БАКАЛАВРСЬКА РОБОТА

ДРБ. ІП - 20.00.00.000 ПЗ

Група ІП-21-1

Лях Ілля

2025

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Лях Ілля Петрович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

БАКАЛАВРСЬКА РОБОТА

Порівняльний аналіз методологій розробки мобільних додатків (назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121– Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____ Лях Ілля Петрович _____
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____Пасєка Надія Мирославівна, к.т.н., доцент_____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

_доц._____Бандура В.В.__
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доцент. В.В. Бандура
“ ____ ” _____2024 р.

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ БАКАЛАВРА СТУДЕНТОВІ

Лях Ілля Петрович

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) "Порівняльний аналіз методологій розробки мобільних додатків”

керівник проекту (роботи) к.т.н., доцент Пасєка Надія Мирославівна

затверджені наказом вищого навчального закладу від “ 28 ” квітня 2025 р. № 264/7

2. Строк подання студентом проекту (роботи) 10 червня 2025 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1 Реальні виклики в розробці мобільних додатків

2 Структура оцінювання для cpdt

3 Підходи до мобільної розробки та особливості android

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Як тестуються ваші нативні мобільні додатки (рис .1.1,ст. 19)

2. Хто відповідає за тестування нативних мобільних додатків (рис .1.2,ст. 21)

3. Фази та стадії Mobile-D (рис .2.1,ст. 29)

4. Запропоновані категорії мобільних додатків на основі (рис .1.1,ст. 36)

5. Етап визначення обсягу (рис .3.2,ст. 37)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання 10 травня 2025 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обгрунтування теми роботи	15.02.2025	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	25.02.2025	виконано
3	Побудова моделі або алгоритму власного рішення	15.03.2025	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	25.04.2025	виконано
5	Оформлення пояснювальної записки кваліфікаційної роботи	10.06.2025	виконано

Студент _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 54 сторінки, 18 рисунків, 1 таблиця, список використаних джерел із 20 найменування

Метою роботи є провести порівняльний аналіз сучасних методологій розробки мобільних додатків, виявити їх ефективність у вирішенні практичних викликів мобільної розробки,

Об'єкт дослідження: процес розробки мобільних додатків у контексті сучасних методологій програмної інженерії.

Предмет дослідження: Методології розробки мобільних додатків, зокрема їх особливості, переваги, недоліки та можливості застосування в умовах різних мобільних платформ, таких як Android та iOS.

Результати дослідження: розробка з урахуванням типу мобільного додатку дозволяє краще налагоджувати процес розробки, підвищувати точність планування та якість продукту.

В першому розділі розглянуто теоретичні основи розробки мобільних додатків, включаючи принципи роботи нативних і крос-платформних підходів, архітектурні патерни

В другому розділі порівняльному аналізу методологій розробки мобільних додатків, що використовуються для створення ефективних, користувацьки орієнтованих і масштабованих рішень для платформ iOS та Android.

В третьому розділі проведено аналіз документації, екосистеми інструментів і підтримки спільноти для кожної методології.

Висновок: Методології розробки мобільних додатків мають адаптуватися до специфіки мобільного середовища: частих змін, обмежених ресурсів та високої конкуренції.

КЛЮЧОВІ СЛОВА: РОЗРОБКА МОБІЛЬНИХ ДОДАТКІВ, НАТИВНА РОЗРОБКА, КРОС-ПЛАТФОРМНА РОЗРОБКА, ГІБРИДНА РОЗРОБКА, SWIFTUI, KOTLIN, FLUTTER, REACT NATIVE, IONIC, ПРОДУКТИВНІСТЬ,

ANNOTATION

Bachelor's thesis contains 54 pages, 18 figures, 1 table, list of used sources with 20 titles

The purpose of the work is to conduct a comparative analysis of modern mobile application development methodologies, to identify their effectiveness in solving practical challenges of mobile development,

Object of research: the process of developing mobile applications in the context of modern software engineering methodologies.

Subject of research: Mobile application development methodologies, in particular their features, advantages, disadvantages and application possibilities in the conditions of different mobile platforms, such as Android and iOS.

Research results: development taking into account the type of mobile application allows you to better adjust the development process, increase the accuracy of planning and product quality.

The first section considers the theoretical foundations of mobile application development, including the principles of native and cross-platform approaches, architectural patterns

The second section provides a comparative analysis of mobile application development methodologies used to create effective, user-oriented and scalable solutions for the iOS and Android platforms.

The third section analyzes the documentation, tool ecosystem, and community support for each methodology.

Conclusion: Mobile application development methodologies must adapt to the specifics of the mobile environment: frequent changes, limited resources, and high competition.

KEYWORDS: MOBILE APPLICATION DEVELOPMENT, NATIVE DEVELOPMENT, CROSS-PLATFORM DEVELOPMENT, HYBRID DEVELOPMENT, SWIFTUI, KOTLIN, FLUTTER, REACT NATIVE, IONIC, PRODUCTIVITY

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ……7

ВСТУП 8

РОЗДІЛ 1. РЕАЛЬНІ ВИКЛИКИ В РОЗРОБЦІ МОБІЛЬНИХ ДОДАТКІВ 10

1.1. Дослідження дизайну 11

1.2. Загальні виклики для мобільних розробників 14

1.3. Розробка для кількох платформ 17

1.4 Висновки до розділу……23

РОЗДІЛ 2. СТРУКТУРА ОЦІНЮВАННЯ ДЛЯ CPDT 24

2.1. Розробка мобільних додатків 24

2.2. Огляд Mobile-D 27

2.3. Підходи до вдосконалення 29

2.4 Висновки до розділу……...31

РОЗДІЛ 3. ПІДХОДИ ДО МОБІЛЬНОЇ РОЗРОБКИ ТА ОСОБЛИВОСТІ ANDROID 32

3.1. Категорії мобільних додатків 32

3.2. Альтернативні підходи до мобільного розвитку 37

3.3. Керування ресурсами Android 41

3.4 Висновки до розділу……...52

ВИСНОВКИ 56

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ……...58

БІБЛІОГРАФІЧНА ДОВІДКА

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

HCI - взаємодії людини з комп’ютером

ASD - адаптивної розробки програмного забезпечення

SDK - наборі програмного забезпечення.

TDD - практика розробки, керованої тестуванням

MDE - інженерії, керованої моделлю (

MDD - розробки, керованої моделлю (

AOSD - методи аспектно-орієнтованої розробки програмного забезпечення (

XP - екстремальному програмуванні

MDA - модельно-керованій архітектурі

PIM - платформено-незалежна модель

OMG - стандартизованою групою управління об’єктами

PSM- низка платформо-специфічних моделей

ВСТУП

Актуальність теми дослідження. Останніми роками світ мобільних операційних систем стає все більш фрагментованим. Кожна компанія або консорціум розробив власний нестандартний метод розробки додатків для своїх пристроїв, використовуючи різноманітні мови програмування та комплекти розробки програмного забезпечення (SDK). Це означає, що програма, створена для операційної системи Google Android, не працюватиме на конкуруючій платформі Apple iOS. Існує подальша фрагментація всередині платформ, оскільки деякі програми не можуть працювати на різних версіях або пристроях однієї платформи, що значно ускладнює розробку.

Фрагментація ринку зумовлена багатьма факторами. Окрім різноманітності програмної платформи, варіації апаратного забезпечення створюють труднощі з перенесенням користувацького досвіду (UX), включаючи метод взаємодії та користувацький інтерфейс (UI) з одного пристрою на інший [3]. Оскільки у 2011 році було продано понад 20 мільйонів планшетних пристроїв, було важко дозволити програмам максимізувати більші екрани цих пристроїв [24]. Багато розробників змушені вибирати підтримку лише деяких платформ і версій через обмежені фінансові ресурси або знання методів кодування для кожної платформи. Без широкої підтримки розробників платформи вважаються слабкими та мають труднощі з маркетингом продуктів. Це стосується нових операційних систем і нових випусків, які порушують сумісність із застарілим програмним забезпеченням.

Прикладом цього є перехід Research in Motion на платформу BB10, що порушує сумісність із власно розробленими програмами BlackBerry OS [27]. Зважаючи на кількість використовуваних мов, розробники повинні бути дуже універсальними, а підприємствам потрібно витрачати значні ресурси, щоб їх програмне забезпечення було доступне на кількох платформах. Навіть з урахуванням додаткових витрат, опитування Appcelerator і IDC за серпень 2012 року показує, що компанії продовжують дуже цікавитися кількома платформами 1 незважаючи на труднощі [7]. У 2011 році розробники виявили інтерес до

роботи на удвічі більшій кількості платформ, ніж у аналогічному опитуванні попереднього року, і зараз у середньому отримують неймовірні чотири операційні системи [4] [6]. Ця тенденція продовжувала посилюватися в 2012 році [7]. З огляду на таку велику зацікавленість у доступності програм для багатьох платформ, існує очевидна проблема, з якою стикаються розробники, вимагаючи можливості зробити свої програми доступними для якнайширшої аудиторії, але не вистачає ресурсів для створення нативно для кожної платформи.

Щоб розширити охоплення додатків, не витрачаючи значний час, необхідний для вивчення особливостей кожної платформи, використання кросплатформних засобів розробки (CPDT) може стати відповіддю. Генеральний директор InRuntime сказав, що використання CPDT скоротило час виходу на ринок на 70% [68]. Це показує, що використання кросплатформних інструментів може бути дуже корисним у багатьох відношеннях. Однак не існує адекватних заходів, які б гарантували, що ці CPDT такі ж потужні, як власні інструменти розробки. Через це розробники не впевнені, чи зможуть вони використовувати ці інструменти для створення потужних додатків із нативною функціональністю та продуктивністю.

Метою роботи провести порівняльний аналіз сучасних методологій розробки мобільних додатків, виявити їх ефективність у вирішенні практичних викликів мобільної розробки, а також надати рекомендації щодо вибору відповідного підходу залежно від категорії додатку, платформи та ресурсних обмежень.

Для досягнення поставленої мети в роботі необхідно вирішити наступні завдання: дослідити сучасні виклики, з якими стикаються розробники мобільних додатків, зокрема при багатоплатформенній розробці, проаналізувати основні засади, структуру та практики методології Mobile-D, визначити переваги та недоліки гнучких підходів до мобільної розробки (Agile, XP, Crystal тощо), розробити критерії для порівняння методологій розробки з урахуванням потреб мобільних додатків, провести порівняльний аналіз методологій з позиції ефективності, адаптивності та відповідності викликам мобільного середовища,

запропонувати удосконалення до існуючих методів (зокрема Mobile-D) з урахуванням категорій мобільних додатків та специфіки Android, узагальнити практичні рекомендації щодо вибору методології для різних типів мобільних застосунків

Об’єктом дослідження є процес розробки мобільних додатків у контексті сучасних методологій програмної інженерії.

Предметом дослідження є методології розробки мобільних додатків, зокрема їх особливості, переваги, недоліки та можливості застосування в умовах різних мобільних платформ, таких як Android та iOS.

В процесі дослідження використовували **такі методи -** аналіз наукових джерел, Контент-аналіз, порівняльний, метод обґрунтованої теорії, систематизація, моделювання

Бакалаврська робота містить 54 сторінки, 18 рисунків, 1 таблиця, 3 розділи, список використаних джерел із 20 найменуванням.

РОЗДІЛ 1. РЕАЛЬНІ ВИКЛИКИ В РОЗРОБЦІ МОБІЛЬНИХ ДОДАТКІВ

Повсюдне поширення та популярність смартфонів серед кінцевих користувачів протягом останніх кількох років все більше привертає увагу розробників програмного забезпечення. Наразі в AppStore Apple [1] є близько 800 000 мобільних програм [1] (38% ринку [2]), 650 000 на Android Market [3] (52 %), 120 000 на Windows Marketplace [4] (3 %) і 100 000 на Blackberry AppWorld [5] (6 %). Останні оцінки показують, що до 2015 року понад 70% усіх мобільних пристроїв будуть смартфонами, здатними запускати мобільні програми [6].

Як і в будь-якому новому домені, розробка мобільних додатків має свій набір нових викликів, які нещодавно почали обговорювати дослідники [7], [8]. Однак більшість цих дискусій мають анекдотичний характер. Хоча існують значні якісні дослідження в різних сферах інженерії програмного забезпечення, наскільки нам відомо, не було проведено жодного дослідження для вивчення проблем, з якими на практиці стикаються розробники мобільних додатків. Мобільні програми загалом поділяються на три категорії: нативні, веб-додатки та гібридні [9], [10]. Власні програми працюють в операційній системі пристрою та потребують адаптації для різних пристроїв. Веб-програми потребують веб-браузера на мобільному пристрої. Гібридні програми - це веб-програми з «власною упаковкою».

Недавнє опитування [11] показало, що розробники в основному зацікавлені у створенні власних програм, оскільки вони можуть використовувати власні функції пристрою (наприклад, камеру, датчики, акселерометр, геолокацію). Тому в цій статті ми в основному зосереджуємося на рідних програмах. Відтепер ми використовуємо термін «мобільна програма» для позначення «власної мобільної програми». Мета нашого дослідження полягає в тому, щоб отримати розуміння поточної практики та проблем у розробці нативних мобільних додатків; ми провели дослідницьке дослідження, дотримуючись підходу Grounded Theory, який є дослідницькою методологією,

що походить із соціальних наук [12] і набуває все більшої популярності в дослідженнях програмної інженерії [13]. Таким чином, замість того, щоб почати із заздалегідь визначених гіпотез, ми поставили перед собою мету виявити процес і проблеми розробки мобільних додатків на різних платформах. З цією метою ми розпочали з проведення та аналізу інтерв'ю з 12 старшими розробниками мобільних додатків із 9 різних промислових компаній, які є експертами з таких платформ, як iOS, An -droid, Windows Mobile/Phone та Blackberry. На основі результатів цих інтерв'ю ми розробили та розповсюдили онлайн-опитування, яке заповнили 188 розробників мобільних додатків у всьому світі. наші результати розкривають труднощі роботи з декількома мобільними платформами під час мобільної розробки.

Хоча мобільні пристрої та платформи значною мірою рухаються до фрагментації, сучасному процесу розробки бракує адаптації для використання знань від платформи до платформи. Зараз розробники обробляють мобільний додаток для кожної платформи окремо та вручну перевіряють, чи зберігається функціональність на кількох платформах. Крім того, розробникам мобільних пристроїв потрібні кращі інструменти аналізу, щоб відстежувати показники своїх програм на етапі розробки. Крім того, тестування є серйозним викликом. Поточні інфраструктури тестування не забезпечують однаковий рівень підтримки для різних платформ, а поточні інструменти тестування не підтримують важливі функції для мобільного тестування, такі як мобільність, служби визначення місцезнаходження, датчики або різні жести та введення.

1.1 Дослідження дизайну

Враховуючи характер нашої дослідницької мети, ми вирішили провести якісне дослідження, дотримуючись підходу обґрунтованої теорії [12], [14]. Обґрунтована теорія найкраще підходить, коли мета полягає в тому, щоб

дізнатися, як люди керують проблемними ситуаціями та як люди розуміють і справляються з тим, що з ними відбувається [15]. Це також корисно, коли область

дослідження не була охоплена попередніми дослідженнями [16] і акцент робиться на створенні нової теорії [17], тобто на розумінні явища.

