
            final Matcher<View> parentMatcher, final int position) {
        return newTypeSafeMatcher<View>() {
            @Override
            public void describeTo(Description description) {
                description.appendText("Child at position "+ position + " in
                parent
");
                parentMatcher.describeTo(description); }
            @Override
            public booleanmatchesSafely(View view) {
                ViewParentparent = view.getParent();
                return parent instanceof ViewGroup &&
                parentMatcher.matches(parent)
                    && view.equals(((ViewGroup) parent).getChildAt(position));
            }
        };
    }
      }
```

Додаток В

Приклад тесту, створеного xCode

```
func testExample() throws {
    let app = XCUIApplication()
    app.launch()
    app.navigationBars["Bookshop"].buttons["Add"].tap()
    app.textFields["Tag Name"].tap()
    let nKey =
app/*@START_MENU_TOKEN
@*/.keys["N"]/*[[".keyboards.keys[\"N\"]",".keys[\"N\"]"],[[[-
1,1],[-1,0]]],[0]]@END_MENU_TOKEN@*/
    nKey.tap()
    nKey.tap()
    let eKey =
app/*@START_MENU
TOKEN@*/.keys["e"]/*[[".keyboards.keys[\"e\"]",".keys[\"e\"]"],[[[-
1,1],[-1,0]]],[0]]@END_MENU_TOKEN@*/
    eKey.tap()
    eKey.tap()
    let wKey =
app/*@START_MENU
TOKEN@*/.keys["w"]/*[[".keyboards.keys[\"w\"]",".keys[\"w\"]"],[[[-
1,1],[-1,0]]],[0]]@END_MENU_TOKEN@*/
    wKey.tap()
      wKey.tap()
    let spaceKey =
app/*@START_MENU
TOKEN@*/.keys["space"]/*[[".keyboards.keys[\"space\"]",".keys[\"sp
ace\"]"],[[[-1,1],[-1,0]]],[0]]@END_MENU_TOKEN@*/
    spaceKey.tap()
    spaceKey.tap()
    let tKey =
app/*@START_MENU
```

```
TOKEN@*/.keys["t"]/*[[".keyboards.keys[\"t\"]",".keys[\"t\"]"],[[[-
1,1],[-1,0]]],[0]]@END_MENU_TOKEN@*/
    tKey.tap()
    tKey.tap()
    let app2 = app
    app2/*@START_MENU
TOKEN@*/.keys["a"]/*[[".keyboards.keys[\"a\"]",".keys[\"a\"]" ],[[[-1,1],[-
1,0]]],[0]]@END_MENU_TOKEN@*/.tap()
    let gKey =
app2/*@START_MENU
TOKEN@*/.keys["g"]/*[[".keyboards.keys[\"g\"]",".keys[\"g\"]"],[[[- 1,1],[-
1,0]]],[0]]@END_MENU_TOKEN@*/
    gKey.tap()
    gKey.tap()
    app2/*@START_MENU
TOKEN@*/.buttons["Done"]/*[[".keyboards",".buttons[\"done\"
]",".buttons[\"Done\"]"],[[[-1,2],[-1,1],[-1,0,1]],[[-1,2],[-
1,1]]],[0]]@END_MENU_TOKEN@*/.tap()
    app/*@START_MENU
TOKEN@*/.otherElements["PopoverDismissRegion"]/*[[".other Elements[\"dismiss
popup\"]",".otherElements[\"PopoverDismissRegion\"]"],[[[-1,1],[-
1,0]]],[0]]@END_MENU_TOKEN@*/.tap()
    let bookshopButton = app.navigationBars["New tag"].buttons["Bookshop"]
    bookshopButton. tap()
    app.tables.children(matching: .cell).element(boundBy:
0).children(matching:
.staticText).matching(identifier: "-").element(boundBy: 0).tap()
    bookshopButton. tap()
  }
```

Додаток Г.

Приклад тестового сценарію перевірки авторизації (Authentication) в МД з регресійного набору тестів

