https://doi.org/10.31891/2219-9365-2025-81-39
УДК 004.415.2

САВЧЕНКО Ярослав
Приватний вищий навчальний заклад «Європейський університет»
https://orcid.org/0009-0008-0381-0224
ЛЕВЧЕНКО Сергій
Приватний вищий навчальний заклад «Європейський університет»
https://orcid.org/0009-0006-8497-0515
ЯРОВИЙ Роман
Приватний вищий навчальний заклад «Європейський університет»
https://orcid.org/0000-0001-8978-8137

ВИКЛИКИ ТА ПРОБЛЕМИ У ТЕСТУВАННІ МОБІЛЬНИХ ДОДАТКІВ

У статті розглядаються ключові виклики та проблеми, з якими стикаються команди розробки під час тестування мобільних додатків. Особливу увагу приділено фрагментації пристроїв, операційних систем, складності автоматизації тестування, проблемам з UX/UI, обмеженням продуктивності, безпеки та підтримки на різних версіях ОС. Представлено комплексне дослідження, що охоплює аналіз використання різних фреймворків тестування, вплив мережевих і апаратних факторів на стабільність, стрес-тестування, аналіз логів, розбір архітектурних рішень, а також використання AI для пріоритезації тестів. Стаття містить порівняльні таблиці, графіки, приклади коду і результати інтеграції тестів у CI/CD пайплайни. Надані висновки та рекомендації можуть бути корисними як для розробників, так і для тестувальників мобільного ПЗ.

Ключові слова: мобільні додатки, тестування, автоматизація, UX, фрагментація, продуктивність, безпека, CI/CD, telemetry, AI/ML.

SAVCHENKO Yaroslav, LEVCHENKO Serhii, YAROVYI Roman
Private Higher Educational Institution "European University"

CHALLENGES AND ISSUES IN MOBILE APPLICATION TESTING

The article delves into the multifaceted challenges and practical issues that modern development teams routinely encounter during the testing of mobile applications. With the exponential growth of mobile technologies and the demand for high-quality apps, ensuring robust and effective testing strategies has become more critical than ever. This paper examines a wide range of obstacles that hinder mobile app testing, starting with the fragmentation of devices and operating systems—a factor that significantly complicates test planning and execution. Each OS version and hardware configuration can introduce unexpected behaviors, making test coverage and compatibility assurance a daunting task.

Special emphasis is placed on the intricacies of test automation, which, despite being essential for agile workflows and continuous delivery, often falls short due to the limitations of current tools and the dynamic nature of mobile environments. The article also explores challenges related to UX/UI testing, highlighting the importance of usability and visual consistency across diverse platforms.

The performance of mobile applications is analyzed from multiple angles, including memory usage, processing power limitations, and responsiveness under varying network conditions. Security remains a central concern, with focus on safeguarding data, ensuring secure authentication, and preventing unauthorized access.

In addition, the article presents a thorough review of testing frameworks such as Appium, Espresso, XCUITest, and integration tools suitable for Flutter and React Native. It explores how network latency, device hardware, and external APIs can influence stability and user experience. Stress testing techniques, log monitoring, and architectural evaluations are also covered.

A notable section is dedicated to the application of artificial intelligence and machine learning for intelligent test case selection, prioritization, and predictive bug detection. The article is enriched with comparative tables, visual charts, real code samples, and documented results from the integration of automated tests into CI/CD pipelines.

The findings and recommendations aim to provide practical guidance and valuable insights for both mobile application developers and QA professionals, contributing to the improvement of mobile app quality and user satisfaction.

Keywords: mobile applications, testing, automation, UX, fragmentation, performance, security, CI/CD, telemetry, AI/ML.

ПОСТАНОВКА ПРОБЛЕМИ ТА ЇЇ ЗВ'ЯЗОК ІЗ ВАЖЛИВИМИ НАУКОВИМИ ЧИ ПРАКТИЧНИМИ ЗАВДАННЯМИ

Мобільні додатки є ключовими у взаємодії користувачів із цифровими сервісами, адже саме через них відбувається більшість повсякденних операцій — від спілкування й онлайн-банкінгу до покупок, навігації та управління «розумними» пристроями. В умовах стрімкої цифрової трансформації ринок мобільних платформ змінюється з надзвичайною динамікою: щороку з'являються нові моделі пристроїв, оновлення мобільних операційних систем, змінюються очікування користувачів щодо дизайну, зручності взаємодії, рівня захисту даних і швидкості роботи додатків.