Таблиця 1.1

Учасники інтерв'ю

ID	Role	Platform Experience	Software Dev Exp (yr)	Mobile Dev Exp (yr)	Company (Mobile Dev Team Size)	Company's Platform Support
P1	iOS Lead	iOS, Android	6-10	6	A (20)	iOS, Android, Windows, Blackberry
P2	Android Lead	Android, iOS	6-10	6	A (20)	iOS, Android, Windows, Blackberry
P3	Blackberry Lead	Blackberry, iOS, Android	6-10	6	A (20)	iOS, Android, Windows, Blackberry
P4	iOS Lead	iOS	6-10	3-4	B (2-5)	iOS, Android
P5	Android Lead	Android	6-10	3	B (2-5)	iOS, Android
P6	iOS Dev	iOS	4-5	3-4	C (20+)	iOS, Android
P7	Windows Mobile Dev	Windows, Android	10+	2	D (1)	Windows
P8	Android Dev	Android	4-5	2-3	E (2-5)	iOS, Android
P9	Android Lead	Android, iOS, Windows	10+	5-6	F (6-10)	iOS, Android, Windows
P10	iOS Dev	iOS, Android	10+	3	G (1)	iOS, Android
P11	Android Lead	Android, Blackberry	10+	6+	H (1)	Android, Blackberry
P12	iOS Dev	iOS, Windows	10+	2-3	I (2-5)	iOS, Windows

Наш підхід до проведення дослідження на основі обґрунтованої теорії включає поєднання інтерв'ю та напівструктурованого опитування. Цільовими інтерв'ю були експерти з розробки мобільних додатків, а опитування було відкритим для загальної спільноти мобільних розробників. Наші інтерв'ю проводилися в ітеративному стилі, і вони є основою процесу збору й аналізу даних. Наприкінці кожного інтерв'ю ми просили респондентів надати відгук щодо наших питань; чого не вистачає, а що зайве. Аналітичний процес передбачає збір, кодування та аналіз даних після кожного інтерв'ю, одночасно розробляючи теорію. Зі стенограм інтерв'ю ми аналізуємо дані рядок за рядком, розбиваємо інтерв'ю на окремі одиниці значення (речення чи абзаци), надаємо коди тексту та позначаємо їх, щоб створити концепції для цих одиниць. Наші коди, де це доречно, взяті з самого тексту. В іншому випадку вони створюються авторами для фіксації виникаючих концепцій. Крім того, ці концепції потім групуються в описові категорії . Вони переоцінюються та відносяться до категорій вищого порядку, щоб створити нову теорію. Теоретична вибірка перетворюється на процес, що постійно змінюється, оскільки коди аналізуються, а категорії та поняття продовжують розвиватися [19]. Ми проводимо постійне

порівняння [12] між проаналізованими даними та новою теорією, доки додаткові дані, зібрані під час інтерв'ю, не додадуть нових знань про категорії.

Таким чином, як тільки відповіді опитуваних починають нагадувати

попередні відповіді, досягається стан насичення [23], і саме тоді ми зупиняємо процес опитування. Базуючись на теорії, що виникла на етапі інтерв'ю, ми розробили напівструктуроване опитування як ще одне джерело даних, щоб оскаржити цю теорію. Перш ніж опублікувати опитування та оприлюднити його, ми попросили чотирьох зовнішніх людей - одного старшого аспіранта та трьох розробників мобільних додатків - переглянути опитування, щоб переконатися, що всі запитання доречні та зрозумілі. Більшість запитань нашого опитування є закритими, але є також кілька додаткових відкритих запитань для збору «інсайтів» і «досвіду» учасників. Відповіді на ці відкриті запитання вводяться в наш етап кодування та аналізу для уточнення результатів, де це можливо. Це опитування, роздане учасникам, доступне онлайн.

Демографія учасників Інтерв'ю. Ми опитали 12 експертів з 9 різних компаній. Кожне інтерв'ю займало в середньому близько 30 хвилин. Ми записували аудіо під час інтерв'ю, а потім транскрибували їх для подальшого аналізу. У таблиці I представлено роль кожного учасника в їхній компанії, мобільні платформи, в яких вони мають досвід роботи, кількість років, які вони мають у розробці програмного забезпечення та розробці мобільних додатків, розмір команди мобільних розробників і, нарешті, усі мобільні платформи, які підтримує кожна компанія. Що стосується досвіду розробки мобільного додатку, п'ятеро мають близько 6 років, чотири – 3-4 роки, а троє – 2-3 роки. П'ять учасників переважно є експертами з iOS, п'ятеро – з Android, один – з Windows і, нарешті, один – з Blackberry.

Опитування. Наше опитування повністю заповнили 188 респондентів. Ми опублікували опитування 13 грудня 2012 року для багатьох груп мобільних розробників. Ми націлилися на популярні групи Mobile Development Meetup, групи LinkedIn, пов'язані з нативною мобільною розробкою, і поділилися

опитуванням через наші облікові записи Twitter. Ми проводили опитування два з половиною місяці. Під час нашої спроби розповсюдити наше онлайн-опитування було цікаво побачити реакцію людей; їм сподобалася наша публікація в групах LinkedIn і вони дали підбадьорливі коментарі, наприклад

« Сподіваюся, це допоможе полегшити життя розробників мобільних додатків ».

Демографія учасників опитування така: 92% чоловіків, 5% жінок. Вони походять із США (48%), Індії (11%), Канади (10%), Ізраїлю (5%), Нідерландів (3%), Великобританії (3%), Нової Зеландії (2%), Мексики (2%) та ще 15 країн. Що стосується досвіду роботи в розробці програмного забезпечення, то 52% мають більше 10 років, 15% - 6-10 років, 20% - 2-5 років, 13% - менше 2 років. Їхній досвід у нативній мобільній розробці варіюється від: 6% понад 6 років, 19% від 4-6 років, 59% мають від 1-3 років до 16% менше 1 року. Платформи, в яких вони мають досвід, включають 72% iOS, 1 http://www.ece.ubc.ca/~merfani/survey.pdf 65% Android, 26% Windows, 13% Blackberry, а 6% обрали інші (наприклад, Symbian, J2ME).

1.2 Загальні виклики для мобільних розробників

У цьому підрозділі ми представляємо найпомітніші загальні проблеми, з якими стикаються розробники мобільних додатків, випливаючи з результатів нашого дослідження. Рух до фрагментації, а не до об’єднання. 76% учасників нашого опитування вважають існування кількох мобільних платформ проблемою для розробки мобільних додатків, тоді як 23% вважають, що це можливість для розвитку технологій, які стимулюють інновації. Більше половини учасників зазначили, що мобільні платформи рухаються до фрагментації, а не до уніфікації:

- Фрагментація між платформами: кожна мобільна платформа відрізняється щодо інтерфейсу користувача, взаємодії з користувачем, стандартів взаємодії людини з комп’ютером (HCI), очікувань користувачів,

метафор взаємодії з користувачем, мов програмування, API/SDK та підтримуваних інструментів.

- Фрагментація в межах однієї платформи: на одній платформі існують різні пристрої з різними властивостями, такими як пам’ять, швидкість процесора та графічна роздільна здатність. Також можлива фрагментація на рівні

операційної системи. Відомим прикладом є фрагментація на пристроях Android із різними розмірами екрана та роздільною -здатністю. Майже кожен розробник Android під час наших інтерв’ю та опитування згадував це як величезну проблему, з якою їм доводиться стикатися регулярно.

Крім того, фрагментація пристрою є проблемою не лише для розробки, але й для тестування. Усі наші учасники вважають, що версії платформи та оновлення є головною проблемою; Наприклад, респондент сказав: « на рівні ОС деякі методи застаріли або навіть видалені». Тому розробникам потрібно тестувати свої програми на різних версіях ОС і розмірах екрана, щоб переконатися, що їхня програма працює. Піддослідний P5 сказав, що вони здебільшого ведуть « список кандидатів різних пристроїв і розмірів ». P11 пояснив: « Оскільки ми відстежуємо нашу програму на основі відгуків користувачів, ми зосереджуємося на тестуванні на найпопулярніших пристроях». Таким чином, поточний стан мобільних платформ додає ще один вимір до вартості, з великою різноманітністю пристроїв і версій ОС для тестування. P11 продовжив: «Зараз ми підтримуємо 5 або 6 різних версій (додатків) лише тому, що існують різні версії ОС, і для кожної з цих версій ОС ми також маємо 3-4 різні розміри екрану, щоб переконатися, що програма працює в кожній із версій Android. » Респондент зазначив: « Ми розділили код навколо версії 2.3 (Android). Отже, у нас є дві різні версії програм: версія до 2.3 і версія після 2.3. І з точки зору нашої політики, ми прийняли це рішення, оскільки занадто складно перенести деякі функції».

Підтримка моніторингу, аналізу та тестування. «Історично майже ніхто не займався тестуванням мобільних додатків», - заявив P10 і пояснив, що до

недавнього часу було дуже мало тестувань і дуже мало спеціалізованих команд тестування. Однак зараз це змінюється, і вони почали прагнути до якості та тестування. Підтримка автоматичного тестування наразі дуже обмежена для рідних мобільних програм. Багато учасників вважають це однією з головних проблем. Поточні інструменти та емулятори не підтримують важливі функції для мобільного тестування, такі як мобільність, служби визначення

місцезнаходження, датчики або різні жести та введення. Наші результати свідчать про сильну потребу розробників мобільних додатків у кращому аналізі та підтримці тестування. Багато хто згадував про необхідність моніторингу, вимірювання та візуалізації різних показників своїх додатків за допомогою кращих інструментів аналізу.

Відкриті/закриті платформи розробки. Android має відкритий вихідний код, тоді як iOS і Windows закриті. Деякі учасники стверджували, що Apple і Microsoft повинні відкрити свої платформи. P5 пояснив: "Ми маємо реальні проблеми з iOS, а не з Android. Тому що у вас немає API для керування, тому вам потрібно стрибати в цикл і шукати задні двері, тому що вхідні двері замкнені. Будь-що, що дозволяє Apple, інколи недостатньо. " Приклад такого браку контролю наведено: "дізнатися, чи ми підключені до Bluetooth". З іншого боку, P9 пояснив, що оскільки Android є відкритим кодом і кожен виробник змінює вихідний код відповідно до власних бажань і випускає його, іноді вони не дотримуються стандартів. Наведено простий приклад: « стандартний Android використовує коми для розділення елементів у списку, але телефони Samsung використовують крапку з комою». Респондент зазначив: «Багато пристроїв Android були погано налаштовані операторами та виробниками оригінального обладнання » .

Програми з інтенсивним використанням даних. Працювати з даними складно для програм, які інтенсивно використовують дані. Як пояснив респондент: «На пристрої не можна зберігати так багато даних, і використання мережевого з’єднання для синхронізації з іншим джерелом даних у серверній

частині є складним завданням». Щодо офлайн-кешування в гібридних рішеннях P1 сказав: «Наші програми мають багато даних, і офлайн-кешування, здається, не працює добре».

Не відставати від частих змін. Однією з проблем, про які згадують багато розробників, є вивчення більшої кількості мов і API для різних платформ, а також регулярні зміни в кожному наборі програмного забезпечення. (SDK). « Більшості розробників мобільних пристроїв колись знадобиться підтримувати більше ніж

одну платформу», - заявив респондент. "Кожна платформа абсолютно різна

(ринки, мови, інструменти, рекомендації щодо дизайну), тому вам потрібні експерти для кожної з них. По суті, це все одно, що намагатися написати книгу одночасно японською та російською; вам потрібен рідний японський і рідний російський, інакше якість буде потворною", – пояснив інший респондент. Як наслідок, вивчення мови, інструментів, методів, найкращих практик і правил HCI іншої платформи є складним завданням. Багато розробників скаржилися на відсутність інтегрованого середовища розробки, яке підтримує різні мобільні платформи. Винятком був PI , який пояснив: « Зараз ми розробляємо на двох основних платформах: iPhone та Android. Це насправді не так вже й складно, рідні SDK досить зрілі, і їх легко освоїти».

1.3 Розробка для кількох платформ

Нативні та гібридні мобільні програми. Предмети PI та P8 підтримують розробку гібридних програм. Решта 10 респондентів виступають за створення чистих нативних програм і вважають, що поточна гібридна модель має тенденцію виглядати та поводитися більше як веб-сторінки, ніж мобільні програми. Pll стверджував, що «нативний підхід пропонує найбільші можливості» і P4 заявив, що «взаємодія з нативними програмами набагато краща [порівняно] з веб-програмою». У ряді випадків учасники повністю ¬відійшли від гібридного до нативного підходу. Наведений повторюваний приклад —

нещодавній перехід Facebook від мобільного додатка на основі HTML5 до нативного. З іншого боку, PI стверджував, що «це дійсно залежить від складності та типу програми» , наприклад, «додатки для обміну інформацією можуть легко прийняти гібридну модель, щоб просувати вміст новин і оновлення на кількох платформах». В опитуванні 82% відповіли, що мають досвід нативної розробки, 11% пробували гібридні рішення, а 7% розробляли мобільні веб-додатки.

Більшість респондентів підтримують нативний підхід: «Мобільний Інтернет не схожий на жодну з платформ». Інші сказали, що: «HTML5 має

великий потенціал і, ймовірно, вирішить багато поточних проблем у майбутньому, оскільки економить час і кошти на розробку»; або: «Оскільки багато великих гравців багато інвестують у HTML5, він може зайняти велику частину інтерфейсу, коли він стане стабільним». Більшість учасників стверджували, що коли витрати на розробку не є проблемою, компанії прагнуть розробляти рідні програми. Звичайно, це також залежить від типу програми; там, де потрібна краща взаємодія з користувачем або особливі функції пристрою, рідний, здається, очевидний вибір. Нарешті, коли ми запитали наших учасників, чи -замінять нативну розробку програм гібридними рішеннями чи розробкою мобільних веб-сайтів через проблеми, усі респонденти та 70% учасників опитування не погодилися, а 10% зазначили, що завжди буде поєднання нативних і гібридних підходів.

Обмеження можливостей пристроїв платформи. Не всі пристрої та операційні системи платформи мають однакові можливості. Наприклад, Android має різні версії, а деякі з них браузери погано підтримують HTML5. Більшість учасників, які підтримують гібридний підхід, вважають, що після завершення адаптації було б більше інтересу спільноти до гібридної розробки.

Повторне використання коду проти написання з нуля. 67% наших Учасники інтерв'ю випробували обидва методи написання нативної мобільної програми з нуля для іншої платформи та повторне використання деяких частин того самого коду на різних платформах. Більшість заявили, що неможливо або

складно перенести функціональні можливості на різні платформи, і що коли код повторно використовується на іншій платформі, якість результатів є незадовільною. Рисунок 1.1 показує, що з 63% респондентів опитування, які мали досвід розробки мобільних додатків на різних платформах, 34% написали те саме додаток для кожної платформи з нуля, а 20% мали досвід перенесення частини існуючого коду. Респондент сказав, що «кожна платформа має різні вимоги до розробки, і портування не завжди забезпечує якість»; або: «На даний момент я вважаю, що найкраще створювати програми з нуля, орієнтовані на окрему ОС». Pll стверджував, що "ми перенесли дуже невелику кількість коду між Android і Blackberry, але ми зазвичай пишемо код з нуля. Хоча вони обидва використовують Java, вони не працюють однаково. Навіть якщо базові низькі рівні Java однакові, вам доведеться переписати код". На додаток до відмінностей на рівні мови програмування (наприклад, Ob jective-C проти Java), P9 уточнив, чому міграція коду не працює: "Простим прикладом є те, як вони [платформи] обробляють push-повідомлення. В Android push-повідомлення активує частини програми та запитує час ЦП. В iOS відправник передає дані на сервер Apple push. Потім відправник надсилає їх на пристрій, і ЦП не витрачає часу на обробку даних. потрібно, Ці відмінності між платформами змушують розробників переписувати ту саму програму для різних платформ без повторного використання коду або з невеликою кількістю. Це вважається одним із головних недоліків нативної розробки додатків.