```
@regression
@login
Feature: Authentication good day
  User specify login and password<br />
  App return validation error for incorrect input<br />
  App return authorization error if user not registered in app<br />
  Authorization is success if user registered in app and specified correct<br
  /> login and password<br />
  Background:
   Given start screen
  Scenario Outline: Prepare users for login
   And user "<user>" registered in app
   Examples:
    | user                  |
    | aconf:at.user.active.owner.login|
    | aconf:at.user.active.admin.login|
    | aconf:at.user.active.dep-admin.login|
    | aconf:at.user.active.dep1.employee.login |
  @correct
  @severity=blocker
      Scenario Outline: Success login with correct credentials and logout
When User specify "<login>" and "<password>"
  Then authorization is success
  And User push "logout" button "aconf:at.general-view.logout"
  And request "logout" is success
  Examples:
   | login              | password              | Comment
   | aconf:at.user.active.owner.login                        |
aconf:at.user.active.owner.pass                              | # login as
owner
```

```
   | aconf:at.user.active.admin.login                          |
aconf:at.user.active.admin.pass                               | # login as
admin
   | aconf:at.user.active.dep-admin.login | aconf:at.user.active.dep-
admin.pass | # login as dep admin
   | aconf:at.user.active.dep1.employee.login|
aconf:at.user.active.dep1.employee.pass | # login as employee
 @fail
 @severity=blocker
 Scenario Outline: Failed login in case of incorrect input
  When User specify "<login>" and "<password>"
  Then app return validation error
  Examples:
   | login         | password | Comment
   | val:          | val:qwasd46gun | # empty login, correct password
   | val:test_user@gmail          | val:123456| # correct login, incorrect
password
   | val:test_user@gmail.com | val:| # correct login, empty password
   | val:@gmail.com    | val:qwasd46gun | # login with no email name, correct
password
   | val:test_usergmailgmail.com | val:qwasd46gun | # login with no @,
correct password
   | val:test_user@.com| val:qwasd46gun | # login with incomplete domain
name, correct
password
 @fail
 @severity=blocker
 Scenario Outline: Log In by not registered user
  When User specify <login> and <password>
  Then app return authorization error
  Examples:
   | login          | password   |
  | "val:unregistered_user@gmail" | "val:123456"|
```

Додаток Д.

Тестовий сценарій для перевірки функціональності

Сценарій для перевірки функціональності «Додавання постачання» (Add supply):

```
@regression
Feature: Supply management good day
 As a department administrator I want to create supply from publishing office

 Scenario : Prepare user for supply
  And user "aconfat.user.active.dep-admin.login" registered in app

 Scenario Outline :
  Given book with "<name>" created in catalog
  Examples :
   | name |
   | aconf:at.supply-view.create-form.values.book1 |
   | aconf:at.supply-view.create-form.values.book2 |
   | aconf:at.supply-view.create-form.values.book3 |

 @severity=blocker
 Scenario : Successfully login with correct credentials
  Given success previous scenario
  When User specify "aconf:at.user.active.dep-admin.login" and
"aconf:at.user.active.dep- admin.pass"
  Then authorization is success

 @severity=blocker
 Scenario : Successfully push button "Create supply"
  Given success previous scenario
  And User watch "supplying" view "aconf:at.supply-view.view"
  When User push "Create supply" button "aconf:at.supply-view.create"
  The User User can see "Create supply" form "aconf:at.supply-view.create-
form "
  Then request "create-form" is success

 @severity=blocker
 Scenario : Successfully creation supply in draft status
  Given success previous scenario
  When User fills "publishing office" field as "aconfat.supply-view.create-
form.values.p- office"
  And User add book "aconf:at.supply-view.create-form.values.book1"
  And User add book "aconf:at.supply-view.create-form.values.book2"
```

