MVVM in the Era of Modern Android: A Systematic Literature Review and Taxonomy of Architectural Trade-offs

Fabian-Enrique Suarez-Carvajal
Faculty of Engineering, Universidad Autnoma de Bucaramanga,
Bucaramanga, Colombia
fsuarez120@unab.edu.co

and

Rene Alejandro Lobo Quintero
Faculty of Engineering, Universidad Distrital Francisco Jos de Caldas,
Bogot, Colombia
raloboq@udistrital.edu.co

and

Julian Santoyo
Faculty of Engineering, Universidad Autnoma de Bucaramanga,
Bucaramanga, Colombia
jsdiaz@unab.edu.co

and

Javier Pinzon-Castellanos
Faculty of Engineering, Universidad Autnoma de Bucaramanga,
Bucaramanga, Colombia
jpinzon408@unab.edu.co

and

Yamid Gamba-Gonzalez
Faculty of Engineering, Universidad Autnoma de Bucaramanga,
Bucaramanga, Colombia
ygamba@unab.edu.co

Abstract

Model-View-ViewModel (MVVM) has become a widely adopted architectural pattern in mobile application development, driven by modern programming languages and the transition toward declarative user interfaces such as Jetpack Compose. Despite its popularity, empirical evidence regarding the impact of MVVM on software quality attributes remains fragmented and often contradictory, particularly within the Android ecosystem. This paper presents a Systematic Literature Review (SLR) of 78 primary studies published between 2017 and 2025, following the guidelines of Kitchenham and Charters [1] and sourced from five major scientific databases. The review focuses on MVVM implementations in modern Android development, with contextual comparisons to iOS and cross-platform environments.

The study contributes (i) a taxonomy of research intentions and adoption contexts for MVVM in mobile software development, (ii) a systematic synthesis of reported trade-offs between key quality attributes, most notably improvements in testability and maintainability versus performance and resource overhead and (iii) an analysis of architectural challenges associated with reactive and declarative paradigms. The results indicate a strong predominance of Android-focused studies and a technological shift toward coroutine and flow based implementations, while also revealing recurring performance and memory concerns linked to excessive data binding and observer usage.

The findings suggest that, although MVVM effectively supports maintainability and testing practices, its uncritical adoption in modern mobile applications may lead to architectural degradation if performance and energy implications are not explicitly addressed. The paper concludes by outlining a research agenda focused on empirical validation in large-scale projects and the standardization of MVVM practices in declarative UI environments.

Keywords: Model-View-ViewModel, MVVM, Mobile Software Architecture,Android Ecosystem,Systematic Literature Review, Software Quality,Jetpack Compose,Technical Debt

1 Introduction

The rapid spread of mobile applications across platforms requires robust, easy-to-maintain architectural models to address the growing complexity of user interfaces and underlying business logic [2]. This context has increased the complexity of mobile software, especially the need to maintain applications on multiple platforms (Android, iOS, and increasingly, cross-platform) with fast release cycles and high quality [3].

Specifically, MVVM facilitates a clear separation between the user interface, the application's data and business rules, and an abstract representation of the view's state and behavior [4, 5]. This architectural paradigm, initially introduced by Microsoft for platforms such as WPF and Silverlight, has gained significant traction in mobile development environments such as iOS, Android, and various cross-platform frameworks due to its ability to mitigate "massive view controllers" issues and its compatibility with reactive programming principles [6, 7].