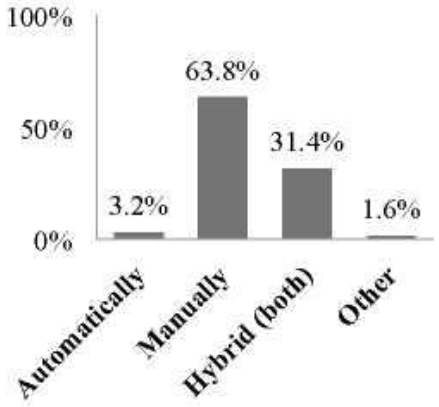

Рисунок 1.1 - Як тестуються ваші нативні мобільні додатки

Послідовність поведінки в порівнянні з конкретними рекомендаціями HCI. В ідеалі даний мобільний додаток повинен забезпечувати однакову функціональність ¬і поведінку незалежно від цільової платформи, на якій він працює. Однак через внутрішні відмінності в різних мобільних пристроях і операційних системах «загального дизайну для всіх платформ не існує»; Наприклад, P12 заявив, що «дизайн Android не може повністю працювати на iPhone». Головним чином це пов'язано з тим, що рекомендації HCI досить різні на різних платформах, оскільки для мобільного світу не існує стандартів, як,

наприклад, для Інтернету. Таким чином, розробники постійно стикаються з двома конкуруючими вимогами:

- Знайомство для користувачів платформи: Кожна платформа відповідає набору конкретних рекомендацій HCI, щоб забезпечити узгоджений вигляд і відчуття для всіх програм на одному пристрої. Це полегшує кінцевим користувачам навігацію та взаємодію з різними програмами.
- Послідовність поведінки на різних платформах: З іншого боку, розробники хотіли б, щоб їхні програми поводилися однаково на різних платформах, наприклад, взаємодія користувача з певною функцією на Blackberry має бути такою ж, як на iPhone та Android.

Таким чином, створення багаторазового базового дизайну, який легко транслюватиметься на всі платформи, зберігаючи поведінкову послідовність, є складним завданням. Як зазначив P9: «Додаток слід переробити для кожної платформи/ОС, щоб він добре працював»», Респондент сказав: «Ми переглядаємо дизайн екрану за екраном для кожної нової платформи»; або: " Різні платформи мають різні переваги та можливості. Нерозумно намагатися зробити програми абсолютно однаковими на різних платформах"; і: '7/ вимагає розгляду мультиплатформи на етапі проектування, і розумні рішення повинні прийматися там, де необхідний дизайн для конкретної платформи».

Час, зусилля та бюджет примножуються. Через відсутність підтримки автоматизованої міграції між платформами розробникам доводиться перепроектувати та повторно впроваджувати більшу частину програми. Таким чином, створення якісних продуктів на різних платформах є не тільки складним завданням, але й трудомістким і дорогим, тобто «розробка мобільних додатків на різних платформах - це як мати набір різних розробників для кожної платформи». заявив Плл. Як наслідок, « перекодування проти надзвичайно різних наборів API» збільшує вартість і час виходу на ринок на етапах проектування, розробки, тестування та обслуговування, що, безумовно, є великою проблемою для компаній-початківців і невеликих компаній.

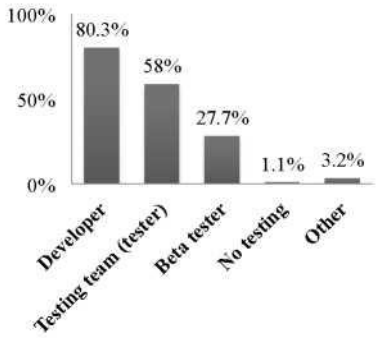

Рисунок 1.2 - Хто відповідає за тестування нативних мобільних додатків

Переважає ручне тестування. Як показано на малюнку 2, 64% учасників нашого опитування тестують свої мобільні програми вручну, 31% застосовують гібридний підхід, тобто поєднання ручного й автоматичного тестування, і лише 3% беруть участь у повністю автоматизованому тестуванні. P3 пояснив: " На даний момент вручну є найкращим варіантом. Це схоже на тестування нової гри, тестування на консолях і пристроях. Я вважаю, що таке тестування може бути меншим, але вам доведеться турбуватися про більше платформ і версій". А респондент зазначив: « Організації, великі й малі, вірять лише в ручне тестування на невеликій підгрупі пристроїв»; а інший сказав: «Це безлад. Навіть великі організації важко переконати проводити автоматизоване тестування».

Розробники є тестувальниками. Існують різні комбінації процесів тестування та підходів, які зараз застосовуються в галузі. Їх можна класифікувати на основі розміру компанії, клієнтів, культури розробки, політики

тестування, типу програми та підтримуваних мобільних платформ. Ці підходи до тестування виконуються різними людьми, такими як розробники, команди тестування, бета-тестери, клієнти, а також сторонні служби тестування. Як зазначено в Таблиці I, компанії наших опитаних відрізняються від невеликих за розміром з 1-2 розробниками до великих мобільних розробників або команд з понад 20 розробниками. Як і очікувалося, великі компанії можуть дозволити собі спеціалізовані команди тестування, тоді як у менших компаніях тестування в основному проводять розробники або клієнти (кінцеві користувачі). На рисунку 3 зображено результати нашого опитування стосовно ролей, відповідальних за тестування. 80% респондентів вказали, що розробники є тестувальниками, 53% мають спеціальні команди тестування або тестувальників, а 28% покладаються на бета-тестерів. Більшість учасників, разом із командами тестування чи без них, заявили, що після розробки нової функції розробники спочатку проводять власне тестування та перевіряють її працездатність і правильність. Це здебільшого ручне тестування на симуляторах і, якщо доступне, на фізичних пристроях. Так інтерв'ю показують, що наші учасники розглядають кожну платформу абсолютно окремо, коли справа доходить до тестування. Наразі не існує узгодженого методу тестування певного мобільного додатка на різних платформах; здатність впоратися з відмінностями на рівні інтерфейсу користувача вважається серйозною проблемою. Тестувальники пишуть « скрипти, специфічні для кожної платформи», і вони « знайомі з функціями програми, але тестують кожну платформу окремо». Ми також помітили, що в одній компанії зазвичай існують окремі команди, кожна з яких займається певною платформою зі своїм набором інструментів і технік; P6, розробник iOS, сказав: «Я не впевнений щодо Android, оскільки команди в нашій компанії настільки різні, і я навіть не знаю, що відбувається з іншою стороною». Відповіді надали 63% учасників нашого опитування, які розвивають те саме рідний мобільний додаток для кількох платформ, підтвердив результати інтерв'ю, заявивши: «Тестові приклади застосовуються до кожної платформи, але вони повинні бути реалізовані унікально на кожній платформі», або: «Те саме, що для

однієї платформи, але кілька разів», і: «Мені доведеться зробити це двічі чи більше, залежно від того, на скількох платформах я буду це робити», або: "Розглядайте їх як окремі проекти, якими вони є по суті, якщо є рідними. Виконуйте тестування незалежно".

1.4 Висновки по розділу

Дослідження, засноване на обґрунтованій теорії, поєднувало глибокі інтерв'ю з 12 експертами та широке опитування 188 мобільних розробників для формування нової теорії про виклики у галузі. Такий підхід дозволив глибоко зрозуміти досвід, проблеми та потреби фахівців, що працюють із різними мобільними платформами, та виявити тенденції на основі реальних практик та ворожень.

Фрагментація мобільних платформ є одним із головних викликів для розробників, оскільки різноманітність пристроїв, версій ОС, мов програмування та інтерфейсів ускладнює створення універсального рішення. Додаткові труднощі спричиняють обмежену підтримку автоматизованого тестування, закритість деяких платформ, потребу постійно оновлювати знання щодо нових API, а також складнощі з обробкою великих обсягів даних у мобільних додатках. Все це збільшує витрати часу, зусиль та ресурсів при розробці.

На основі опитування та інтерв'ю більшість розробників надають перевагу нативній розробці через її вищу продуктивність, кращу взаємодію з користувачем та специфіку платформ. Незважаючи на потенційні переваги гібридних рішень, такі як економія часу та коду, складність підтримки погодженого дизайну та функціональності на різних платформах, а також труднощі з автоматизованим тестуванням змушують розробників підходити до кожної ОС як до окремого проекту.

РОЗДІЛ 2 . СТРУКТУРА ОЦІНЮВАННЯ ДЛЯ CPDT

2.1 Розробка мобільних додатків

Ринок мобільних додатків зараз швидко розвивається, оскільки мобільні платформи продовжують покращувати продуктивність, а потреба користувачів у різноманітних мобільних додатках зростає. Новітні мобільні платформи дозволяють широко використовувати мережеві ресурси, а отже, пропонують сильну альтернативу робочим станціям і пов’язаному з ними програмному забезпеченню. Розробка програмного забезпечення для мобільних платформ має унікальні функції та обмеження, які застосовуються до більшості етапів життєвого циклу. Середовище розробки та технології, які підтримують програмне забезпечення, відрізняються від «традиційних» налаштувань. Найважливіші відмінні характеристики визначені в [1]. Особливості середовища включають: високий рівень конкурентоспроможності; обов'язково короткі терміни доставки; і додала складність у визначенні зацікавлених сторін та їхніх вимог. Команди розробників повинні стикатися з проблемами динамічного середовища з частими змінами потреб і очікувань клієнтів [1]. Технологічні обмеження застосовуються до мобільних платформ у вигляді обмежених фізичних ресурсів і швидкої зміни специфікацій. Існує також велика різноманітність пристроїв, кожен з яких має певні апаратні характеристики, прошивку та операційні системи. Інший погляд на обмеження, пов'язані з мобільними додатками, представлений у [5] Автор згадує два типи обмежень, еволюційні та властиві. Обмеження, що змінюються, такі як пропускна здатність, покриття та безпека, наразі застосовуються до мобільних технологій, але, ймовірно, вони будуть розглянуті та, можливо, вирішені в найближчому майбутньому. З іншого боку, властиві обмеження, такі як обмежений простір екрана, обмежені можливості введення даних (через обмежену клавіатуру, наприклад), ємність пам’яті, потужність обробки та обмежений запас потужності, є постійними, принаймні щодо настільних середовищ. Необхідно

використовувати різні підходи, щоб зменшити вплив внутрішніх обмежень. Через значні відмінності в середовищі та специфікаціях платформ розробка мобільних додатків вимагає відповідної методології розробки. Беручи до уваги основні особливості сценарію розробки мобільних додатків, можна визначити відповідну парадигму розробки. Ці функції представлені в (Abrahamsson, 2005): програмне забезпечення випускається в невизначеному та динамічному середовищі з високим рівнем конкуренції. Команди, які розробляють мобільні додатки, як правило, невеликі та середні, розташовані разом і зазвичай використовують об'єктно-орієнтовані інструменти та практики. Самі додатки невеликі за розміром, не є критично важливими для безпеки та не мають відповідати обмеженням сумісності чи надійності. Вони постачаються у швидких випусках, щоб задовольнити потреби ринку, і орієнтовані на велику кількість кінцевих користувачів. Автор пропонує гнучкі методи як відповідний підхід до розробки, порівнюючи наведені вище характеристики з гнучкими характеристиками «домашньої землі»: маломасштабне програмне забезпечення прикладного рівня, розроблене у високодинамічному середовищі невеликою та середньою командою з використанням об'єктно-орієнтованих підходів за відносно короткі цикли розробки. У наступному розділі подано короткий огляд гнучких методів, зосереджуючись на їх придатності для розробки мобільних додатків.

Гнучка розробка для мобільних додатків

Гнучкі методи являють собою відносно новий підхід до розробки програмного забезпечення, який набув широкого поширення в останнє десятиліття. Ідеї, що лежать в основі цих методів, походять від принципів Lean Manufacturing (у 1940-х роках) і Agile Manufacturing (1990-ті роки), які наголошували на адаптованості підприємств до динамічного середовища (Salo, 2006). Унікальні особливості гнучких методів випливають із переліку принципів, викладених у «Маніфесті гнучкості»: особи та взаємодія важливіші, ніж процеси та інструменти, робоче програмне забезпечення є більш цінним, ніж вичерпна документація, співпраця з клієнтами є кращою, ніж переговори щодо контрактів,

а адаптивність цінується вище, ніж створення та дотримання плану [3] визначають фундаментальні концепції гнучкої розробки: прості принципи проектування, велика кількість випусків за короткий проміжок часу, широке використання рефакторинга, парне програмування, розробка, керована тестуванням, і бачення змін як переваги. Інше визначення гнучких методів надано в [7]: гнучкий метод розробки є поступовим (кілька випусків), кооперативним (тісна співпраця між розробником і клієнтом), простим (легким для розуміння та модифікації) та адаптивним (дозволяє часті зміни).

Використання гнучких методів у розробці програмного забезпечення отримало аргументи як на підтримку, так і проти. Основним аргументом проти гнучких методів є стверджувана відсутність наукового обґрунтування пов'язаних дій і практик, а також складність інтеграції планових практик із гнучкими. Дійсно, деякі проекти представляють суміш характеристик на основі плану та гнучкого домашнього майданчика, і в цьому випадку необхідно досягти балансу у використанні обох типів методів [11]. Існує також певна невизначеність у відрізненні гнучких методів від спеціального програмування. Однак, як зазначено в [19] гнучкі методи дійсно забезпечують організований підхід до розробки. Під час спроби порівняти характеристики мобільних додатків із характеристиками гнучкого методу труднощі виникають частково через те, що межі гнучких методологій не встановлені чітко. Повний огляд досліджень у цій галузі представлено в [9] Автори поділяють дослідження на чотири категорії: впровадження та адаптація, людські та соціальні фактори, сприйняття гнучких методів та порівняльні дослідження. Результати показують, що впровадження гнучких методів у проекти програмного забезпечення приносить переваги, особливо якщо гнучкі практики не повністю замінюють традиційні, а працюють у поєднанні з ними. Однак, за словами авторів, дослідження в цій галузі здебільшого зосереджені на екстремальному програмуванні (XP), обмежені в кількості та мають сумнівну якість.

