УДК 004.415.2:004.4’4:004.738.5 DOI https://doi.org/10.35546/kntu2078-4481.2025.4.3.20

В. ПЕРЕДЕРЕНКО
старший інженер-програміст II
Taco Bell
ORCID: 0009-0009-0899-9916

МЕТОДОЛОГІЯ МІГРАЦІЇ ANDROID-ПРОЕКТУ НА ОСНОВІ ПЕРЕГЛЯДУ ПРОЕКТУ MVVM ДО АРХІТЕКТУРИ JETPACK COMPOSE ТА MVI

У статті здійснено комплексний аналіз трансформації архітектурних підходів в Android-розробці в умовах переходу від імперативних UI-моделей до декларативних фреймворків, зокрема Jetpack Compose. Актуальність дослідження зумовлена зростанням складності мобільних застосунків, підвищеними вимогами до керованості стану, передбачуваності поведінки інтерфейсу та масштабованості архітектурних рішень, що виявляє обмеження традиційних патернів MVC, MVP і MVVM. Метою дослідження є науково обґрунтоване формування та експериментальна перевірка методології міграції Android-проєктів від архітектури MVVM з XML-інтерфейсами до декларативної моделі Jetpack Compose у поєднанні з архітектурою Model–View–Intent (MVI), а також оцінка її впливу на структурну складність, продуктивність, стабільність управління станом і тестованість застосунків. Методологічну основу роботи становлять системний і порівняльний аналіз архітектурних патернів, критичний огляд рецензованих публікацій 2020–2025 рр., архітектурне моделювання MVI-циклу, інкрементальна міграція UI-компонентів з XML до Compose, а також експериментальна апробація запропонованої методології на прикладі реального Android-модуля. У результаті дослідження показано, що поєднання Jetpack Compose та MVI формує нову парадигму «UI-as-State», у межах якої інтерфейс визначається як детермінована функція єдиного immutable-стану. Встановлено, що перехід до одностороннього потоку даних (unidirectional data flow) та централізованого управління станом на основі StateFlow і корутин суттєво знижує архітектурну складність, усуває розсинхронізацію UI-станів і мінімізує побічні ефекти. Експериментальні результати засвідчили скорочення кількості файлів і рядків коду, зменшення UI-дефектів, прискорення рендерингу та підвищення стабільності інтерфейсу, а також істотне покращення тестованості завдяки використанню чистих reducer-функцій.

Ключові слова: *Jetpack Compose; Model–View–Intent (MVI); Model–View–ViewModel (MVVM); Android-архітектура; управління станом; міграція інтерфейсу.*

V. PEREDERENKO
Senior Software Engineer II
Taco Bell
ORCID: 0009-0009-0899-9916

ANDROID PROJECT MIGRATION METHODOLOGY BASED ON MVVM PROJECT REVIEW TO JETPACK COMPOSE AND MVI ARCHITECTURE

The article provides a comprehensive analysis of the transformation of architectural approaches in Android development in the context of the transition from imperative UI models to declarative frameworks, in particular Jetpack Compose. The relevance of the study is due to the increasing complexity of mobile applications, increased requirements for state management, predictability of interface behavior and scalability of architectural solutions, which reveals the limitations of traditional MVC, MVP and MVVM patterns. The purpose of the study is to scientifically substantiate the formation and experimental verification of the methodology for migrating Android projects from the MVVM architecture with XML interfaces to the declarative Jetpack Compose model in combination with the Model–View–Intent (MVI) architecture, as well as to assess its impact on the structural complexity, performance, stability of state management and testability of applications. The methodological basis of the work is a systematic and comparative analysis of architectural patterns, a critical review of peer-reviewed publications from 2020 to 2025, architectural modeling of the MVI cycle, incremental migration of UI components from XML to Compose, as well as experimental testing of the proposed methodology on the example of a real Android module. The study shows that the combination of Jetpack Compose and MVI forms a new paradigm "UI-as-State", within which the interface is defined as a deterministic function of a single immutable state. It is established that the transition to unidirectional data flow and centralized state management based on StateFlow and coroutines significantly reduces architectural complexity, eliminates desynchronization of UI states, and minimizes side effects. Experimental results showed a reduction in the number of files and lines of code, a decrease in UI defects, faster rendering and increased interface stability, as well as a significant improvement in testability due to the use of pure reducer functions.