Traditional architectural patterns have shown significant limitations in modern mobile development, where the complexity of applications demands higher code quality and maintainability. Recent empirical studies identify the lack of a suitable architecture as one of the main technical challenges impacting the evolution of mobile software [8]. In response, architectures such as Model-View-ViewModel (MVVM) have gained industry adoption for their ability to decouple the interface from the logic, although academic literature warns that the choice of pattern alone does not guarantee the absence of defects, underscoring the need to objectively measure its impact [9]. To address this evaluation systematically, this study follows the Goal-Question-Metric (GQM) paradigm originally formalized in software engineering by Basili & Rombach [10], defining the objective of this review as analyzing the Model-View-ViewModel (MVVM) architecture with the purpose of characterizing its adoption, comparison with other architectures, metrics used, benefits, limitations, and research gaps from the point of view of researchers and mobile software development professionals in the context of native and cross-platform mobile applications published between 2017 and 2025. Both native developments (Android with Kotlin/Java, iOS with Swift/Objective-C) and cross-platform developments (Xamarin, Flutter (when using MVVM), React Native with similar patterns, etc.) are included.

It should be noted that although the MVVM pattern is applicable to various platforms (iOS, Xamarin, MAUI), this study is deliberately limited to the native Android ecosystem. This methodological decision responds to three critical factors: (1) Android's hegemony in the global market, which exacerbates the need for robust architectural standards; (2) the unique fragmentation of hardware and operating system lifecycle,

which presents state management challenges not replicable on other platforms; and (3) Google's explicit promotion of MVVM as an official standard, which has generated a specific body of literature that requires independent synthesis.

This study presents a comprehensive systematic review of the MVVM architecture in the mobile ecosystem, based on the analysis of a corpus of 78 primary studies rigorously selected from 2017 to 2025. The main contribution of this work lies in the construction of a comprehensive characterization that classifies adoption contexts and the predominant quality metrics, allowing us to contrast the theoretical benefits promoted by the official documentation with the tangible limitations, such as technical debt and performance challenges, reported in the scientific literature. It also consolidates an updated knowledge base that identifies critical gaps in the state of the art and defines future lines of research to guide both academia and industry.

The manuscript begins by establishing the background and related work to situate the research in the current technological context. Next, the methodological protocol, designed under the software engineering guidelines of Kitchenham & Charters [1], is detailed. On this methodological basis, quantitative and qualitative findings, organized by research questions, are presented, followed by a critical discussion that interprets their practical and theoretical implications. Finally, the study addresses threats to validity inherent in the review process and summarizes the main conclusions along with recommendations for future work.

2 Background

The development of mobile applications for Android has established itself as one of the most dynamic domains of software engineering, characterized by heterogeneous devices and technological volatility that challenge the stability of projects [11]. In this hostile environment, software architecture transcends mere structure; it becomes the determining factor of critical attributes such as maintainability, technical debt, and energy consumption [12].

Since 2017, the industry has dogmatically adopted the Model-View-ViewModel (MVVM) architecture promoted by Google and Kotlin. However, this standardization masks a disturbing paradox: while its adoption is widespread, scientific evidence of its actual effectiveness is contradictory. Empirical studies warn that the overload of abstractions in layered architectures can lead to higher energy consumption than in monolithic architectures if not rigorously optimized [13].

Added to this uncertainty is a problem of accidental cognitive complexity. The evolution of MVVM towards a purely reactive paradigm (using Kotlin Flows and Coroutines) has transformed linear control flow into a complex asynchronous system [14]. Using Coroutines and Flows has increased the mental load for developers. This complexity often leads to the 'God-ViewModel' antipattern, where the ViewModel handles too many responsibilities like navigation and data mapping [15]. This transition has triggered a propensity for concurrency errors and race conditions in state management, making debugging difficult and generating architectural "Code Smells" [14]. This violation of the Single Responsibility Principle creates technical debt that is difficult to detect with standard static analysis tools [16, 15].

Likewise, the ecosystem's volatility has generated a phenomenon of "accelerated architectural erosion." The rapid obsolescence of key components, such as the replacement of LiveData by StateFlow and the emergence of declarative UI with Jetpack Compose, invalidates much of the previous literature [17] and leaves projects in a state of "hybrid architecture" or zombie. This technological schism raises a critical question: are the classic MVVM principles valid in the new declarative environment, or are we perpetuating obsolete practices that create technical lag?

The absence of a synthesis that contrasts these limitations (performance, reactive complexity, and obsolescence) with the supposed benefits leaves industry and academia without an empirical compass.