У [15] автор проводить пряме порівняння між характеристиками гнучкого методу та функціями мобільних додатків, зосереджуючись на мінливості

середовища, кількості створеної документації, обсязі планування, розмірі групи розробників, масштабі розробки програми, ідентифікації клієнта та орієнтації на об’єкт. Окрім ідентифікації клієнтів, усі інші гнучкі характеристики роблять методи придатними для розробки мобільних додатків. Клієнта можна ідентифікувати як розповсюджувача програмного забезпечення. Однак, особливо у випадку мобільних додатків, проблема ідентифікації клієнта набагато складніша, як буде детально описано в наступному розділі цієї роботи. Нова методологія розробки, спеціально розроблена для розробки мобільних додатків, називається Mobile-D, представлена в [4]). Метод базується на гнучких практиках, черпаючи елементи з добре відомих гнучких методів, таких як Extreme Programming і Crystal Methodologies, а також із «важчого» Rational Unified Process. Додаткову інформацію про XP можна знайти в [2], тоді як Crystal Methodologies детально описано в [6]. Раціональний уніфікований процес пояснюється з практичної точки зору в [Практики, пов’язані з Mobile-D , включають тестову розробку, 17] парне програмування, безперервну інтеграцію, рефакторинг, а також завдання вдосконалення програмного процесу.

2.2 Огляд Mobile-D

Відповідно до [6], процес Mobile-D має використовуватися командою щонайбільше десяти розробників, які працюють разом, щоб забезпечити доставку продукту протягом десяти тижнів. Існує дев’ять основних елементів, задіяних у різних практиках протягом усього циклу розробки:

1. Поетапність і розміщення
2. Лінія архітектури
3. Мобільна розробка на основі тестування
4. Безперервна інтеграція
5. Парне програмування
6. Метрики
7. Гнучке вдосконалення програмного процесу

8. Замовник поза сайтом

9. Фокус, орієнтований на користувача

Архітектурна лінія в методології є новим доповненням до вже встановлених гнучких практик. Лінія архітектури використовується для отримання інформації організації про архітектурні рішення як із внутрішніх, так і зовнішніх джерел, а також для використання цих рішень у разі потреби.

Mobile-D включає п’ять етапів: дослідження, ініціалізація, виробництво, стабілізація та тестування та виправлення системи. Кожен із цих етапів має низку пов’язаних етапів, завдань і практик. Повні специфікації методу доступні в [21]. На першому етапі, Explore , команда розробників повинна створити план і визначити характеристики проекту. Це робиться в три етапи: встановлення зацікавлених сторін , визначення обсягу та створення проекту. Завдання, пов’язані з цією фазою, включають встановлення клієнтів (тих клієнтів, які беруть активну участь у процесі розробки), початкове планування проекту та збір вимог, а також встановлення процесу. На наступному етапі, Initialize , команда розробників і всі активні зацікавлені сторони розуміють продукт, який розробляється, і готують ключові ресурси, необхідні для виробничої діяльності, такі як фізичні, технологічні та комунікаційні ресурси. Цей етап ділиться на три етапи: налаштування проекту , початкове планування та пробний день . Фаза Productionize в основному включає діяльність із впровадження. Наприкінці цього етапу більша частина реалізації має бути завершена. Ця фаза поділяється на дні планування , робочі дні та дні випуску . Дні планування спрямовані на вдосконалення процесу розробки, визначення пріоритетів і аналіз вимог, планування вмісту ітерації та створення приймальних тестів, які будуть запущені пізніше в дні випуску. У робочі дні для реалізації функціональних можливостей використовується практика розробки, керованої тестуванням (TDD), відповідно до попередньо встановленого плану для поточної ітерації. Використовуючи TDD разом із безперервною інтеграцією, розробники створюють модульні тести, пишуть код, який проходить тести, і інтегрують новий код із наявною версією продукту, усуваючи будь-які помилки, які можуть виникнути в процесі

інтеграції. Нарешті, у дні випуску створюється робоча версія системи та перевіряється шляхом приймального тестування. Останні дві фази, Stabilize і System Test & Fix , використовуються для фіналізації продукту та тестування відповідно. Вони включають етапи, подібні до етапу Productionize , з деякими змінами для створення документації та тестування системи.

Mobile-D вже застосовувався в проектах розробки, і було помічено деякі переваги, такі як покращена видимість прогресу, раннє виявлення та усунення технічних проблем, низька щільність дефектів у кінцевому продукті та постійний прогрес у розробці [6] Інші застосування методу представлені в [27].

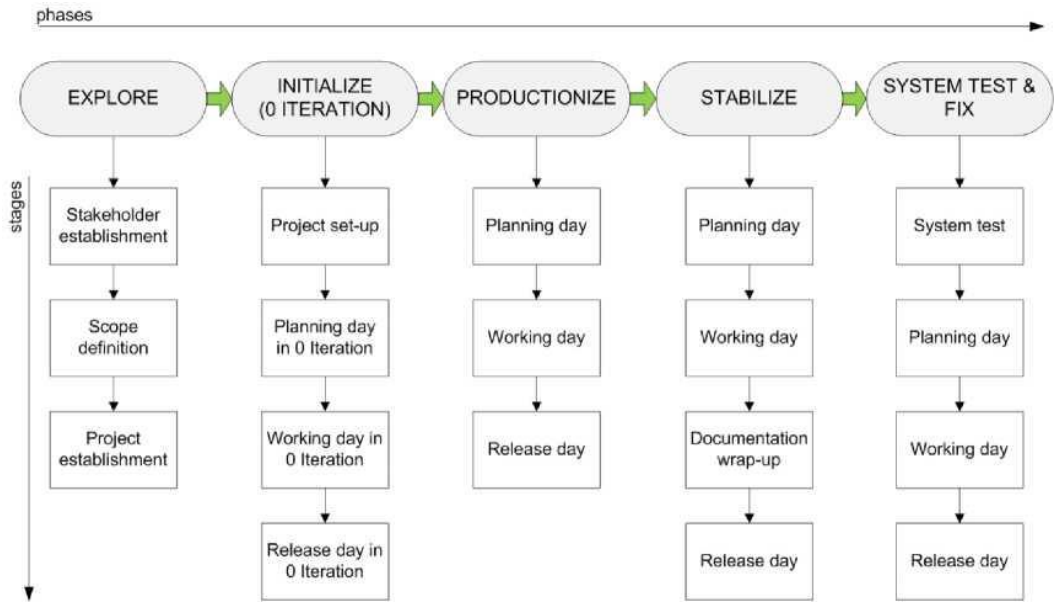

Рисунок 2.1 - Фази та стадії Mobile-D; Джерело: (VTT Electronics, 2006)

2.3 Підходи до вдосконалення

Щоб отримати набір удосконалень даної методології розробки, необхідно спочатку проаналізувати ключові характеристики методу, які дали успішні результати в попередніх проектах. Для методів розробки мобільних додатків ключові характеристики успіху визначені в [23]. Це гнучкість підходу, ринкова свідомість, підтримка лінійки програмних продуктів, розробка на основі архітектури, підтримка багаторазового використання, включення сеансів перегляду та навчання та рання специфікація фізичної архітектури. Деякі з цих

ключових функцій уже можна знайти в методі Mobile-D (гнучкість, рання специфікація фізичної архітектури, розробка на основі архітектури, а також сеанси перегляду та навчання); однак метод можна вдосконалити, якщо інтегрувати більше цих ключових функцій успіху. Можливі доповнення до Mobile-D включають кращу обізнаність про ринок, підтримку лінійки програмних продуктів і підтримку повторного використання. Останні дві функції можуть бути реалізовані з кінцевою метою мінімізації витрат, але можуть бути недосяжними для мікрокомпаній або компаній з низьким досвідом розробки мобільних додатків, оскільки ці компанії можуть не мати створених бібліотек компонентів для успішного застосування принципів повторного використання програмного забезпечення. Огляд і навчальні сеанси можна знайти в методології як післяітераційні робочі операції та підсумки, призначені для покращення процесу розробки шляхом визначення його сильних і слабких сторін, а також для повідомлення про прогрес і проблеми всередині команди.

Перелік ідеальних рис, знайдений у [25] можна розширити. У випадку мобільних додатків ідентифікація кінцевого користувача не є простою. Це також було зазначено в [6], коли порівнювали гнучкі характеристики домашнього майданчика з вимогами до розробки мобільних додатків. Для гнучких методів клієнт має бути легко ідентифікованим, що не завжди стосується мобільних додатків. Щоб вирішити цю проблему, перелік ключових характеристик методології розробки мобільних пристроїв можна розширити, щоб включити збалансованість вимог клієнта, кінцевого користувача та іноді постачальника платформи. Ці три сутності рідко збігаються при роботі з мобільними додатками. У випадках, коли організація, яка запитує програмний продукт, відрізняється від кінцевого користувача, установа-замовник має бути поінформована про можливий інший набір вимог кінцевого користувача. Для команди розробників це означає, що необхідно встановити баланс між клієнтом, вимогами, очікуваннями кінцевого користувача та обмеженнями постачальника платформи. Ми можемо припустити, що виявлення та включення кінцевих користувачів у процес розробки виявиться корисним для забезпечення їх вимог і

очікувань, а також для пошуку можливих дефектів. Підхід до інтеграції вимог кінцевого користувача в процес розробки детально описано в наступному розділі. Цю проблему вирішували організації за допомогою сеансів тестування користувачів у спеціальних лабораторіях, спостерігаючи за тим, як користувачі взаємодіють із системою. У наведеному нижче списку представлено адаптоване визначення пріоритетів для успішних методологій розробки мобільних додатків.

1. Спритність
2. Ринкова свідомість
3. Рання специфікація фізичної архітектури
4. Підтримка зворотного зв'язку кінцевого користувача
5. Підтримка лінійки програмних продуктів
6. Підтримка повторного використання
7. Розробка на основі архітектури
8. Перегляд і навчання

Ця робота зосереджена на вдосконаленні методів розробки мобільних додатків загалом і методу Mobile-D зокрема. Використовуючи наведений вище список як орієнтир, проект досліджує можливі покращення в окремих підсферах. Вивчаючи категорії успішних мобільних додатків, продукт, що розробляється, можна віднести до однієї категорії, і можна вжити конкретних заходів для покращення якості та мінімізації ризику для поточної програми. Процедуру вирівнювання можна включити як завдання на певному етапі Mobile-D . Принципи з інших методологій розробки можна інтегрувати в Mobile-D, якщо сценарій сприятиме їх використанню. Весь метод може імпортувати концепції з інших підходів і стати адаптованим до поточного проекту. Необхідно визначити внесок кінцевих користувачів у процес розробки, щоб забезпечити успіх продукту. Точніше, необхідно встановити час, ступінь і потенційні вигоди від участі кінцевих користувачів. Досягнення балансу між трьома наборами вимог, клієнта, кінцевого користувача та постачальника платформи, важливо, особливо в тих випадках, коли вимоги не збігаються. Мобільні платформи страждають від обмежень продуктивності. Mobile-D широко використовує практику розробки,

керованої тестуванням; однак тестові випадки не враховують використання ресурсів, частково тому, що інструментальна підтримка для цього завдання є поганою. Щоб удосконалити методологію, TDD необхідно адаптувати для врахування тестування ресурсів, особливо для ситуацій з обмеженими ресурсами. Ця модифікація також впливає на пов’язані завдання на різних етапах методології Mobile-D . Коли виявлено потенційні вузькі місця ресурсів, їх можна усунути шляхом написання кращого коду або пізніше шляхом рефакторингу та використання шаблонів. Нарешті, методологію можна вдосконалити, вивчивши переваги принципів лінійки програмних продуктів і встановивши їх правильну інтеграцію з Mobile-D.

2.4 Висновки по розділу

Розробка мобільних додатків потребує адаптивних методів через динамічну среду, технічні обмеження та різноманітність платформ. Гнучкі методи, зокрема Mobile-D, добре підходять до таких умов, оскільки підтримують швидкі ітерації, мінімальну документацію, активну участь клієнта та адаптацію до змін, що робить їх ефективним інструментом для створення сучасних мобільних рішень. Методологія Mobile-D, орієнтована на невеликі команди та швидку розробку, поєднує гнучкі практики з унікальними елементами, такими як архітектурна лінія, безперервна інтеграція та тестування, кероване розробкою.

Удосконалення методології Mobile-D передбачає інтеграцію ключових характеристик успішної розробки мобільних додатків, таких як ринкова обізнаність, повторне використання коду та врахування зворотного зв'язку кінцевого користувача.

РОЗДІЛ 3. ПІДХОДИ ДО МОБІЛЬНОЇ РОЗРОБКИ ТА ОСОБЛИВОСТІ ANDROID

3.1 Категорії мобільних додатків

Існує багато способів класифікації мобільних програм. Тим не менш, будь-який правдоподібний розділ може призвести до кращих результатів у процесі розробки завдяки більшій зосередженості на проблемах, які є специфічними для відповідного типу програми. Залежно від досвіду команди розробників можна вжити різних заходів. Для досвідченої команди визначення типу програми означає, що можна використати досвід розробки подібних програм у минулому. Команди з меншим досвідом розробки також можуть отримати користь від категоризації, отримавши та впровадивши певний набір вказівок і принципів для конкретного типу програми. У [20] автори виділяють дванадцять класів програм мобільної комерції. Приклади класів включають мобільні фінансові програми (банкінг і мікроплатежі), місцезнаходження продукту та покупки (пошук і замовлення товарів), а також мобільні розважальні послуги (відео на вимогу та подібні послуги). Однак ці класи застосовуються лише до мобільних комерційних додатків (мобільних додатків, які передбачають транзакції товарів і послуг) і не допомагають надавати вказівки для розробки нових додатків. Для цієї мети результати роботи [12] виявилися більш корисними. Посилаючись на звіт Рамзі та Нільсена про зручність використання WAP, автори поділяють мобільні програми на дві групи: високоцільові та орієнтовані на розваги. Визначення кожної групи досить просте: цілеспрямовані програми спрямовані на надання швидких відповідей на запити, тоді як програми, орієнтовані на розваги, допомагають користувачам згаяти час. Далі автори надають сім керівних принципів для розробки цілеспрямованих мобільних послуг: мобільність (надають інформацію під час руху), корисність, релевантність (включають лише релевантну інформацію), легкість у

використанні, плавність навігації (найважливішу інформацію має бути найлегше знайти), орієнтованість на користувача (адаптація до способу взаємодії та способу мислення користувачів) та персоналізація (адаптація до потреб користувачів і можливості). Надання вказівок для категорії додатків, орієнтованих на розваги, може бути досить складним і виходить за рамки поточного обговорення.