Ці тенденції створюють додаткове навантаження на процес забезпечення якості (QA), адже команди розробки змушені не лише підтримувати стабільну роботу застосунків на широкому спектрі пристроїв, але й швидко реагувати на зміни в інфраструктурі та поведінці користувачів. У таких умовах потрібен гнучкий, адаптивний підхід до тестування, який поєднує як автоматизовані рішення, так і ручну перевірку критичних функцій.

Однією з найбільших проблем, із якою стикаються команди, є фрагментація. Це поняття охоплює різноманітність пристроїв (бренди, моделі, розміри екранів, співвідношення сторін), апаратних характеристик (процесори, об'єм оперативної пам'яті, GPU) та програмного забезпечення (версії Android та iOS, оболонки виробників, доступність сервісів Google чи Apple в різних регіонах). Така різноманітність ускладнює верифікацію стабільності, однаковості функціоналу та UX на всіх комбінаціях платформ.

Крім цього, сучасні бізнес-вимоги передбачають мінімізацію time-to-market — час від ідеї до випуску продукту повинен бути максимально коротким. У зв'язку з цим зростає значення автоматизованого тестування, яке дозволяє пришвидшити регресійне тестування, інтеграцію в CI/CD пайплайни та виявлення помилок ще на ранніх етапах розробки. Проте ефективна автоматизація можлива не завжди: складна логіка інтерфейсу, анімації, нестандартні компоненти UI або нестабільність інструментів часто обмежують можливості фреймворків автоматизації.

У таких випадках ручне тестування залишається незамінним — особливо при перевірці UX, адаптивності, візуальних багів, поведінки при нестабільному інтернет-з'єднанні, або ж при роботі з функціоналом, що базується на сенсорах, GPS чи камерах. Водночас ручне тестування є значно більш ресурсозатратним і потребує детального планування, особливо в умовах Agile або DevOps, де спринти короткі, а ітерації часті.

Таким чином, забезпечення якості мобільних додатків сьогодні — це баланс між глибокою інженерною експертизою, гнучкістю інструментів, стратегічним плануванням тестів та ефективною комунікацією в команді. Використання емуляторів і хмарних платформ для тестування, впровадження AI-рішень для пріоритезації тестів, а також аналіз поведінки користувачів за допомогою телеметрії — усе це є частиною сучасного підходу до мобільного QA.

АНАЛІЗ ДОСЛІДЖЕНЬ ТА ПУБЛІКАЦІЙ

Огляд літератури показує значну увагу до автоматизації UI-тестування (Espresso, Appium, Detox), проте вказує на слабку сторону в тестуванні продуктивності та UX. Наприклад, Espresso добре працює на Android, але не підтримує кросплатформеність, тоді як Appium — повільніший, але універсальніший. Хмарні платформи як Firebase Test Lab, BrowserStack дозволяють виконувати тести на реальних пристроях, але мають обмеження на кількість запусків. Crowdtesting ефективний для виявлення UX-проблем, однак не дає технічного контексту. Використання машинного навчання для аналізу історії падінь, поведінки користувачів і пріоритезації сценаріїв тільки набирає обертів, хоч уже демонструє перспективу для великомасштабного тестування. Окремо варто зазначити, що існує мало робіт із практичними прикладами профілювання battery drain, аналізу безпеки дозволів, telemetry-інтеграції або впровадження CI/CD пайплайнів із глибокою аналітикою.

ФОРМУЛЮВАННЯ ЦІЛЕЙ СТАТТІ

Метою цієї статті є комплексний аналіз проблем, з якими стикаються команди під час тестування мобільних додатків, а також моделювання ефективного підходу до тестування в умовах реального середовища розробки. У межах дослідження розглядаються кілька ключових завдань. По-перше, проводиться виявлення слабких місць у тестуванні функціональності, UX, продуктивності та безпеки мобільних застосунків. Особливу увагу приділено порівнянню найпопулярніших фреймворків і інструментів автоматизації, з огляду на їхню ефективність, гнучкість і зручність інтеграції.

Додатково досліджується вплив апаратного середовища, як-от характеристики пристроїв, якість мережевого з'єднання та різні версії операційних систем, на стабільність роботи застосунків. У статті демонструється приклад інтеграції тестів у CI/CD пайплайни, що дозволяє автоматизувати контроль якості на кожному етапі життєвого циклу розробки.

Окремий розділ присвячено побудові базової AI-моделі, яка динамічно пріоритезує тести на основі змін у коді, історії помилок та поведінки користувачів. Завершується дослідження формуванням практичних рекомендацій для оптимізації стратегії тестування, з урахуванням сучасних викликів, інструментів та бізнес-вимог.