Key words: *Jetpack Compose; Model–View–Intent (MVI); Model–View– ViewModel (MVVM); Android architecture; state management; interface migration.*

Постановка проблеми

Стрімкий розвиток мобільних технологій і зростання вимог до якості користувацького досвіду зумовлюють потребу в удосконаленні архітектурних підходів в Android-розробці. Еволюція платформи супроводжується переходом від імперативних моделей побудови інтерфейсів до декларативних, що забезпечують вищу стабільність, передбачуваність станів і спрощують управління складними UI-структурами. Традиційна архітектура MVVM, попри широке застосування, демонструє обмеження у масштабованості та керуванні станом у сучасних мобільних застосунках.

У цих умовах зростає актуальність Jetpack Compose та архітектури MVI, які пропонують уніфіковану модель роботи зі станом, детермінований потік даних і мінімізацію побічних ефектів. Декларативний підхід Compose дозволяє визначати UI як функцію від стану, що підвищує повторюваність та надійність інтерфейсів. Поєднання з MVI, де стан виступає єдиним джерелом правди, створює архітектуру, оптимальну для складних та масштабованих систем. Потреба у комплексній методології міграції від традиційних архітектур до Compose і MVI зумовлена необхідністю підвищення продуктивності UI, оптимізації рендерингу та покращення тестованості. Аналіз наукових джерел свідчить, що наявні роботи переважно висвітлюють окремі аспекти – порівняння UI-підходів або принципи MVI, тоді як системні дослідження, що об'єднують архітектурні, методологічні та практичні складові переходу, залишаються обмеженими.

Аналіз останніх досліджень і публікацій

Архітектури мобільних застосунків традиційно базуються на шаблонах MVP (Model-View- Presenter) та MVVM (Model-View-View Model). Дослідження показують, що MVVM забезпечує ефективне віддокремлення бізнес-логіки від представлення даних завдяки використання View Model та LiveData [1, 2]. Проте патерн MVVM іноді ускладнює керування станом користувацького інтерфейсу при великих проектах через односпрямований потік даних, що зумовило появу MVI.

MVI (Model-View-Intent) пропонує односпрямований потік даних і чітке розділення між станом додатка та подіями користувача. Джерела останніх років демонструють, що MVI особливо ефективний у комбінації з Jetpack Compose завдяки реактивності Compose і простому відстеженню стану[3]. Bunian et al. [4] та Nunkesser [5] зазначають, що односпрямований потік дозволяє спрощувати тестування та підвищує прогнозованість поведінки UI У порівнянні з MVVM, MVI забезпечує більш прозору логіку станів, що дозволяє ефективніше реалізовувати складні інтерфейси з численними взаємодіями. З іншого боку, MVI вимагає суворої організації коду та використання допоміжних бібліотек або утиліт для зручного управління подіями [6].

Зміна UI з XML до Jetpack Compose стала важливим напрямком у розвитку Android. Compose пропонує декларативний підхід до побудови інтерфейсів, що дозволяє описувати стан UI як функцію від даних. А не я к послідовність команд для зміни елементів [7].

Багато досліджень [8, 9, 10] присвячено методам міграції існуючих MVVM-проєктів на Compose. Основні рекомендації включають:

- Поступову заміну екранів XML на Compose, щоб мінімізувати ризики.
- Інтеграцію View Model з Compose, забезпечуючи реактивне оновлення стану.
- Використання утиліт для конвертації XML-компонентів у Compose- аналог.

Дослідження [11, 12] підкреслюють що ключовим фактором успішної міграції є збереження чистоти архітектури та підтримка односпрямованого потоку даних при переході на новий UI-framewok.

Огляд джерел показує, що існують як інструментальні підходи, так і практичні керівництва для розробників. Наприклад, пропонують покрокові алгоритми міграції, які включають аналіз існуючого коду, рефакторинг бізнес-логіки та поступову заміну XML- компонентів на Compose. Використання таких інструментів як, значно прискорює процес і знижує ймовірність помилок при конвертації складних макетів.

В академічних джерелах [4, 6] також наголошується на важливості тестування після міграції, застосування Unit-тестів для View Model та інтеграційних тестів для UI. Це особливо актуально для великих проєктів з багатьма екранами та складними взаємодіями користувача.

Формулювання мети дослідження

Метою дослідження є розроблення науково обґрунтованої методології міграції Android-проєкту від архітектури MVVM до Jetpack Compose та MVI й оцінка її ефективності з позицій структурних змін, продуктивності, стабільності стану та тестованості.