Таксономія мобільних додатків з точки зору підприємства встановлена в [8] Автори стверджують, що така організація та представлення мобільних додатків зробить вимоги до додатків більш помітними та допоможе розробникам зосередитися на найважливіших аспектах дизайну та реалізації для кожного проекту. Найнижчий рівень таксономії (впорядкований за багатством і складністю додатків) представлений програмами мобільного мовлення (M-broadcast) , які спрямовані на забезпечення широкомасштабної трансляції інформації на мобільні платформи. Програми вищого рівня – це програми мобільної інформації (M-інформація) , які надають інформацію, необхідну мобільним користувачам, наприклад погодні умови. Третій рівень додатків - це мобільні транзакції (M-транзакції), які полегшують електронні транзакції та управління взаємовідносинами з клієнтами. Четвертий рівень, мобільна операція або M-операція, стосується операційних аспектів бізнесу, таких як управління запасами або управління ланцюгом поставок. Нарешті, верхній рівень таксономії представлений мобільною співпрацею (M-collaboration), класом додатків, які підтримують співпрацю всередині та за межами підприємства. Незважаючи на те, що автори аналізують виключно мобільні програми в корпоративному контексті, рекомендації надаються для кожного типу програми; їх можна застосувати в більшості подібних проектів. У додатках M-broadcast контент транслюється великій кількості незареєстрованих користувачів, тоді як у M-information користувачі запитують і отримують інформацію індивідуально. Проблеми, пов’язані з цією категорією програм, включають зручність використання та конфіденційність, але безпека не має особливого значення. Програми M-transaction дозволяють здійснювати мобільні транзакції, такі як

розміщення та відстеження замовлень і здійснення електронних платежів. Ця категорія додатків має вищі вимоги щодо безпеки, швидкості реагування та надійності та потребує зв'язку між трьома сторонами: користувачем, постачальником послуг і фінансовим посередником (наприклад, шлюз онлайн-платежів). Програми M-operation повинні надавати інформацію в реальному часі, а також інтегрувати серверні системи та бази даних. Остання група програм, M-collaboration, пов'язана з проблемами кодування та керування даними через необхідну підтримку взаємодії між різними програмними модулями.

У [18] визначено шість різних категорій мобільних програм: автономні програми (ігри чи утиліти), програмне забезпечення для персональної продуктивності (word). процесори та офісні додатки), Інтернет-додатки (клієнти електронної пошти, браузери), вертикально інтегровані бізнес-додатки (безпека), додатки з визначенням місця розташування (планувальники турів та інтерактивні гіди), а також додатки спеціальної мережі та групового програмного забезпечення (група користувачів встановлює спеціальну мережу для обміну документами). Автори вказують на деякі важливі вимоги до визначених груп мобільних додатків. Для програмного забезпечення для персональної продуктивності синхронізація між мобільною та настільною версіями програмного забезпечення вказана як важлива вимога. Для третьої категорії, Інтернет-додатків, наголошується на питанні продуктивності клієнтських додатків і вимог до ресурсів. Автори стверджують, що мобільний клієнтський додаток не може «позичити» у немобільних клієнтських додатків, оскільки вони мають зовсім інші базові припущення щодо вимог до продуктивності та доступності ресурсів. Ці проблеми також стосуються вертикально інтегрованих бізнес-додатків, оскільки сервери не повинні знати про тип клієнта, з яким вони спілкуються (мобільний чи немобільний), щоб полегшити розгортання мобільних додатків. Описані вище роботи служать основою для створення способу категоризації мобільних додатків та інтеграції завдання категоризації з методологією Mobiie-D . Нова запропонована таксономія представлена на рисунку 3.1

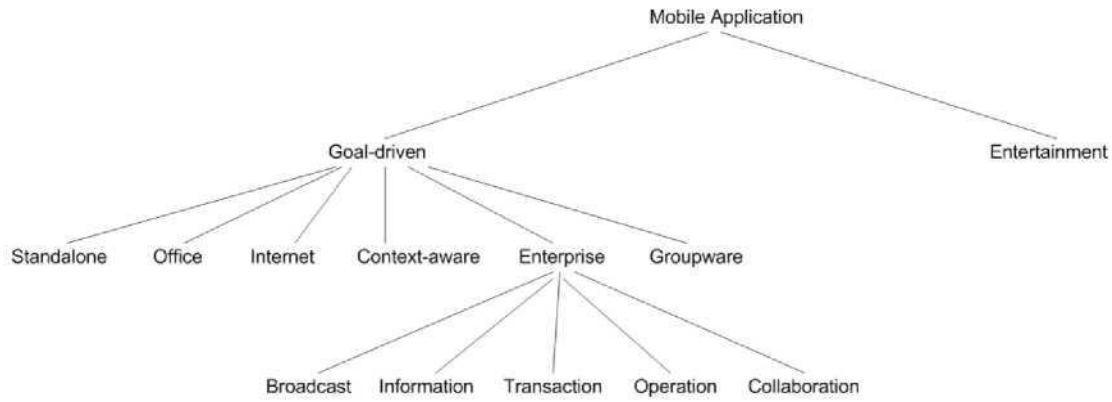

Рисунок 3.1 - Запропоновані категорії мобільних додатків на основі

Категорії не є вичерпними чи виключними. Кожна команда або компанія може розширити категорію відповідно до власного досвіду та минулих проектів. Наприклад, «Розваги» не має підкатегорій через велику різноманітність цього типу програм. Компанія, що спеціалізується на мобільних іграх, може ще більше розширити цю категорію, взявши до уваги різні типи ігор, які вона розробила в минулому. Підсумовуючи, переваги категоризації програм у життєвому циклі подвійні: отримання набору вказівок, налаштованих для конкретного типу програми, та використання попереднього досвіду для розробки нової програми того самого типу. Цей досвід можна використовувати, наприклад, для оцінки зусиль, подібно до використання методу стандартної оцінки компонентів у завданнях оцінки програмного забезпечення. Оптимальне розташування завдання категоризації в рамках методу Mobile-D знаходиться на етапі дослідження , який містить етап визначення обсягу (рис.3.1) . Відповідно до специфікацій Mobile-D [14] етап визначення масштабу визначає цілі та встановлює часові рамки для проекту. Якщо команда визначає категорію програми, яку вони розробляють, вони можуть встановити цілі проекту, які відповідають конкретним рекомендаціям, і можуть сформувати початковий графік відповідно до даних, зібраних із попередніх подібних проектів. Етап визначення обсягу складається з двох основних завдань: початкового планування проекту та збору початкових вимог. Виконуючи початкове завдання планування проекту , команда розробників встановлює часові рамки проекту, ритм розвитку

та оцінює необхідні інвестиції для проекту з точки зору зусиль, фінансів тощо. На результат цього завдання можна позитивно вплинути, якщо йому передує нове завдання, пов’язане зі встановленням категорії проекту, так що оцінки на етапі будуть більш точними. Новий потік завдань на етапі визначення обсягу представлено на рисунку 3.2

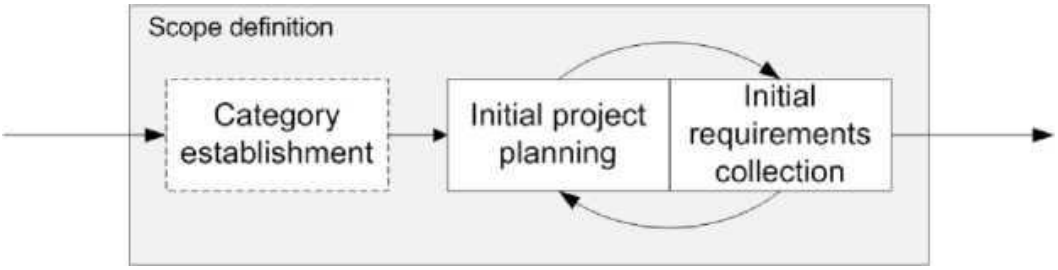

Рисунок 3.2 - Етап визначення обсягу

Коли проект завершено, досвід щодо рекомендованих практик, інструментів і оцінок слід записати в конкретну категорію для подальшого використання.

3.2 Альтернативні підходи до мобільного розвитку

Незважаючи на те, що гнучкі методології пропонують гарне рішення для розробки мобільних додатків, у літературі існують різні підходи. Mobile-D , очевидно, є найдетальнішою методологією для цієї мети, яка має вичерпну специфікацію для кожної фази та етапу, а також для відповідних завдань. Однак інші багатообіцяючі підходи можуть покращити Mobile-D у деяких аспектах, а також підвищити гнучкість методу. Тема, що повторюється у відповідних дослідженнях, - це використання інженерії, керованої моделлю (MDE), точніше, розробки, керованої моделлю (MDD), для розробки мобільних додатків. Відповідно до [20]), розробка, керована моделями, передбачає використання моделей не лише для документування коду, але й для того, щоб служити основою для розробки програм. У [10] автори пропонують підхід до розробки, який поєднує принципи як MDD, так і взаємодії людини з комп’ютером (HCI),

точніше з області дизайну, орієнтованого на користувача. Метою підходу є отримання системи, яка дозволить користувачам-початківцям без досвіду програмування створювати власні мобільні програми. Основним аргументом на користь MDD у розробці мобільних додатків є можливість створити незалежну від платформи модель додатку, яка автоматично трансформуватиметься в платформо-специфічний код. Це важлива перевага для MDD, оскільки кількість платформ для програмного забезпечення велика і постійно збільшується. Дозволяючи кінцевим користувачам створювати власні додатки та роблячи цей процес зручним за допомогою принципів проектування, орієнтованого на користувача, зникне потреба у зовнішніх розробниках, що зменшить витрати та час розробки.

Крім великої кількості платформ, на яких має працювати програмне забезпечення, існує ще одна проблема, пов'язана з мобільними програмами, а саме фактори, які впливають на кожен елемент програми , наприклад обмеження ресурсів. Щоб вирішити обидві ці проблеми, [22] пропонує підхід до розробки, який поєднує методи аспектно-орієнтованої розробки програмного забезпечення (AOSD) із методами розробки, керованої моделлю. Автори говорять про «наскрізні фактори», такі як обмеження пристроїв або підключення користувачів, як вплив, який нелегко впоратися традиційними підходами до програмування, такими як об'єктно-орієнтоване або процедурне програмування, і які впливають на всю програму. Ці фактори призводять до низки проблем. По-перше, розробники програмного забезпечення не можуть легко відокремити семантику програми від технічної специфіки, а по-друге, розробникам доводиться впроваджувати ці наскрізні проблеми в усій програмі. Це, у свою чергу, призводить до зниження зручності обслуговування, розширення та зрозумілості програмного забезпечення. Наскрізні проблеми є основою для аспектів аспектно-орієнтованого проектування, тому, використовуючи цей підхід у розробці, проблеми можуть бути модульними, таким чином підвищуючи зручність обслуговування, розширюваність і зрозумілість коду. У цьому підході MDD використовується, як і раніше, для перетворення високорівневого

незалежного від платформи дизайну в кілька платформно-залежних реалізацій. У модельно-керованій архітектурі (MDA), яка є MDD, стандартизованою групою управління об'єктами (OMG), низка платформо-специфічних моделей (PSM) генерується з платформено-незалежної моделі (PIM) за допомогою перетворень. Загалом, поєднання AOSD і MDA дає переваги від обох підходів, мінімізуючи вплив наскрізних проблем і проблем, пов'язаних із платформою. Важливою частиною MDD є перетворення моделей у код (генерація коду). У розглянутих роботах це питання не було деталізовано; однак підхід до створення коду в розробці мобільних додатків доступний у (Amanquah & Eporwei, 2009).

Незважаючи на те, що тема MDD для мобільних додатків ще значною мірою не вивчена, є деякі перші враження від використання. У (Braun & Eckhaus, 2008) MDD використовується для розробки архітектури, яка підтримує надання мобільної служби як веб-служби, так і мобільної програми. Мета полягала в тому, щоб забезпечити доступ до наданих послуг як через вбудований браузер XHTML, так і через попередньо встановлену програму Java. Результат підходу був успішним, оскільки MDD виявився достатньо гнучким для виконання цього завдання. Інший підхід MDD був задокументований у (Khambati, et al., 2008) для розробки мобільних персональних програм охорони здоров'я. У цьому випадку MDD допомагає постачальникам медичних послуг створювати персоналізовані програми, змодельовані на основі плану медичного обслуговування, спеціального для кожного пацієнта. На жаль, в літературі немає комплексних досліджень впливу MDD у мобільних додатках, а також емпіричних даних у розглянутих статтях. Це залишає MDD як експериментальний підхід, який можна інтегрувати в Mobile-D, лише якщо команда розробників має досвід використання відповідних практик і вважає, що використання MDD може отримати помітні переваги. Інший підхід представлено в (Rahimian & Ramsin, 2008). Автори використовують методологічний інженерний підхід, який називається Hybrid Method Engineering, щоб створити метод, придатний для розробки мобільних додатків. Методологічна інженерія - це дисципліна, яка займається створенням методологій, придатних для різних сценаріїв розвитку,

мотивованих переконанням, що жоден процес не підходить для всіх ситуацій. Дизайн гібридної методології використовує попередньо встановлені ключові вимоги та висновки як вхідні дані, щоб ітеративно генерувати бажану методологію. У даному випадку автори використовують як вхідні дані список ключових рис методології, подібний до тих, що були описані раніше на сторінці 7, а також висновки з відповідної роботи в цій галузі. Кожна ітерація методу містить наступні завдання: встановлення пріоритетів вимог, вибір підходів до проектування, які будуть використовуватися в поточній ітерації, застосування вибраних підходів до проектування, перегляд, уточнення та реструктуризація методології, створеної до цього моменту, визначення рівня абстракції для наступної ітерації та, нарешті, перегляд та уточнення вимог, визначення пріоритетів для наступної ітерації. Запропонована методологія мобільної розробки створена в чотири ітерації, починаючи з загального життєвого циклу розробки програмного забезпечення (аналіз, проектування, впровадження, тестування та перехід). У першій ітерації методологія деталізується шляхом додавання практик, які зазвичай зустрічаються в гнучких методах. Беручи до уваги ринкові міркування, друга ітерація включає дії з розробки нового продукту, процесу, пов'язаного з представленням нового продукту або послуги на ринку. У третій ітерації ідеї адаптивної розробки програмного забезпечення (ASD) були інтегровані в методологію, а в останній ітерації було додано прототипування, щоб зменшити ймовірні ризики, пов'язані з технологією. Остаточні етапи методології, запропоновані авторами, представлені на рисунку 3.3.

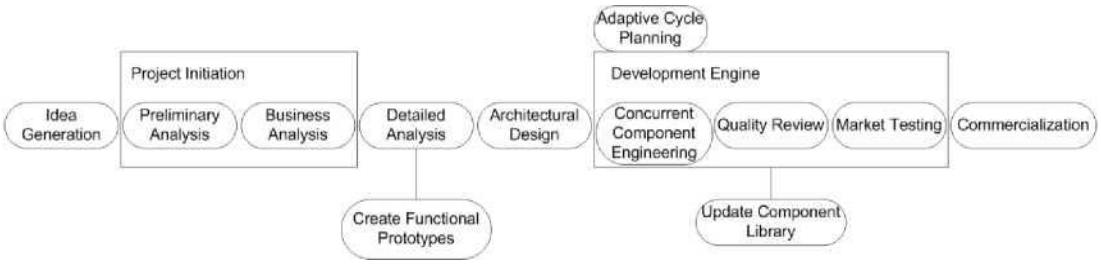

Рисунок 3.3 - Мобільний метод розробки, запропонований у (Rahimian & Ramsin, 2008)

Запропонована структура розробки враховує більшість проблем, виявлених у відповідній роботі на місцях. Проте методологія поки що на високому рівні, і конкретних завдань для визначених етапів не передбачено. За словами авторів, майбутня робота включає виконання подальших ітерацій для отримання завдань нижчого рівня в процесі. У своєму поточному стані методологія надто високого рівня для інтеграції з Mobile-D. У [26] автори описують структуру для розробки корпоративних мобільних додатків. Їхній підхід використовує багатошарову архітектуру, що складається з комунікаційного (мережа з протоколами), інформаційного (база даних), проміжного програмного забезпечення та зв'язування (сервісні структури), прикладного (бізнес-програми, такі як бронювання та платежі) та презентаційного рівня (інтерфейс користувача), з рівнем безпеки, ортогональним п'яти попереднім. Представлений фреймворк був успішно застосований у трьох корпоративних проектах, при цьому більше уваги приділено певним рівням, відповідно до профілю організації. Якщо категоризація використовується в методології Mobile-D , як пояснювалося раніше, і проект, про який йде мова, є корпоративним, структура, представлена в [24] може надати корисну архітектурну модель для команди розробників. Інформацію можна взяти до уваги на етапі дослідження , етапі створення проекту (див. рис. 1) , під час виконання завдання під назвою «Визначення лінії архітектури». Відповідно до специфікації Mobile-D , мета завдання - отримати впевненість в архітектурному підході, щоб проект був успішно реалізований. Точніше, мета завдання – визначити архітектурну лінію для проекту. Подальші відомості про лінію архітектури в Mobile-D доступні в [20].