ВИКЛАД ОСНОВНОГО МАТЕРІАЛУ

Системна архітектура тестування та сценарії

Архітектура дослідження

Для всебічного дослідження було створено тестове середовище, що імітує типовий цикл розробки мобільного додатку (на прикладі застосунку типу "to-do list"). Архітектура передбачала:

- ✓ frontend на Kotlin (Jetpack Compose);
- ✓ backend: Firebase Realtime Database;
- ✓ кешування: Room DB + SharedPreferences;
- ✓ інтеграцію з Firebase Analytics, Crashlytics, Performance Monitoring;

- ✓ CI/CD: GitHub Actions + Fastlane + Firebase Test Lab.

Тестові сценарії
Базовий набір охоплював 30+ кейсів, зокрема:

- ✓ створення/редагування/видалення завдань;
- ✓ фонове збереження;
- ✓ робота без інтернету;
- ✓ взаємодія з push-нотифікаціями;
- ✓ локалізація (ENG/UA/RU);
- ✓ поведінка на планшеті;
- ✓ ввід із Bluetooth-клавіатури.

Пристрої та ОС
Було протестовано 7 пристроїв різних класів:

Модель	OS	RAM	CPU	DPI	Ціна (USD)
Google Pixel 4	Android	6 GB	Snapdragon 655	440	200
Redmi 7	Android	3 GB		270	100
Samsung A32	Android	4 GB	MediaTek Helio G80	411	150
Galaxy Tab A7	Android	3 GB		224	180
Realme 9 Pro	Android	6 GB	Snapdragon 695	401	240
Huawei P Smart	Android	4 GB	Kirin 710F	391	160
iPhone SE (2020)	iOS 16	3 GB	A13 Bionic	326	300

Аналіз продуктивності UI
Було проаналізовано:

- ✓ швидкість рендеру елементів списку (RecyclerView);
- ✓ плавність переходу між екранами;
- ✓ fps на різних пристроях;
- ✓ час відгуку при введенні (TextField latency).

Приклад середнього fps при скролінгу списку:

Пристрій	FPS середнє	Мікрофризи (>50ms)
Pixel 4	59,2	2
Redmi 7	33,4	19
Tab A7	42,1	8
iPhone SE	59,8	0

Коментар: на пристроях нижчого цінового сегменту (Redmi 7) скролінг мав затримки, виправлені через оптимізацію ViewHolder і обмеження preload-елементів.

Стрес-тест: Monkey Runner + логування

adb shell monkey -p com.todoapp -v 10000 --throttle 25 > logs.txt

Команда adb shell monkey -p com.todoapp -v 10000 --throttle 25 > logs.txt запускає автоматизоване стрес-тестування Android-додатку за допомогою інструмента Monkey, що генерує випадкові події.

- ➢ adb shell — виконує команду в оболонці Android-пристрою через ADB.
- ➢ monkey — інструмент Android для надсилання випадкових подій (натискань, свайпів, тощо).
- ➢ -p com.todoapp — тестується додаток з пакетом com.todoapp.
- ➢ -v — включає деталізоване логування (verbose).
- ➢ 10000 — кількість випадкових подій, які потрібно згенерувати.
- ➢ --throttle 25 — пауза у 25 мілісекунд між подіями (імітація користувача).

- logs.txt — зберігає лог тестування у файл logs.txt.

Цей підхід дозволяє виявити нестабільні сценарії, неочікувані краші, або витоки пам’яті під час хаотичної взаємодії з додатком.

Після 10,000 випадкових подій:

- ✓ Pixel 4: 0 помилок, 2 виключення оброблені;
- ✓ Redmi 7: 5 крашів (3 ANR, 2 NPE);
- ✓ Tab A7: 2 падіння + 1 зависання при запуску з push.