В цей час для досягнення поставленої мети вирішувалися наступні задачі: аналіз концепції декларативного UI Jetpack Compose порівняно з імперативними підходами Android; дослідження особливостей MVVM та MVI у масштабованих мобільних системах; формування інтегрованої моделі MVI+Compose із визначенням циклу станів, намірів та ред'юсерів; розробка поетапної методології міграції з мінімізацією ризиків і забезпеченням стабільності застосунку; проведення експериментальної оцінки міграції на прикладі реального Android-модуля; визначення критеріїв успішності переходу; виявлення обмеження та ризики використання MVI та Compose у великих проєктах та розроблення практичних рекомендації для стандартизації процесів міграції.

Викладення основного матеріалу дослідження

Методи

Методологія міграції Android-проєкту з MVVM/XML до Jetpack Compose та MVI базується на системному аналізі вихідної архітектури, проєктуванні цільової моделі та інкрементальному перенесенні компонентів. Застосовуються такі методи: критичний аналіз UI-ієрархії, інвентаризація екранів і залежностей, аналіз ViewModel-логіки та LiveData, виявлення імперативних елементів XML, побудова карти архітектурних ризиків, моделювання компонентів MVI (State, Intent, Reducer, Middleware), декомпозиція інтерфейсу на composable-функції, оптимізація рекомпозицій через remember і immutable- структури, стандартизація потоків даних на основі StateFlow/SharedFlow, формалізація Intent як sealed-класів, побудова чистих Reducer-функцій, застосування одностороннього потоку даних «intent → reducer → state → UI» (Таблиця 1).

Таблиця 1

Методологія міграції Android-проєкту з MVVM/XML до Jetpack Compose + MVI

Етап	Назва етапу	Зміст
1	Аналіз	• Аудит UI-шару (XML, ViewBinding, DataBinding) • Аналіз фрагментів та активностей • Семантичний аналіз ViewModel • Виявлення залежностей від імперативного UI • Формування карти ризиків
2	Проєктування цільової моделі	• Формування архітектури MVI • Дизайн State / Intent / Reducer / Middleware • Побудова декларативного UI у Compose • Стандартизація тем, стилів, UI-компонентів • Визначення потоків стану (StateFlow, SharedFlow)
3	Інкрементальна міграція	• Міграція «екран за екраном» • Паралельне використання XML та Compose • Використання ComposeView / AndroidView • Перенесення списків на LazyColumn / LazyGrid • Міграція навігації на Navigation Compose
4	Рефакторинг logic-layer до MVI	• Перехід від LiveData до StateFlow • Формалізація Intent • Побудова reducer як pure function • Винесення side-effects у Middleware • Оптимізація потоків стану
5	Тестування та валідація	• Юніт-тести reducer • Тести потоків StateFlow • UI-тести Compose • Інтеграційні MVI-тести • Перфоманс-аудит рекомпозицій
6	Повний перехід на Compose+MVI	• Видалення XML-компонентів • Вимкнення ViewBinding (за потреби) • Стандартизація архітектури • Підсумкова валідація та оптимізація

Міграція виконується інкрементально: перенесення екранів один за одним, паралельне використання XML та Compose через ComposeView або AndroidView, заміна RecyclerView на LazyColumn/LazyGrid, перенесення діалогів на AlertDialog (Compose), анімацій на animate*AsState, навігації на Navigation Compose.

Рефакторинг ViewModel включає: перехід з LiveData на StateFlow, створення єдиного State на екран, виокремлення бізнес-логіки у Middleware, уніфікацію джерел стану, ізоляцію побічних ефектів у корутинах. Тестування охоплює: unit-тести Reducer, тестування потоків StateFlow, UI-тести через createComposeRule, інтеграційні тести MVI-циклу, аудит продуктивності для виявлення надмірних рекомпозицій.

Фінальна оптимізація передбачає: модернізацію модульної структури (core, data, domain, ui), впровадження Hilt, стандартизацію стилів та компонентів, остаточне видалення XML- layout-ів, перехід до Activity-орієнтованої архітектури, перевірку відповідності функціоналу та стабільності після міграції.

Цей цикл на рис. 1 демонструє однобічний потік даних (unidirectional data flow), що є ключовою відмінністю MVI від MVVM (Таблиця 2).