3.3 Керування ресурсами Android

Цей підрозділ присвячено новому інструменту підтримки для проектів розробки мобільного програмного забезпечення. Як зазначалося раніше, підтримка інструментів для таких дій, як тестування продуктивності та журнал використання програми, є досить поганою, особливо у випадку мобільного

програмного забезпечення. У своїй поточній формі інструмент спрямований на те, щоб допомогти розробникам впоратися з використанням двох основних типів ресурсів: фізичних ресурсів платформи та ресурсів дизайну. У той час як управління фізичними ресурсами здійснюється шляхом тестування та оцінки використання ресурсів, використання проектних ресурсів оптимізується шляхом розробки лише корисних компонентів для певного продукту та економії часу та зусиль, де це можливо, шляхом припинення невикористовуваних функцій. У наступній частині система буде описана з точки зору можливостей і компонентів високого рівня на прикладах; потім буде обговорено вплив інструменту підтримки на методологію Mobiie-D . Основні вимоги до інструменту підтримки наведені нижче: 1. Оцініть продуктивність компонентів у перевіреній системі за низкою показників 2. Підтримка життєвого циклу журналу використання До нефункціональних вимог належать: простота використання, щоб розробники могли використовувати інструмент у своїх інтересах; легкість розповсюдження та інтеграції, що робить інструмент доступним і готовим до застосування; і розширюваність, що дозволяє розробникам додавати функціональність відповідно до своїх конкретних потреб. Щоб відповідати цим вимогам, необхідно вибрати відповідні платформи розробки. Завдяки високій популярності, широкому використанню та потужному API обрано платформу Android (Android, 2010); для задоволення вищевказаних нефункціональних вимог інструмент був розроблений як плагін Eclipse (Eclipse, 2010). Загальний підхід для оцінки ефективності полягає в налаштованих тестових прикладах і твердженнях. Використовуючи їх, розробники можуть затверджувати максимальний рівень для певної метрики продуктивності, маючи результат невдачі, якщо рівень перевищено. Відповідно до (Android, 2010), програма для Android використовує чотири різні типи компонентів: дії , послуги, приймачі трансляції та постачальники контенту. Інструмент підтримки, описаний тут, має справу лише з компонентами діяльності під час оцінки ефективності, як буде пояснено пізніше. На рисунку 3.4 показана діаграма варіантів використання системи.

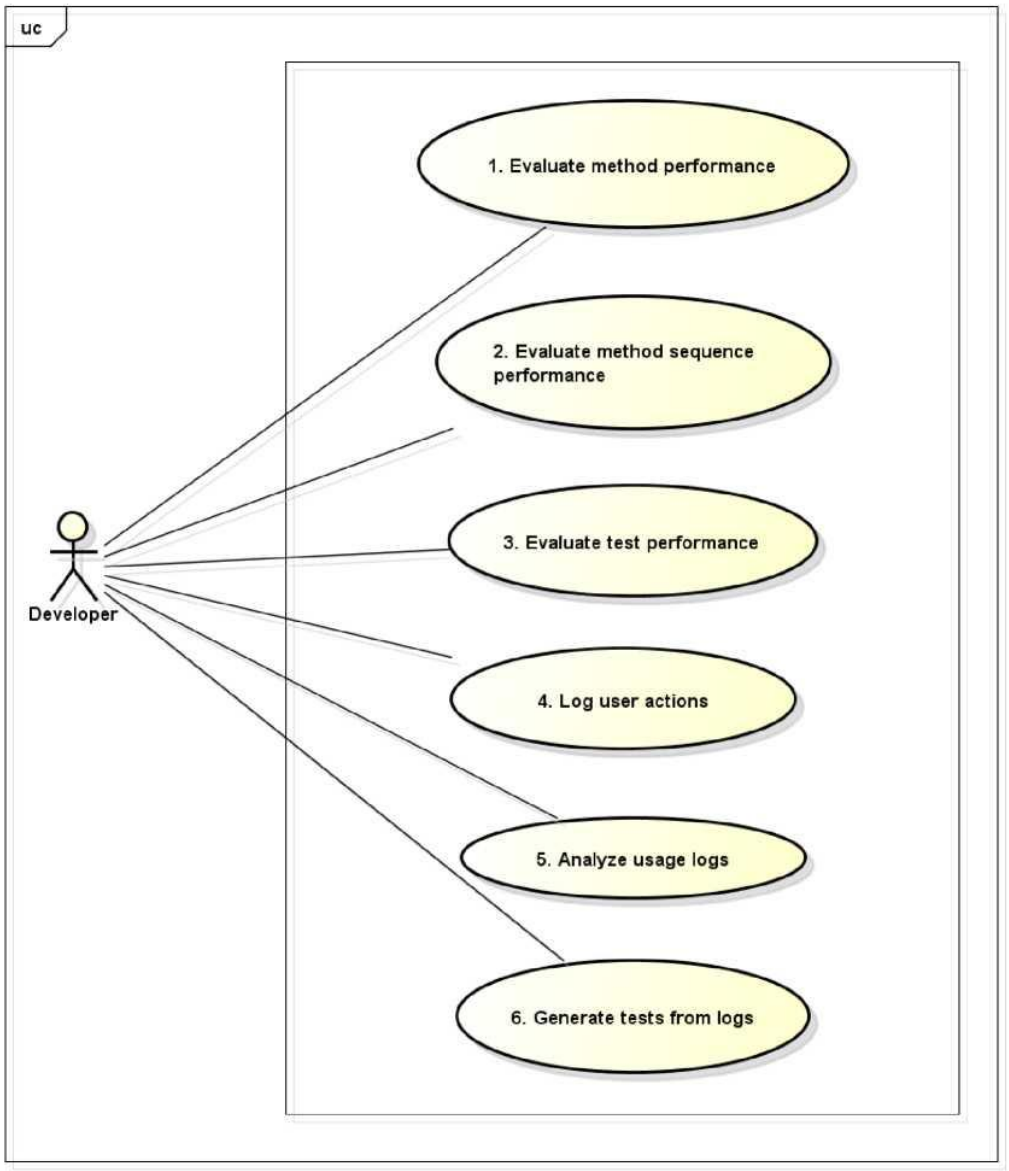

Рисунок 3.4 - Діаграма варіантів використання для системи

Варіанти використання з 1 по 3 стосуються фізичних ресурсів, тоді як варіанти з 4 по 6 зосереджуються на ресурсах розробки. Із чотирьох згаданих раніше компонентів додатків Android для тестування продуктивності розглядалися лише дії (випадки використання 1–3) через кілька причин. По-перше, вони завжди присутні в програмах і містять значну кількість чутливого до продуктивності коду, наприклад, обробки графічних інтерфейсів. Існує можливість розширити інструмент підтримки, щоб включити тестування компонентів служби; однак у багатьох випадках служби надають фонову

функціональність для програми та працюють протягом невизначеного періоду часу, зазвичай приєднані до дії та не мають інтерфейсу користувача. Це означає, що тестування на час може бути нерелевантним для послуг. Нарешті, перевірка продуктивності двох інших типів компонентів (приймачів трансляції та постачальників контенту) є проблемою для подальшого вивчення, оскільки ця функція може бути некорисною для розробників.

Щоб продемонструвати функціональність інструменту підтримки, ми будемо використовувати дві програми, доступні як зразки в дистрибутивах Android SDK, під назвою Spinner і MuItiRes. Перша є простою діяльністю, яка представляє користувачеві меню вибору планет, а MuItiRes дозволяє користувачеві переміщатися по колекції фотографій. Додатки зображені нижче.

Рисунок 3.5 - Програми Spinner (ліворуч) і MuItiRes (праворуч).

Цей розділ інструменту дозволяє розробникам тестувати продуктивність методів у Activity щодо часу виконання (у мілісекундах), розміру розподілу пам’яті (у байтах) і кількості виділень пам’яті. Методи, які використовуються для цього, надаються в новому класі, який розширює широко використовуваний ActivityInstrumentationTestCase2 , наданий у Android SDK. Щоб створити тестові випадки продуктивності, розробники можуть використовувати майстри, надані плагіном.

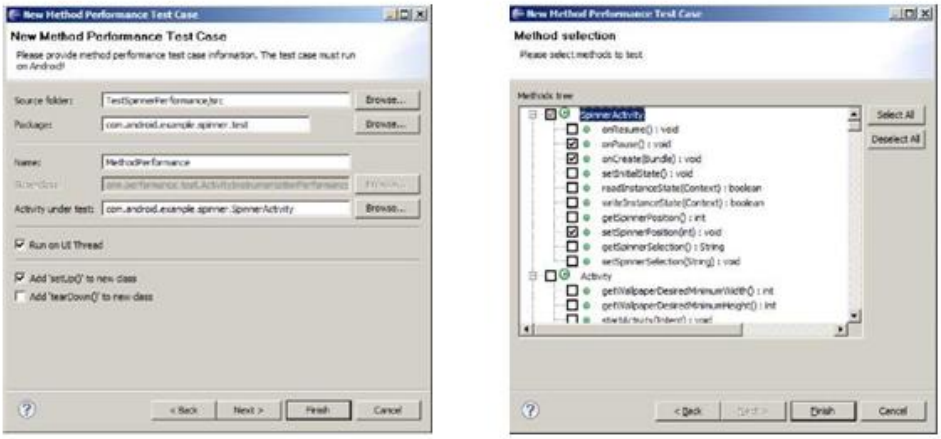

Рисунок 3.6 - Майстер, який використовується для створення прикладу тестування продуктивності для методів

Як показано на рисунку 3.6, майстер допомагає створити тестовий приклад для продуктивності методу. Користувач повинен створити цей новий тестовий приклад всередині тестового проекту Android, який є спеціальним проектом Eclipse, який використовується для тестування програм Android, і запустити щойно згенерований тестовий приклад як Android Test Case . Таким чином, тестовий приклад встановлюється на підключеному емуляторі або пристрої, тести запускаються, а результати надсилаються в Eclipse. Користувач повинен надати конкретну інформацію на першій сторінці, таку як тестований клас Activity, чи потрібні заглушки setUp або tearDown або чи потрібно запускати тести в потоці інтерфейсу користувача активності. Останній параметр фактично запитує користувача, чи впливають методи, що перевіряються на продуктивність, безпосередньо на основні екрани інтерфейсу користувача. Через специфіку дій на основну (UI) нитку діяльності неможливо вплинути ззовні потоки (наприклад, методи виклику тестового потоку), тому для обходу цієї вимоги потрібно використовувати анотацію. Хорошим прикладом методу, який виконується в потоці інтерфейсу користувача активності, є onCreate, метод, який викликається кожного разу, коли активність запускається, і реалізується всіма діями. Друга сторінка майстра дозволяє користувачеві вибрати методи, які вони бажають оцінити. Інтерфейс майстра натхненний майстрами JUnit для кращого

знайомства. На рисунку 3.7 показано найважливіші розділи коду, згенерованого майстром.

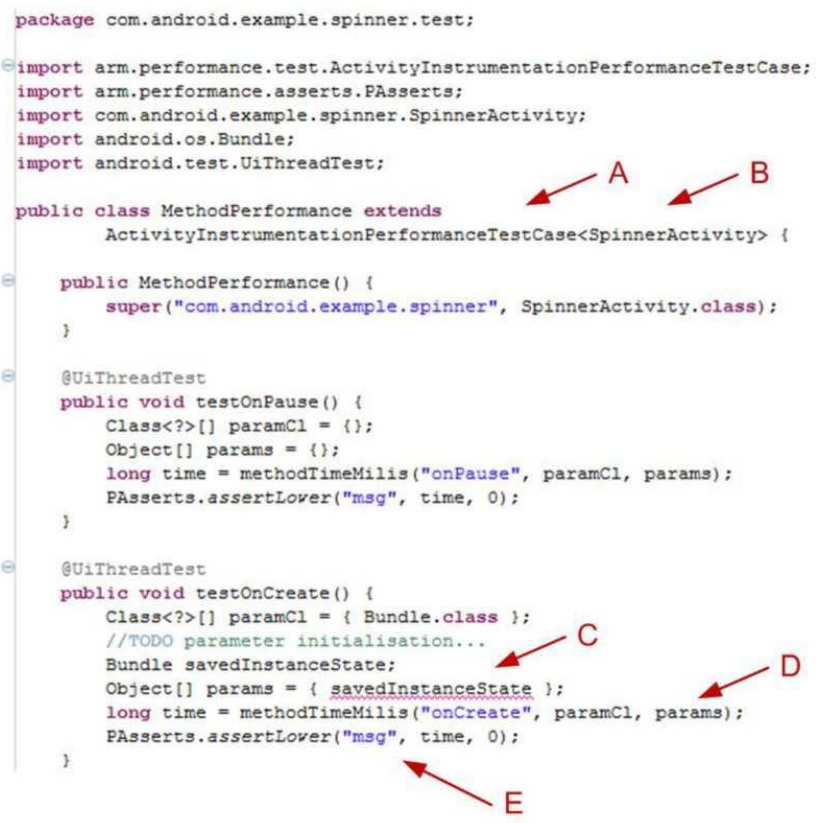

Рисунок 3.7 - Код, згенерований майстром продуктивності методу

На рисунку 3.7 показано згенерований код. У пункті А поточний тестовий приклад оголошено для розширення нового класу тестового тесту продуктивності з інструменту підтримки. Генерики використовуються для визначення активності, що тестується; тут це SpinnerActivity (точка В). У точці С, згенерована тестова заглушка залишає об'єкти параметрів неініціалізованими, залишаючи розробникам вирішувати значення параметрів. У точці D викликається метод methodTimeMHHs, щоб отримати час виконання в мілісекундах тестованого методу для заданих параметрів, і, нарешті, у точці Е робиться твердження щодо часу виконання; твердження генерується протягом 0 мс за замовчуванням, щоб дозволити розробникам вибрати відповідний час. Розробники можуть вибрати інші методи, які підходять для тестування інших характеристик продуктивності, а також інших тверджень, які надаються

бібліотеками засобів підтримки. У таблиці 3.1 перераховані всі методи, доступні розробнику.