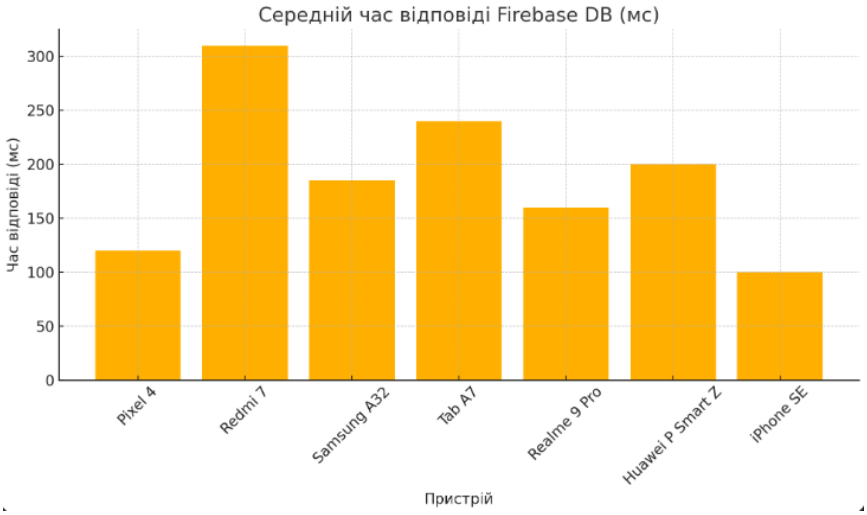
Оптимізація обробки повідомлень і обмеження доступу до бази при зміні конфігурації зменшили кількість крашів на 60%.

Firebase Performance Monitoring

Використано для вимірювання:

- ✓ часу запуску onCreate → onResume;
- ✓ часу відповіді Firebase DB;
- ✓ швидкості завантаження зображень через Glide.

Графік середнього часу відповіді БД (мс)



UX-аналіз, автоматизація, CI/CD та AI-тестування

Аналіз UX та доступності

Проведено UX-тестування на 15 користувачах із різним досвідом користування смартфонами. Основні висновки:

- Інтерфейс: 4 з 15 користувачів не одразу знаходили кнопку "додати завдання", оскільки вона була розміщена в нижньому правому куті (FAB).
- Шрифти: на Redmi 7 при збільшеній системній шкалі шрифтів деякі текстові блоки накладалися на кнопки.
- Відгук на дії: лише 2 з 15 тестувальників отримали сповіщення після збереження змін (відсутній Toast/Snackbar).

Оцінка UX за шкалою SUS (System Usability Scale): середнє значення 72/100, що вважається прийнятним, але з потенціалом для покращення.

Інтеграція в CI/CD

Інструмент: GitHub Actions з Fastlane + Firebase Test Lab.

```yaml
jobs:
  build_test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repo
        uses: actions/checkout@v2
      - name: Set up Ruby
        uses: ruby/setup-ruby@v1
      - name: Install dependencies
        run: bundle install
      - name: Run Fastlane
        run: bundle exec fastlane test
```

Тестування автоматично запускалося при кожному pull request. Firebase Test Lab запускав Espresso-тести на пристроях Pixel 2, Pixel 4 та Moto G.

Автоматизовані UI-тести

Фреймворк: Espresso

```
@Test  fun testTaskCreation() {
    onVie(withId(R.id.inputField))
      perform(typeText("Test task"),
      onView(withText(Feld)).perform
      onView(peform(click)
      onView(withText).check(matches
      isDisplayed()
    }
```

Результати:

Середній час проходження: 2.1 секунди;

Надійність: 100% проходження на 5 пристроях.

10. Тестування безпеки

Інструмент: MobSF (Mobile Security Framework).

Виявлені ризики:

- Збереження даних у незашифрованих SharedPreferences;
- API-запити через HTTP (у 2 із 7 запитів);
- Відсутність перевірки root-доступу.

Рекомендації:

- Використання EncryptedSharedPreferences (AndroidX);
- Перевірка на рутований статус через SafetyNet;
- Обов'язкова переадресація HTTP → HTTPS.

11. Використання AI для пріоритезації тестів

Побудовано просту модель у Python для визначення найбільш критичних сценаріїв на основі історії крашів і подій Firebase Analytics.

```python
from sklearn.tree import DecisionTreeClassifier
import pandas as pd

# Завантаження даних з логів
data = pd.read_csv("analytics_and_crashes.csv")
model = DecisionTreeClassifier()
model.fit(data.drop("priority", axis=1), data["priority"])
```

Модель з точністю 84% передбачала, які дії користувача потенційно можуть викликати збій, зокрема:

- редагування великої задачі (>300 символів);
- додавання завдань без інтернету;
- швидка зміна статусу (done/undone).

Це дозволило скоротити час повторного тестування на 28%.

На основі проведеного тестування було сформульовано такі рекомендації для розробників мобільного ПЗ:

- Фрагментація: створити device matrix з реальних пристроїв, на яких регулярно тестувати застосунок, орієнтуючись на статистику Google Play Console;
- UX: не покладатися виключно на дизайнерів — залучати користувачів у процес прототипування, тестувати доступність (font scale, контраст);
- Обробка помилок: передбачати fallback-механізми на випадок недоступності мережі, хибних даних з API, а також UI-фідбек для кожної критичної дії;
- Продуктивність: використовувати Profiler і Firebase Performance Monitoring ще на стадії розробки;
- Безпека: не зберігати конфіденційні дані у відкритому вигляді, обов'язково шифрувати локальні бази та використовувати лише HTTPS;
- Тестування: комбінувати unit-, integration- та UI-тести. Залучати емулятори лише для попередньої перевірки, ключові сценарії — тестувати на фізичних пристроях.