```
  And User add book "aconf:at.supply-view.create-form.values.book3"
  And User fills "Description" field as "val:Some test description"
  And User push "Save draft" button "aconf:at.supply-view.create-form.save"
  Then request "save" is success

 @severity=blocker
 Scenario : Successfully view supply in draft status
  Given success previous scenario
  When User open "supply" object "store:createdSupply"
  The field "publishing office" is equal to "aconf:at.supply-view.create-
form.values.p-office"
  And supply contains book "aconf:at.supply-view.create-form.values.book1"
  And supply contains book "aconf:at.supply-view.create-form.values.book2"
  And supply contains book "aconf:at.supply-view.create-form.values.book3"
  And field "Description" is equal to "val:Some test description"
  And supply has status "DRAFT"

 @severity=blocker
 Scenario : Successfully editing supply in draft status
  Given success previous scenario
  And User open "supply" object "store:createdSupply"
  When User remove book "aconf:at.supply-view.edit-form.values.book2" from
supply
  And User fills "Description" field as "val:Some another test description"
  And User push "Save draft" button "aconfat.supply-view.create-form.save"
  Then request "save" is success

 @severity=blocker
 Scenario : Successfully view supply in draft status
  Given success previous scenario
  When User open "supply" object "store:createdSupply"
  У цьому полі "publishing office" is equal to "aconfat.supply- view.edit -
form.values.p-office"
  And supply doesn't contain book "aconf:at.supply-view.create-
form.values.book2"
  And field "Description" is equal to "val:Some another test description"
  And supply has status "DRAFT"

 @severity=blocker
 Scenario : Successfully send supply email to publishing office
  Given success " Successfully creation supply in draft status " scenario
  When User open "supply" object "store:createdSupply"
  And User push "Запустити офіс" button "aconfat.supply-view.edit-
form.publish"
  Then supply has status "SENT"
```

```
 And publishing office receive email from "aconfat.notifier.email"
 And email contains "book" "aconf:at.supply-view.create-form.values.book1"
 And email contains "book" "aconf:at.supply-view.create-form.values.book3"
 And email contains "description" "val:Some another test description"

@severity=blocker
Scenario : Successfully logout
 Given success " Successfully login with correct credentials " scenario
 And User push "logout" button "aconfat.general-view.logout"
 Then request "logout" is success
```

Додаток Е.

Код тесту, написаний на основі сценарію Authentication на мові Java

```
@Slf4j
public class LoginSteps extends ASteps {
 @Given("start screen")
 public void startScreen() {
   getUrl(StorageUtils. getStr ("aconf:ui.url"));
 }
 @Then("app return validation error")
 public void appReturnValidationError() {
  try {
   createResponseStorageAttachment(
     auth2(user.get("login"), user.get("password"))
   );
  } catch (IOException e) {
   createExceptionStackTraceAttachment (e);
  }
 }
 @Then("authorization is success")
 public void authorizationIsSuccess() {
  try {
   createResponseStorageAttachment(
     authCheck(user.get("login"), user.get("password")) );
  } catch (IOException e) {
   createExceptionStackTraceAttachment(e);
  }
 }
 @Then("app return authorization error")
 public void appReturnAuthorizationError() {
  try {
   createResponseStorageAttachment(
     auth(user.get("login"), user.get("password"))
   );
```

```
  } catch (IOException e) {
   createExceptionStackTraceAttachment(e);
  }
  }
  @When("User specify {string} and {string}")
  public void userSpecifyCorrectLoginAndPassword(String login, String
password) {
   user = ImmutableMap. of(
     "login", StorageUtils. getStr (login),
     "password", StorageUtils. getStr (password)
   );
   createMapAsJsonAttachmentWithName(user, "Current User"); }
  @And("user {string} registered in app") public void
  userRegisteredInApp(String user) { Map userModel = ImmutableMap. of(
     "login", StorageUtils. getStr (user)
   );
   createMapAsJsonAttachmentWithName(userModel, "Verified User"); } }
```