У деяких випадках, особливо коли використовується тестова розробка, розробники мають чітке уявлення про залежності між різними компонентами. Ця функціональність вирішує проблему тестування продуктивності послідовності методів, перевіряючи, чи сумарний час виконання цих методів знаходиться в межах заданого ліміту, і спостерігаючи за компромісом у використанні ресурсів між залежними методами. Це робиться так само, як і в попередньому варіанті використання, але вибрані методи викликаються послідовно та під час кожного виклику обчислюється час, доступний цьому методу. За замовчуванням, коли таймер закінчився, виконання зупиняється та повертає помилку. Цей варіант використання має пов’язаний майстер, подібний до представленого на рисунку 3.6. На рисунку 3.8 показано згенерований код, необхідний для проведення оцінки ефективності послідовності методів.

```
@UiThreadTest
public void testMethodQueue() {
    MethodQueue queue = new MethodQueue();

    String name0 = "onCreate";
    Class<?>[] paramCl0 = { Bundle.class };
    //TODO parameter initialisation...
    Bundle savedInstanceState;
    Object[] params0 = { savedInstanceState };
    MethodTrace mt0 = new MethodTrace(name0, paramCl0, params0);

    String name1 = "setSpinnerPosition";
    Class<?>[] paramCl1 = { int.class };
    //TODO parameter initialisation...
    int pos;
    Object[] params1 = { pos };
    MethodTrace mt1 = new MethodTrace(name1, paramCl1, params1);

    //TODO check method order...
    queue.addToQueue(mt0);
    queue.addToQueue(mt1);

    assertTrue(this.checkAggregateTime(queue, 0));
}
```

A
B
C
D

Рисунок 3.8. - Код, згенерований майстром продуктивності послідовності методів

Наведений вище малюнок зосереджений на одному тесті, створеному майстром. Решта згенерованого класу така ж, як і в попередньому випадку (рисунок 3.7) . MethodQueue (точка A) генерується та заповнюється MethodTraces

(точка B) у порядку їх виклику (точка C). Як і раніше, параметри залишаються неініціалізованими. Нарешті, у пункті D викликається метод checkAggregateTime , щоб перевірити, чи буде викликана черга методів до закінчення часу. Генерація коду встановлює загальний цільовий час на 0, який розробники змінюють до відповідного значення перед запуском тесту. Результатом тесту є «склав/не склав», а також записи в Android Logger. На малюнку 19 показано результати, зареєстровані тестом, для черги з 4 методів із заданим цільовим часом виконання 300 мс.

Логування використання програми може принести важливі переваги розробникам. Вони включають отримання кращого розуміння того, які компоненти активно використовуються, а які ні, і прийняття майбутніх проектних рішень на основі цих висновків, а також отримання даних про помилки та збої, з якими стикаються користувачі. Щоб допомогти з цими завданнями, представлений тут інструмент підтримки пропонує рішення для охоплення найважливіших аспектів життєвого циклу журналу використання, представленого на рисунку 3.9. Зокрема, інструмент підтримки пропонує реєстратор, приєднаний до класу, здатний реєструвати такі події, як виклики методів, виклики конструкторів і помилки. Коли реєстратор запускається, він створює порожній файл журналу у власному каталозі програми, який знаходиться на зовнішньому носії пам’яті пристрою Android; наступні події реєструються у файлі з такою інформацією, як час виникнення, назва події (ім’я методу або конструктора) і навіть параметри виклику, якщо подія є помилкою. Використання цих параметрів буде пояснено в наступному розділі, присвяченому створенню тестових випадків.

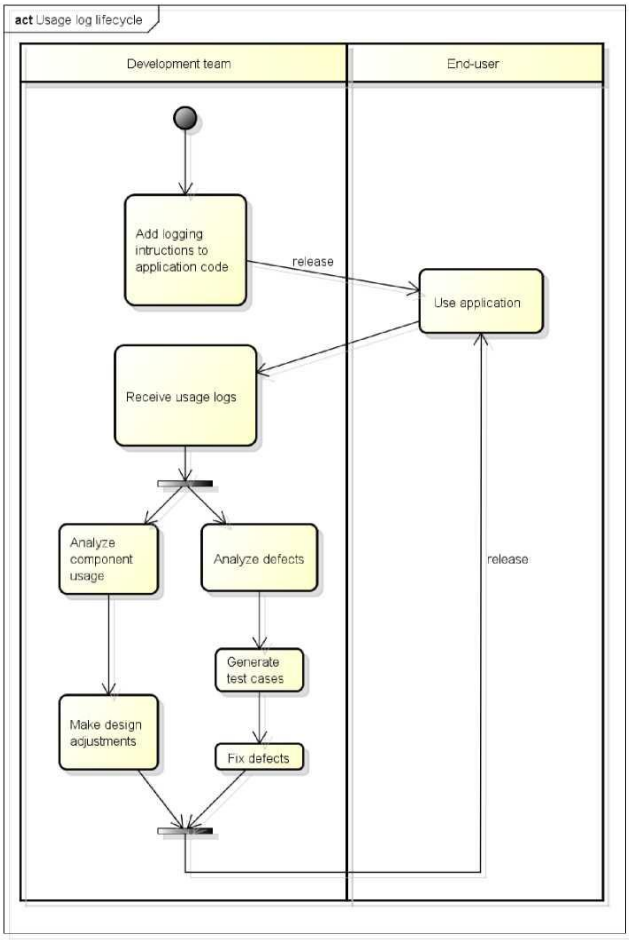

Рисунок 3.9 - Життєвий цикл журналу використання

Реєстратор запускається з параметром, що вказує максимальну кількість «запусків» перед тим, як файл журналу, пов’язаний із поточним класом, можна буде надіслати назад розробникам. «Запуск» у цьому випадку означає кількість запуску реєстратора; якщо реєстратор запускається в логічній точці входу програми, як-от метод onCreate для дій Android, кількість запусків реєстратора відповідає кількості разів додаток було запущено. Було створено просту процедуру завантаження, щоб імітувати надсилання файлів журналу розробникам. Реалізована процедура копіює відповідні файли з каталогу програми в корінь зовнішнього носія; однак надано інтерфейс n (під назвою UploadProcedure) , який розробники можуть реалізувати для завантаження даних про використання в бажаний спосіб. Щоб зменшити робоче навантаження, необхідне для додавання інструкцій журналювання до коду програми, реєстратор використовується з якомога меншим кодуванням. Для решти

варіантів використання наведено приклади з використанням другої програми, представленої раніше (MultiRes). Ми запустимо реєстратор і використаємо його для запису подій. На рисунку 3.10 показано частини основного коду програми з доданими рядками для цілей реєстрації.

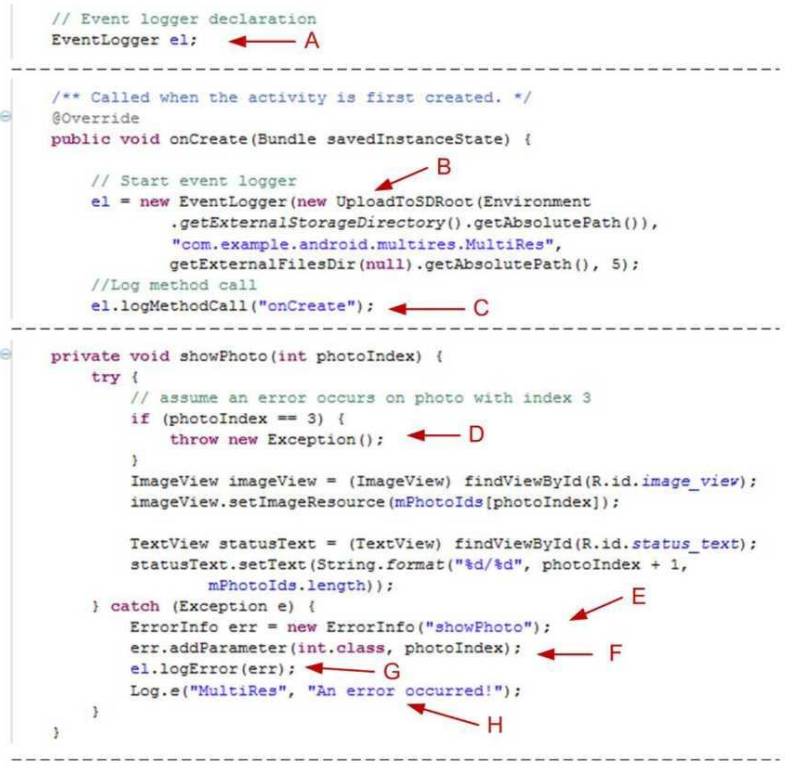

Рисунок 3.10 - Інструкції з реєстрації

У пункті A новий EventLogger створюється як змінна класу. Потім він запускається всередині методу onCreate (точка B). Параметри представляють: процедуру завантаження, повну назву класу, поточний каталог даних програми та максимальну кількість запусків перед спробою завантаження. Усі зовнішні шляхи зберігання надаються класами середовища та контексту Android для забезпечення сумісності з налаштуваннями кінцевого користувача. У точці C реєструється поточний виклик методу. Щоб продемонструвати реєстрацію помилок, ми припустимо, що виникає виняток, якщо метод showPhoto викликається з параметром 3 (точка D). Виняток обробляється за допомогою даних журналу. У пункті E оголошується ErrorInfo з назвою методу, який її спричинив. Потім у точці F клас параметра та екземпляр додаються до об'єкта

інформації про помилку. У точці G помилка реєструється за допомогою об'єкта ErrorInfo ; нарешті, в пункті H помилка також записана в журналі Android. Параметри методу реєструються, щоб спробувати відтворити ці помилки, коли журнали потрапляють до розробників.

```
*header
full_class_name: com.example.android.multires.MultiRes
upload_procedure: SD_ROOT
last_started: 2010/08/17_14:59:30
current_run: 5
max_runs: 5
*events
method_call onCreate 2010/08/17_14:58:26
method_call onCreate 2010/08/17_14:59:06
method_call onCreate 2010/08/17_14:59:14
method_call onCreate 2010/08/17_14:59:19
method_call onCreate 2010/08/17_14:59:30
*errors
showPhoto 2010/08/17_14:59:21 params-> int:Integer3690,
showPhoto 2010/08/17_14:59:33 params-> int:Integer9440,
showPhoto 2010/08/17_14:59:40 params-> int:Integer7732,
```

Рисунок 3.11 - Журнал, згенерований для 5 запусків програми MultiRes

На рисунку 3.12 показано файл журналу, створений під час запуску програми. Метод onCreate викликався 5 разів, що відповідає кількості запуску програми. У розділі помилок показано, що користувач зіткнувся з 3 помилками. Як показано на рисунку 3.12, це означає, що користувач досяг фотографії з індексом 3 і було створено виняток. Помилки реєструються з назвою методу, міткою часу та параметрами (тип і посилання на фактичний об’єкт, серіалізовані та збережені на носії). Під час завантаження журналів використання ці об’єктні файли також надсилаються для створення тестових випадків.

Останньою функціональністю інструменту підтримки, описаного тут, є генерація тестів, метою якої є відтворення помилки, з якою зіткнувся кінцевий користувач. Щоб досягти цього, інструмент підтримки бере журнал використання та пов’язані серіалізовані об’єкти та використовує Java Reflection API, щоб спробувати перевести тестову програму до тієї самої точки виконання, яка була зареєстрована як помилка. Розглядаючи життєвий цикл журналювання використання, показаний на рисунку 3.9, можна побачити, що більшість дій

покриваються інструментом підтримки. Ті, які не охоплюються, або надто високорівневі, щоб бути автоматизованими (виправлення дефектів, виявлених тестовими прикладами, перепроектування компонентів), або специфічні для організації (процедура завантаження журналів і пов'язаних файлів). У цьому розділі описано використання журналів і серіалізованих об'єктів, завантажених групі розробників із пристроїв кінцевих користувачів.

Як показано на рисунку 3.10, коли реєструється помилка, реєстратор вимагає параметрів методу, у якому сталася помилка. Потім ці параметри серіалізуються та зберігаються на зовнішньому носії пристрою, щоб потім надсилатися разом із файлом журналу, коли буде досягнуто максимальної кількості запусків. Коли розробники отримують журнали та серіалізовані об'єктні файли, вони можуть створити тестові приклади, які мають на меті отримати таку ж поведінку, а потім перейти до її виправлення.

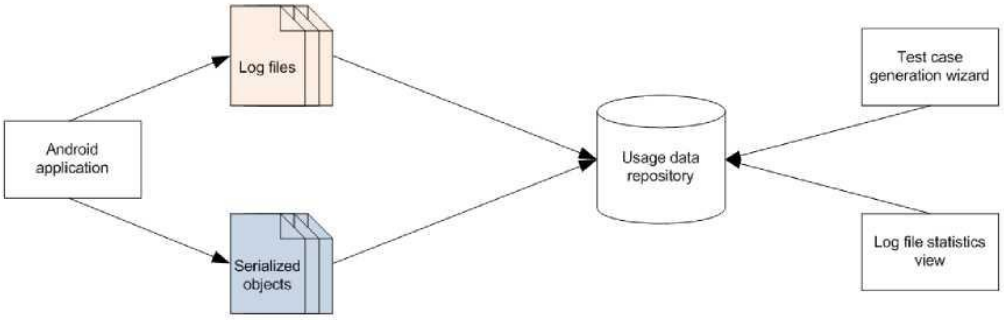

Рисунок 3.12 - Журнали та серіалізовані об'єкти

Підхід здатний створити тестовий приклад, який переводить програму в ту саму точку виконання, що описана в журналі, коли сама помилка викликана параметром у виклику методу. Звичайно, помилка може надходити з інших джерел, таких як змінні класу або зовнішній вхід. Однак, використовуючи цей підхід, якщо згенерований тестовий приклад не може призвести до того, що програма досягне зареєстрованої точки виконання, причину можна звузити до інших факторів. Крім того, реєстратор може використовувати лише серіалізовані об'єкти для реєстрації як параметри; на щастя, більшість класів Java можна

серіалізувати. Недоліком є те, що значна кількість класів Android такими не є, цю функціональність замінено інтерфейсом Android Parcelable . Якщо параметри не можна серіалізувати, розробники мають можливість зареєструвати помилку без параметрів; це все одно надасть деяку інформацію про виникнення помилок, навіть якщо генерація тестів більше не буде можливою. Останній варіант, доступний розробникам для того, щоб отримати уявлення про помилку, яка сталася, - це зареєструвати виняток замість параметрів, використовуючи той самий механізм журналювання. Оскільки всі винятки можна серіалізувати та містять трасування стека, замість журналювання та серіалізації об'єктів параметрів об'єкт винятку можна серіалізувати та надіслати розробникам для аналізу пізніше.