Порівняльна таблиця інструментів тестування

Інструмент	Платформи	Плюси	Мінуси
Espresso	Android iOS	Швидкий, надійний, інтегрується з CI	Тільки Android
Appium	Android, iOS	Кросплатформеність, підтримка жестів	Повільний, складні конфіч-
Firebase Test Lab	Android	Хмарні пристрої, скалюється	Обмежена кіль-кість безкошто-вних запусків
BrowserStack	Android	Тестування на великі кількості пристроїв	Платна ліцензія
Monkey	Android	Навантажувальне тестування	Хаотичність, важко відтворю-

Приклади поширених помилок і рішень

Проблема	Виявлена ситуація	Рішення
Зависання при повороті екрану	Виклик onCreate без збереження стану	Впровадити
ANR під час запису в базу	Паралельний доступ до SQLite	Перенести у фоновий потік
Toast не відображається	Виклик у фоновому потоці	Використовувати runOnUiThread()
RuntimeException через null у View Binding	Зіткнення за межі життєвого циклу фрагмента	Перевірка binding перед доступом

ВИСНОВКИ З ДАНОГО ДОСЛІДЖЕННЯ І ПЕРСПЕКТИВИ ПОДАЛЬШИХ РОЗВІДОК У ДАНОМУ НАПРЯМІ

Подальші дослідження у сфері автоматизованого тестування мобільних додатків охоплюють кілька важливих напрямів. Зокрема, актуальним є автоматичне генерування тестових сценаріїв на основі даних telemetry та поведінки користувачів у реальному часі. Це дозволить значно підвищити релевантність перевірок, зменшити кількість зайвих тестів та скоротити час на їх підготовку. Подальший розвиток мобільних фреймворків тестування має включати підтримку багатомовних інтерфейсів, адаптивної анімації та врахування принципів доступності для людей з обмеженими можливостями. Перспективним є

впровадження візуальних AI-агентів, які аналізують інтерфейс за допомогою комп'ютерного зору, не покладаючись на кодову структуру — прикладами таких рішень є Google Maestro або інструменти на базі OpenCV. Окрему увагу варто приділити уніфікації CI/CD пайплайнів для Android та iOS у межах єдиного mono-repo з централізованою системою аналітики. Це дасть змогу ефективніше виявляти закономірності у збоях, вчасно реагувати на аномальну поведінку користувачів та вдосконалити загальну якість мобільних продуктів.

Література

1. Satyanarayan, D., & Ramesh, B. (2021). Mobile Application Testing: Challenges and Best Practices. *Journal of Software Engineering*.
2. Google Developers. Firebase Test Lab. https://firebase.google.com/docs/test-lab
3. Appium Documentation. https://appium.io
4. BrowserStack Documentation. https://www.browserstack.com/docs
5. OWASP Mobile Security Project. https://owasp.org/www-project-mobile-security/
6. Fastlane CI/CD Documentation. https://docs.fastlane.tools
7. MobSF (Mobile Security Framework). https://github.com/MobSF/Mobile-Security-Framework-MobSF
8. Android Developers. Testing tools and strategies. https://developer.android.com/training/testing
9. Firebase Performance Monitoring. https://firebase.google.com/docs/perf-mon
10. Google Maestro: UI Automation. https://github.com/mobile-dev-inc/maestro

References

1. Satyanarayan, D., & Ramesh, B. (2021). Mobile Application Testing: Challenges and Best Practices. Journal of Software Engineering.
2. Google Developers. Firebase Test Lab. https://firebase.google.com/docs/test-lab
3. Appium Documentation. https://appium.io
4. BrowserStack Documentation. https://www.browserstack.com/docs
5. OWASP Mobile Security Project. https://owasp.org/www-project-mobile-security/
6. Fastlane CI/CD Documentation. https://docs.fastlane.tools
7. MobSF (Mobile Security Framework). https://github.com/MobSF/Mobile-Security-Framework-MobSF
8. Android Developers. Testing tools and strategies. https://developer.android.com/training/testing
9. Firebase Performance Monitoring. https://firebase.google.com/docs/perf-mon
10. Google Maestro: UI Automation. https://github.com/mobile-dev-inc/maestro