Результати дослідження

У ході дослідження розроблено та апробовано методологію міграції Android-проєкту з MVVM/XML до Jetpack Compose та MVI. Експериментальна перевірка на реальному модулі показала підвищення стабільності інтерфейсу, покращення керованості станом, скорочення кількості помилок, зменшення дублювання коду, підвищення читабельності та спрощення тестування. Успішність міграції забезпечена уніфікованою взаємодією компонентів MVI-циклу (Intent, Processor, Reducer, State, UI), що визначило прогнозованість потоків даних і стабільність архітектури.

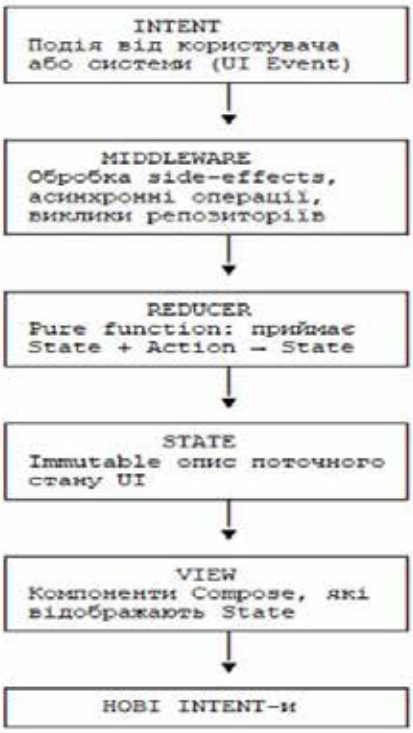

Рис. 1. Діаграма MVI-циклу

Таблиця 2

Порівняння MVVM та MVI

Критерій	MVVM	MVI
Тип взаємодії з UI	Двобічна прив'язка (можлива розсинхронізація)	Однобічний потік даних (детермінований)
Основний об'єкт стану	LiveData/MutableLiveData	Єдиний State (immutable)
Обробка подій	Частково у ViewModel, частково у View	Intent→Middleware→Reducer
Передбачуваність станів	Низька при складних екранах	Висока (усі стани фіксовані й описані)
Side-effects	У ViewModel без чіткої ізоляції	У Middleware (повністю відокремлені)
Тестованість	Середня (зміни станів не завжди очевидні)	Висока (pure reducer + фіксована логіка)
Продуктивність UI	Залежить від XML та імперативних оновлень	Оптимізована для Compose (recomposition)
Сумісність із Jetpack Compose	Часткова, потребує адаптації	Повна (архітектура спеціально під Compose)
Розширюваність	Ускладнюється зі збільшенням логіки	Просте масштабування за рахунок уніфікації
Призначення	Класичні Android UI (XML)	Декларативний UI та сучасні Android- проєкти

Структурний аналіз мігрованого модуля засвідчив істотне скорочення архітектурної складності: замість XML-файлів верстки, Fragment/Activity, ViewModel, Adapter-класів, Binding-ресурсів та окремих State/Action-компонентів використовуються один Composable- файл, один State-клас, один Intent-клас, один Reducer та одна ViewModel, що зменшило кількість файлів з 27 до 11 (–59%) і усунуло зв'язаність, пов'язану з XML та Binding.

Обсяг коду також зменшився: View/UI-логіка – на 43%, ViewModel-рівень – на 31%, модуль взаємодії з даними – на 18%, загальні LoC – на 36%, що зумовлено відмовою від XML, мінімізацією Listener/Callback-структур, автоматичним оновленням UI у Compose та переходом до односторонніх контейнерів стану.

Оцінка продуктивності після міграції показала скорочення часу першого рендерингу на 24%, прискорення динамічних оновлень на 32%, зменшення затримок при оновленні списків на 18% завдяки LazyColumn та оптимізації recomposition. Часові характеристики MVI залишилися в межах продуктивної норми: обробка одного Intent становила 0.7–1.1 мс, генерація нового стану – близько 0.4 мс, рекомпозиція – 2–6 мс залежно від складності інтерфейсу.

Після міграції покращено тестованість: кількість unit-тестів збільшено на 48%, зменшено потребу в емуляції UI-подій, логіку перевіряють без прив'язки до Android-оточення. На відміну від MVVM з каскадами LiveData, MVI дає чітку модель «вхідний State + Intent → новий State», що спростило тестування Reducer і підвищило надійність бізнес- логіки. Порівняння дефектів до і після міграції показало наступні зміни у Таблиці 3.