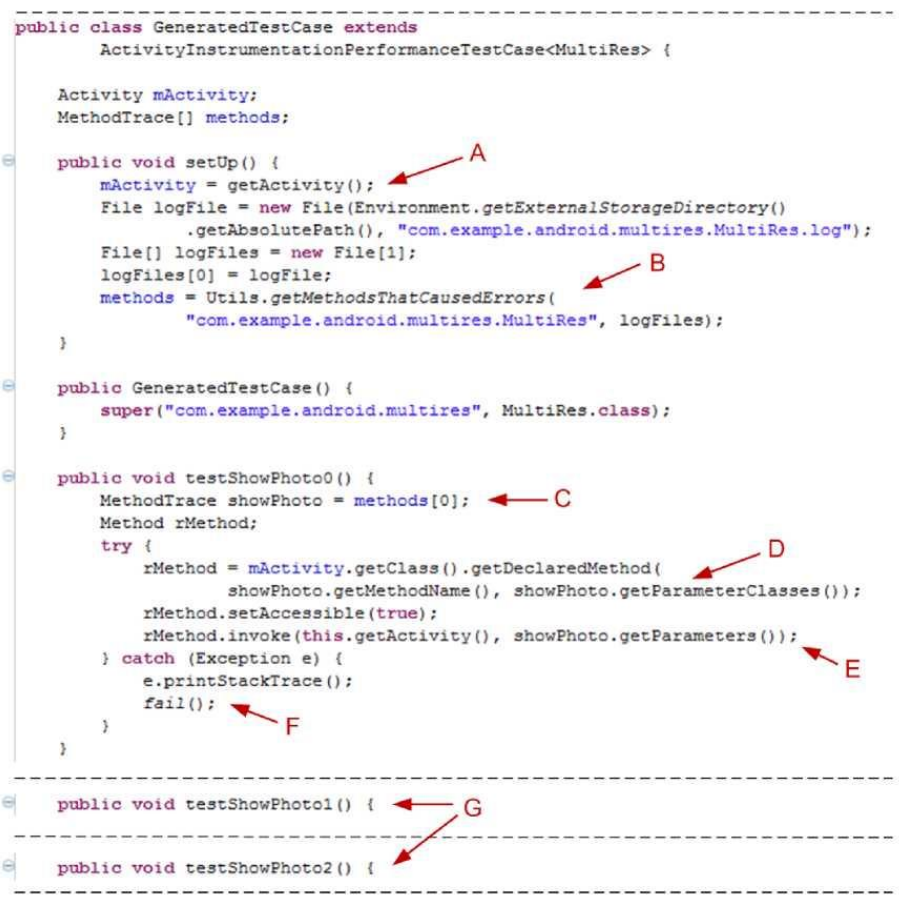

Рисунок 3.13 - Код, створений майстром генерації тестів

Щоб успішно запустити згенерований тестовий приклад на пристрої або емуляторі Android, необхідні файли мають бути завантажені в кореневий каталог

зовнішнього носія пам’яті, ця дія полегшується майстром. Коли файл журналу завантажується, користувачі можуть вказати назву зовнішнього носія, який використовує їхній поточний пристрій, і майстер завантажить файл журналу та всі необхідні серіалізовані об’єктні файли в кореневий каталог цього носія. У точці A файл журналу завантажується з кореневого каталогу. Потім у пункті B отримують трасування методів для тих методів, які викликали помилки. Цей крок також включає завантаження необхідних серіалізованих об’єктів. Потім для кожної помилки в завантаженому файлі журналу створюється тестовий метод (точка G) і приписується трасування методу (точка C). Використовуючи Java Reflection API, метод, який викликав відповідну помилку, відображається та викликається за допомогою отриманої траси методу (точки D і E). Тестовий приклад не виконується, якщо відображення або виклик методу не вдається або якщо метод генерує виняток під час виклику, і проходить, якщо метод був успішно викликаний із отриманими параметрами. Після успішного досягнення точки виконання, яка була зареєстрована як помилка, розробники мають проаналізувати причини такої поведінки. Якщо тест пройдено, а зареєстрована точка виконання не досягнута, це означає, що параметри методу не були причиною поведінки, і необхідно дослідити інші джерела. На рисунку 3.15 показана інтерпретація результатів тесту.

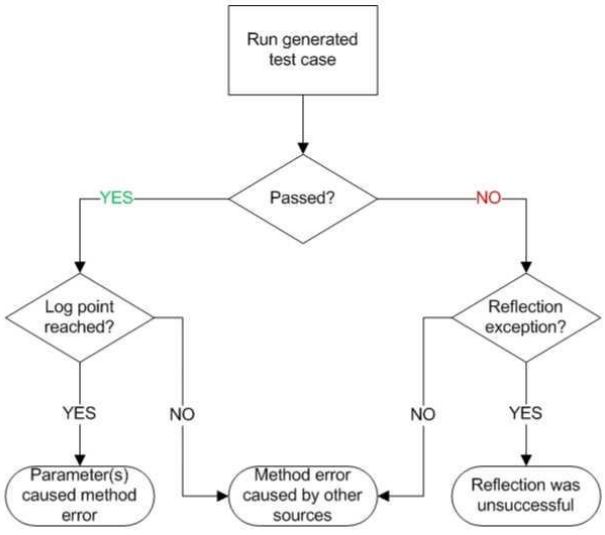

Рисунок 3.14 - Інтерпретація результатів тесту для згенерованих тестів

Інструмент підтримки поширюється у вигляді трьох JAR-файлів, доступних для завантаження з веб-сайту проекту. Одним з них є плагін Eclipse; другий містить необхідні бібліотеки, тоді як третій містить лише бібліотеку реєстратора, яка розповсюджується разом із програмами Android. Завдання з тестування фізичних ресурсів доступні для класів діяльності (для методів і послідовностей методів); заплановані тести можна виконувати на будь-якому типі тестів Android (включно з тестами постачальників послуг і контенту); завдання журналювання можна виконувати на будь-якому типі класу. Фактично, реєстратор, аналізатор і генератор тестів можна використовувати для будь-якої програми Java після незначних змін у реєстраторі. Інструмент підтримки розроблено для Eclipse 3.5 (Galileo), Android 1.6, і вимагає бібліотек JUnit , а також плагіна Android Development Tools (ADT) для Eclipse. Інструмент підтримки також було протестовано в програмі Android, яка була частиною магістерського проекту студента. Поведінка була очікуваною; вибрані методи перевірялися на продуктивність, тоді як пристрій Android використовувався для створення журналів, які потім знімалися з пристрою та аналізувалися. Тестові приклади успішно створено з цих журналів.

Поточний проект намагався вдосконалити найкращі практики мобільної розробки, розглянувши популярну та зрілу методологію розробки, Mobile-D, і вдосконаливши її за допомогою ідей, заснованих на відповідній літературі. Це було зроблено після аналізу надійності фундаментальних принципів методології та того, як ці принципи відповідають проектам мобільної розробки. Останнім основним підходом до вдосконалення стала розробка інструменту підтримки для мобільних розробників, який намагається заповнити прогалину в поточній підтримці інструментів, а також підтримувати запропоновані вдосконалення методології.

3.4 Висновки по розділу

В даному розділі розглянуто Android Resource Management , інструмент підтримки, розроблений як плагін Eclipse. Основними функціями інструменту є тестування продуктивності методів активності Android, визначення часу тестів Android і підтримка життєвого циклу журналювання використання. Інструмент підтримки виконує дві цілі: заповнює прогалину в доступних на даний момент інструментах підтримки для тестування продуктивності та журналювання, а також забезпечує серйозні вдосконалення Mobile-D, такі як інтеграція відгуків кінцевих користувачів після випуску і виконання тестів продуктивності Функціональні можливості інструменту підтримки включають: тестування продуктивності методу, тестування продуктивності послідовності методів, тестування продуктивності сегмента коду, журнал використання, аналіз журналу використання та автоматичне створення тестових прикладів із журналів використання.

ВИСНОВКИ

Дослідження, проведене в рамках цієї роботи, підкреслює складність і багатогранність викликів, з якими стикаються розробники мобільних додатків, а також важливість системного підходу до їх подолання. Аналіз реальних викликів у розробці мобільних додатків показав, що дизайн додатків відіграє ключову роль у забезпеченні зручності користування та залучення користувачів. Дослідження дизайну, розглянуте в першому розділі, виявило необхідність балансування між естетичними аспектами та функціональністю, враховуючи обмеження мобільних пристроїв. Загальні виклики, такі як оптимізація продуктивності, управління ресурсами та забезпечення сумісності з різними пристроями, залишаються актуальними для розробників. Розробка для кількох платформ, зокрема iOS та Android, вимагає використання кросплатформних інструментів або адаптації під специфічні екосистеми, що ускладнює процес розробки, але водночас відкриває можливості для ширшої аудиторії.

Другий розділ, присвячений структурі оцінювання для CPDT (Cross-Platform Development Tools), продемонстрував, що чітко визначений каркас оцінювання дозволяє систематично аналізувати ефективність інструментів розробки. Огляд Mobile-D показав, що сучасні інструменти, такі як Flutter чи React Native, забезпечують швидке створення додатків із високим рівнем продуктивності, але потребують ретельного налаштування для специфічних платформ. Підходи до вдосконалення, такі як модульна архітектура та автоматизоване тестування, сприяють підвищенню якості коду та спрощенню підтримки додатків, що є критично важливим у довгостроковій перспективі.

Третій розділ, зосереджений на підходах до мобільної розробки та особливостях Android, підкреслив різноманітність категорій мобільних додатків (нативні, гібридні, веб-додатки) та їх вплив на вибір технологічного стеку. Альтернативні підходи, такі як прогресивні веб-додатки (PWA) чи кросплатформна розробка, пропонують гнучкість, але потребують додаткових зусиль для оптимізації продуктивності. Керування ресурсами Android, зокрема пам’яттю та енергоспоживанням, виявилося ключовим аспектом для

забезпечення стабільної роботи додатків на широкому спектрі пристроїв, що підтверджує необхідність глибокого розуміння платформних особливостей.

Узагальнюючи, розробка мобільних додатків вимагає комплексного підходу, який поєднує аналіз викликів, вибір відповідних інструментів і врахування платформних особливостей. Результати дослідження можуть бути використані розробниками для створення ефективних і зручних мобільних додатків, які відповідають сучасним вимогам користувачів. Перспективи подальших досліджень включають удосконалення кросплатформних фреймворків для підвищення їх продуктивності, інтеграцію штучного інтелекту для автоматизації дизайну та тестування, а також розробку нових методів оптимізації ресурсів для енергоефективних додатків. Ці напрямки сприятимуть створенню мобільних рішень, здатних відповідати викликам майбутнього цифрового середовища.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Wasserman A. I. Software Engineering Issues for Mobile Application Development. Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research, 2010. С. 397–400. Режим доступу: https://doi.org/10.1145/1882362.1882443

2. Abrahamsson P., Hanhineva A., Hulkko H., et al. Mobile-D: An Agile Approach for Mobile Application Development. Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications, 2004. С. 174–175. Режим доступу: https://doi.org/10.1145/1028664.1028746

3. Flora H. K., Chande S. V. A Review and Analysis on Mobile Application Development Processes using Agile Methodologies. International Journal of Research in Computer Science, 2013. Vol. 3, No. 4. С. 9–18. Режим доступу: https://doi.org/10.7815/ijorcs.34.2013.068

4. Snell J., Powers B. Flutter in Action: Building Cross-Platform Mobile Apps. Manning Publications, 2020. 368 с. Режим доступу: https://www.manning.com/books/flutter-in-action

5. Masiello E., Friedmann J. Mastering React Native: Building Professional Mobile Apps. Packt Publishing, 2017. 496 с. Режим доступу: https://www.packtpub.com/product/mastering-react-native/9781785885785

6. Hermes D. Xamarin Mobile Application Development: Cross-Platform C# and Xamarin.Forms Fundamentals. Apress, 2015. 432 с. Режим доступу: https://www.apress.com/gp/book/9781484202159

7. Griffith A. Apache Cordova in Action: Developing Cross-Platform Mobile Apps. Manning Publications, 2015. 264 с. Режим доступу: https://www.manning.com/books/apache-cordova-in-action

8. Corral L., Sillitti A., Succi G. Mobile Multiplatform Development: An Experiment for Performance Analysis. Procedia Computer Science, 2012. Vol. 10. С. 736–743. Режим доступу: https://doi.org/10.1016/j.procs.2012.06.094

9. Dehlinger J., Dixon J. Mobile Application Software Engineering: Challenges and Research Directions. Workshop on Mobile Software Engineering, 2011. С. 29–32. Режим доступу: https://doi.org/10.1145/2021004.2021017

10. Holzer A., Ondrus J. Mobile Application Market: A Developer's Perspective. Telematics and Informatics, 2011. Vol. 28, No. 1. С. 22–31 Режим доступу: https://doi.org/10.1016/j.tele.2010.05.006

11. Zammetti F. Practical Flutter: Improve Your Mobile Development with Google's Latest Open-Source SDK. Apress, 2019. 384 с. Режим доступу: https://www.apress.com/gp/book/9781484249710

12. Horton J. Ionic Framework By Example: Build Cross-Platform Mobile Apps with Ionic. Packt Publishing, 2016. 234 с. Режим доступу: https://www.packtpub.com/product/ionic-framework-by-example/9781785282720

13. Freeman A. Pro .NET MAUI: Cross-Platform App Development for iOS, Android, macOS, and Windows. Apress, 2023. 400 с. Режим доступу: https://www.apress.com/gp/book/9781484295809

14. Joorabchi M. E., Mesbah A., Kruchten P. Real Challenges in Mobile App Development. 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2013. С. 15–24. Режим доступу: https://doi.org/10.1109/ESEM.2013.9

15. Nagappan M., Shihab E. Future Trends in Software Engineering Research for Mobile Apps. - 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), 2016. С. 21–32. - Режим доступу: https://doi.org/10.1109/SANER.2016.37

16. Google Developers. Flutter: Build Apps for Any Screen. - 2023. - Режим доступу: https://flutter.dev/docs

17. React Native Documentation. React Native: A Framework for Building Native Apps Using React. - 2023. - Режим доступу: https://reactnative.dev/docs/getting-started

18. Microsoft Docs. .NET MAUI Documentation: Cross-Platform App

Development. - 2023. - Режим доступу: https://docs.microsoft.com/en-us/dotnet/maui/

19. Ionic Framework. Ionic Docs: Cross-Platform Mobile App Development. - 2023. - Режим доступу: https://ionicframework.com/docs

20. Dalmasso I., Datta S. K., Bonnet C., Nikaein N. Survey, Comparison and Evaluation of Cross Platform Mobile Application Development Tools. - 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC), 2013. - С. 323–328. - Режим доступу: https://doi.org/10.1109/IWCMC.2013.6583580

21. Designtalk. Storyboard VR: Rapid Prototyping for Virtual Reality. designtalk.club, 2016. - Режим доступу: https://designtalk.club/storyboard-vr

22. DOU. Virtual Reality in Education: Tools and Trends. dou.ua, 2023. - Режим доступу: https://dou.ua/lenta/articles/vr-in-education

23. Epic Games. Unreal Engine 5: VR Development Guide. unrealengine.com, 2025. - Режим доступу: https://www.unrealengine.com/en-US/developer-resources/vr.

24. Unity Technologies. Unity Documentation: Machine Learning and AI Integration. - 2025. - Режим доступу: https://docs.unity3d.com/Packages/com.unity.ml-agents@latest

25. Greengard, S. Virtual Reality. - Cambridge, MA: MIT Press, 2019. - 264 с. - Режим доступу: https://mitpress.mit.edu/books/virtual-reality

26. Heim, M. Virtual Realism. - Oxford, UK: Oxford University Press, 2000. - 264 с. - Режим доступу: https://doi.org/10.1093/acprof:oso/9780195138740.001.0001

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: "Порівняльний аналіз методологій розробки мобільних додатків"

Обсяг пояснювальної записки: 54 аркушів

Дата закінчення дипломної роботи 10 червня 2024р.

Підпис студента _____