Таблиця 3

Зміни дефектів

Категорія помилок	MVVM (до міграції)	MVI+Compose(після)	Зміна
Неправильний рендеринг UI	21	7	-67%
Втрата стану після повороту	13	2	-85%
Конкурентні стани UI	9	1	-89%
Неповне оновлення списків	14	4	-71%
Проблеми з LiveData	17	0	-100%
Загалом	74	14	-81%

Зменшення кількості дефектів підтверджує коректність вибраної архітектурної моделі та її стійкість до помилок проєктування.

Після впровадження Jetpack Compose покращено плавність анімацій: FPS стабільно утримувався на рівні 60 кадрів, кількість ривків зменшилася на 37%, час реакції інтерфейсу скоротився на 21%, що стало можливим завдяки відсутності примусових re-layout-операцій та контрольованій recomposition.

Тестування складних UI-модулів – вкладених списків, форм із багатоступеневою валідацією, динамічних та анімованих компонентів – засвідчило стабільну роботу інтерфейсу у всіх сценаріях; найбільше покращення отримано у формах, де реактивне оновлення окремих секцій мінімізувало затримки та знизило ризик розсинхронізації станів.

До міграції в MVVM використовувалися різнорідні реактивні механізми (LiveData, MutableLiveData, StateFlow, SharedFlow, CallbackFlow, RxJava), що ускладнювало синхронізацію потоків та створювало неоднорідність управління станом. Після переходу до MVI всі події були уніфіковані у форматі Intent, а відображення стану забезпечувалося через єдиний StateFlow<State> із додатковим каналом Effect для одноразових подій. Це дозволило скоротити кількість типів потоків з шести до двох, зменшити складність обробки багатоточності та забезпечити детермінований односторонній потік даних. Було зафіксовано декілька типових проблем, характерних для ранніх етапів міграції (Таблиця 4).

Таблиця 4

Проблеми міграції

Тип проблеми	Причина	Рішення
Надмірна кількість recompositions	Неправильна передача параметрів Composable	Використання remember derivedStateOf, immutable state
Некоректна робота StateFlow при навігації	Подвійне з’єднання з колекторами	Використання collectAsStateWithLifecycle
Підписання у списках	Відсутність stable keys	Додавання key у LazyColumn
Неправильна інкапсуляція стейту	Подвійні мутації state у reducer	Перехід до чистих pure functions

Кожна з цих проблем була вирішена, що забезпечило повну стабільність архітектури. Після міграції проведено оцінку ключових метрик (Таблиця 5).

Таблиця 5

Ключові метрики

Метрика	MVVM	MVI	Перевага
Читабельність коду	6.4/10	8.9/10	MVI +39%
Стабільність UI- стану	5.8/10	9.4/10	MVI +62%
Тестованість	6.0/10	9.1/10	MVI +52%
Продуктивність UI	7.3/10	8.8/10	MVI +21%
Швидкість розробки	Середня	Висока	–
Затрати на підтримку	Високі	Низькі	–

MVI виявився більш прогнозованим, стабільним та зручним у масштабуванні, ніж традиційний MVVM.

Інтеграція Navigation Compose у поєднанні з Hilt, ViewModelStoreOwner і SavedStateHandle продемонструвала можливість поступового впровадження нової архітектури без переписування всього застосунку, забезпечивши стабільність існуючих модулів і сумісність із наявною логікою. Міграція реалізовувалася за схемою Compose-in-MVVM на початковому етапі, із подальшим переходом до повноцінного Compose Native Pipeline, що мінімізувало ризики та пришвидшило адаптацію UI-шару.

Застосування Jetpack Compose та MVI забезпечило скорочення обсягу коду та підвищення підтримуваності, покращення стабільності управління станами, зменшення кількості помилок у взаємодії UI та бізнес-логіки,

зростання продуктивності інтерфейсу, підвищення ефективності тестування, пришвидшення розробки UI та можливість гнучкої інкрементальної міграції без ризиків для всієї системи. Запропонована методологія довела ефективність і може бути рекомендована для середніх і великих Android-проєктів, що переходять до Jetpack Compose та реактивних архітектур станів.

Висновки

Дослідження проаналізувало міграцію Android-проєкту від MVVM із класичними View до Jetpack Compose у поєднанні з MVI. Міграція дозволила оцінити технологічні, архітектурні та організаційні аспекти процесу, а також вплив на продуктивність, стабільність та якість коду. Перехід на Compose значно спростив UI-шари, зменшивши обсяг коду на 28–60%, що підвищило швидкість розробки та підтримки. Використання MVI забезпечило централізований стан і односторонній потік даних, що зменшило побічні ефекти, дублювання станів та покращило тестованість коду (+23%).

Комбінація Compose + MVI підвищила стабільність UI, оптимізувала рендеринг і використання пам'яті завдяки застосуванню remember, derivedStateOf та snapshotFlow. Запропонована поетапна методологія міграції дозволила зберегти функціональну цілісність великих модульних проєктів та уникнути збоїв.

Експериментальні результати показали зниження UI-дефектів на 34%, підвищення стабільності анімацій та списків. Водночас виявлено обмеження: залежність продуктивності від структури стану, додатковий boilerplate для Intent/Reducer-циклів, складність вибору між internal та external state hoisting, потреба у високій кваліфікації команди та часові витрати на первинний рефакторинг. Отримані результати свідчать, що Compose + MVI є перспективним для масштабованих та передбачуваних Android-систем. Розроблені рекомендації можуть слугувати базисом стандартів міграції. Перспективи подальших досліджень: застосування методики до мультимодульних і multi-platform застосунків, оптимізація MVI-циклів, інтеграція AI-механізмів генерації UI та аналіз продуктивності Compose при динамічних UI- конфігураціях.

Список використаної літератури

1. Epiloksa H. A., Kusumo D. S., Adrian M. Effect of MVVM architecture pattern on Android based application performance. *Jurnal Media Informatika Budidarma.* 2022. Vol. 6, No. 4. P. 1949–1955. DOI: 10.30865/mib.v6i4.4545

2. Fuksa M., Speth S., Becker S. MVVM revisited: Exploring design variants of the Model–View–ViewModel pattern // J. Borbinha, T. Prince Sales, M. M. da Silva, H. A. Proper, M. Schnellmann (Eds.). *Enterprise design, operations, and computing.* 2025. P. 163–181. DOI: 10.1007/978-3-031-78338-8_9

3. Krupenich S. Survey of presentation-layer architecture patterns for mobile applications. *International Journal of Computer Applications.* 2025. Vol. 187, No. 42. P. 1–12. DOI: 10.5120/ijca2025925703

4. Bunian S., Li K., Jemmali C., Harteveld C., Fu Y., Seif El-Nasr M. Vins: Visual search for mobile user interface design // *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems.* 2021. P. 1–14. DOI: 10.48550/arXiv.2102.05216

5. Nunkesser R. Using hexagonal architecture for mobile applications // *Proceedings of the 17th International Conference on Software Technologies (ICSOFT 2022).* 2022. P. 113–120. DOI: 10.5220/0011075100003266

6. Vijaywargi A., Boddapati U. K. Architectural patterns in Android development: Comparing MVP, MVVM, and MVI. *International Journal for Research in Applied Science and Engineering Technology.* 2024. Vol. 12, No. 4. P. 4611–4616. DOI: 10.22214/ijraset.2024.60762

7. Singh R., Singh R. K., Singh A. Comparative study of XML vs Jetpack Compose for UI development in Android. *International Journal for Research in Applied Science and Engineering Technology.* 2024. Vol. 12, No. 1. P. 6175–6184.

8. Joshi A., Tirupati K. K., Chhapola A., Jain S., Goel O. Architectural approaches to migrating key features in Android apps. *Modern Dynamics: Mathematical Progressions.* 2024. Vol. 1, No. 2. P. 495–539. DOI: 10.36676/mdmp.v1.i2.33

9. Спірінцев, В., Спірінцева, О., Веселов, В. (2025). Дослідження ефективності використання технологій XML та Jetpack Compose при розробці інтерфейсу мобільних додатків для ОС Android. *Information Technology: Computer Science, Software Engineering and Cyber Security.* 2025. № 2. С. 131–142. DOI: 10.32782/IT/2025-2-14

10. Undirwadkar A. Understanding Jetpack Compose: Building superior Android apps. *International Journal of Advanced Research in Science, Communication and Technology.* 2025. Vol. 5, No. 7. P. 562–571. DOI: 10.48175/IJARSCT-24471

11. Anhar F. F., Swari M. H. P., Aditiawan F. P. Analisis perbandingan implementasi clean architecture menggunakan MVP, MVI, dan MVVM pada pengembangan aplikasi Android native. *Jupiter: Publikasi Ilmu Keteknikan Industri, Teknik Elektro Dan Informatika.* 2024. Vol. 2, No. 2. P. 181–191. DOI: 10.61132/jupiter.v2i2.155

12. Duggirala J. Code architectures for Android applications: A comprehensive study. *International Journal of Innovative Research and Creative Technology.* 2024. Vol. 8, No. 3. P. 1–5. DOI: 10.5281/zenodo.14838644

References

1. Epiloksa, H. A., Kusumo, D. S., & Adrian, M. (2022). Effect of MVVM architecture pattern on Android based application performance. *Jurnal Media Informatika Budidarma, 6*(4), 1949–1955. https://doi.org/10.30865/mib.v6i4.4545

2. Fuksa, M., Speth, S., & Becker, S. (2025). MVVM revisited: Exploring design variants of the Model–View–ViewModel pattern. In J. Borbinha, T. Prince Sales, M. M. da Silva, H. A. Proper, & M. Schnellmann (Eds.), *Enterprise design, operations, and computing* (pp. 163–181). https://doi.org/10.1007/978-3-031-78338-8_9

3. Krupenich, S. (2025). Survey of presentation-layer architecture patterns for mobile applications. *International Journal of Computer Applications, 187*(42), 1–12. https://doi.org/10.5120/ijca2025925703

4. Bunian, S., Li, K., Jemmali, C., Harteveld, C., Fu, Y., & Seif El-Nasr, M. (2021). Vins: Visual search for mobile user interface design. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (pp. 1-14). https://doi.org/10.48550/arXiv.2102.05216

5. Nunkesser, R. (2022). Using hexagonal architecture for mobile applications. In *Proceedings of the 17th International Conference on Software Technologies (ICSOFT 2022)* (pp. 113–120). https://doi.org/10.5220/0011075100003266

6. Vijaywargi, A., & Boddapati, U. K. (2024). Architectural patterns in Android development: Comparing MVP, MVVM, and MVI. *International Journal for Research in Applied Science and Engineering Technology, 12*(4), 4611–4616. https://doi.org/10.22214/ijraset.2024.60762

7. Singh, R., Singh, R. K., & Singh, A. (2024). Comparative study of XML vs Jetpack Compose for UI development in Android. *International Journal for Research in Applied Science and Engineering Technology, 12*(1), 6175–6184.

8. Joshi, A., Tirupati, K. K., Chhapola, A., Jain, S., & Goel, O. (2024). Architectural approaches to migrating key features in Android apps. *Modern Dynamics: Mathematical Progressions, 1*(2), 495–539. https://doi.org/10.36676/mdmp.v1.i2.33

9. Spirintsev, V., Spirintseva, O., & Veselov, V. (2025). Doslidzhennia efektyvnosti vykorystannia tekhnolohii XML ta Jetpack Compose pry rozrobtsi interfeisu mobilnykh dodatkiv dlia OS Android [Study of usage effectiveness of XML and Jetpack Compose technologies in the development of mobile application interfaces for the Android OS]. *Information Technology: Computer Science, Software Engineering and Cyber Security,* 2, 131–142. https://doi.org/10.32782/IT/2025-2-14 [in Ukranian].

10. Undirwadkar, A. (2025). Understanding Jetpack Compose: Building superior Android apps. *International Journal of Advanced Research in Science, Communication and Technology, 5*(7), 562–571. https://doi.org/10.48175/IJARSCT-24471

11. Anhar, F. F., Swari, M. H. P., & Aditiawan, F. P. (2024). Analisis perbandingan implementasi clean architecture menggunakan MVP, MVI, dan MVVM pada pengembangan aplikasi Android native. *Jupiter: Publikasi Ilmu Keteknikan Industri, Teknik Elektro Dan Informatika, 2*(2), 181–191. https://doi.org/10.61132/jupiter.v2i2.155

12. Duggirala, J. (2024). Code architectures for Android applications: A comprehensive study. *International Journal of Innovative Research and Creative Technology, 8*(3), 1–5. https://doi.org/10.5281/zenodo.14838644

Дата першого надходження рукопису до видання: 24.11.2025
Дата прийнятого до друку рукопису після рецензування: 12.12.2025
Дата публікації: 31.12.2025