3 Method

Research method / procedure

To comprehensively address the issue raised and mitigate the selection bias inherent in traditional reviews, this work used the Systematic Literature Review (SLR) methodology. The protocol design strictly adhered to the fundamental guidelines for software engineering proposed by Kitchenham & Charters [1] to ensure the transparency and auditability of the process.

Figure 1 systematizes the methodological flow adopted in this research, which strictly adheres to the three fundamental phases proposed by the guidelines of Kitchenham & Charters [1]. In the Planning Phase, the study's validity was established by designing an iterative review protocol, which was refined in advance to ensure the accuracy of the search chain. The Execution Phase operationalized the protocol by searching five high-impact databases (IEEE Xplore, Scopus, WoS, ScienceDirect, and SpringerLink), applying sequential

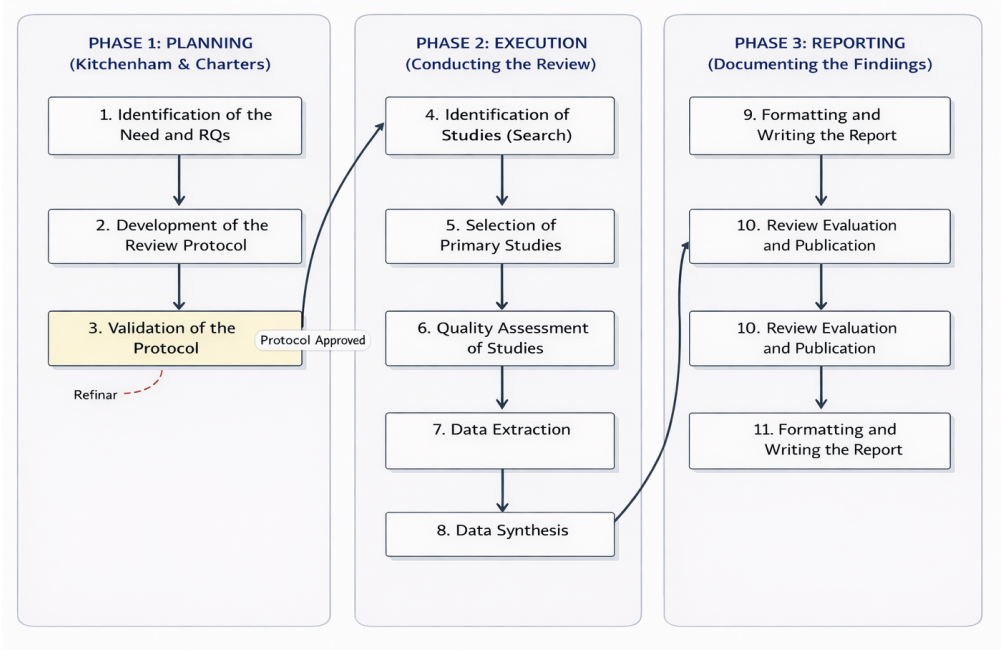

Figure 1: Protocol-Driven Systematic Literature Review Process

selection filters, and conducting a methodological quality assessment, resulting in a final corpus of 78 primary studies. Finally, the Reporting Phase articulated the synthesis of the evidence extracted to answer the research questions and structure the conclusions of this manuscript.

Research questions

Based on the problem identified, and with the aim of addressing it from the principles of software engineering and technological development, the following research questions are posed to guide the state of the art:

Table 1: Research Questions and Motivation for the Systematic Literature Review

Code	Research Question	Motivation
RQ1	How has the reference implementation of MVVM evolved in the Android ecosystem (2017–2025) following the adoption of modern native paradigms (Kotlin, Coroutines, Flow)?	The transition from Java to Kotlin as the official language for Android has redefined development practices, enabling features such as coroutines and flows that naturally integrate with reactive MVVM components (e.g., LiveData). This research question aims to synthesize and classify emerging implementation patterns (e.g., StateFlow, ViewModel with SavedStateHandle, dependency injection) reported in the literature as de facto standards or best practices within the modern Kotlin ecosystem.
RQ2	What is the reported empirical impact of MVVM on critical quality attributes in Android (maintainability, testability, and energy efficiency) compared to traditional patterns (MVC, MVP)?	To move beyond anecdotal comparisons by synthesizing empirical evidence that evaluates whether the architectural complexity of MVVM results in measurable improvements in software quality, in alignment with ISO/IEC 25010.
RQ3	What technical debt and architectural adaptation challenges arise during the transition from imperative views (XML) to declarative interfaces (Jetpack Compose) within the MVVM pattern?	To address current industry uncertainty regarding state management and data binding adaptation in declarative UI paradigms, identifying patterns of architectural obsolescence and migration challenges.
RQ4	What is the level of empirical rigor and maturity of evidence in current studies on MVVM in Android?	To critically assess the methodological quality of existing studies, distinguishing between practitioner-oriented opinions and scientifically grounded evidence, and identifying gaps in research reliability.

Search strategy

The main search string was:

```
("MVVM" OR "Model-View-ViewModel") AND ("Mobile Application" OR "Android" OR "iOS" OR "Cross-platform"
OR "Kotlin" OR "Swift") AND ("Architecture" OR "Design Pattern" OR "Performance" OR "Maintainability" OR
"Energy Efficiency")
```

The search strategy was designed with a holistic approach to capture the state of the art of MVVM across the mobile spectrum. While the primary focus of the study is on the Android ecosystem, explicit terms for other platforms (iOS, Cross-platform) were included to identify comparative studies that allow contextualizing the performance of MVVM against native and hybrid alternatives. A third block of terms ("Architecture," "Performance") was added to filter out trivial applications and ensure that the results addressed software quality attributes. In the databases: IEEE Xplore, Scopus, Web of Science, ScienceDirect, and SpringerLink.

Study Selection Process

The selection process was carried out according to a strict multi-stage protocol to ensure reproducibility and minimize selection bias, as illustrated in the flowchart (Figure 1).

Identification and Screening: After searching the five databases, duplicates were removed, and titles and abstracts were screened. Studies that were clearly irrelevant or did not address MVVM architecture as the central focus were discarded.

Eligibility Assessment: The preselected candidates underwent full-text reading to verify compliance with the technical criteria (CI/CE), prioritizing those with empirical validation.

Bias Control and Consensus: The selection was made independently by two researchers. To validate the reliability of the process, Cohen's Kappa agreement index was calculated on a random sample of 20% of the studies, yielding a value of k=0.86, indicating "almost perfect" agreement according to the Landis & Koch scale [18]. Residual discrepancies were resolved by consensus and arbitration by a third expert researcher.

Table 2: Inclusion and Exclusion Criteria for Study Selection

ID	Criterion	Justification
CI1	Topic and scope	Studies that implement, evaluate, or compare the MVVM architectural pattern in mobile application development. While the primary focus is on the Android ecosystem, studies targeting iOS or multiplatform environments are included when they provide relevant comparative insights applicable to Android-based MVVM implementations.
CI2	Type of evidence	Studies reporting empirical quantitative metrics (e.g., performance, energy consumption) or qualitative assessments (e.g., maintainability, adoption, developer perception) related to the use of MVVM.
CI3	Time window	Studies published between 2017 and 2025, reflecting the period associated with the consolidation of Kotlin and modern Android development paradigms.
CE1	Outside technological context	Studies exclusively focused on non-Android platforms (e.g., MVVM in SwiftUI or WPF) that do not provide transferable comparisons or insights relevant to the Android ecosystem.
CE2	Grey literature and abstracts	Grey literature, including undergraduate or master theses, posters, keynotes, blogs, white papers, and short papers (less than four pages) that lack empirical evaluation or methodological rigor.
CE3	Language	Studies whose full text is not available in English.

Data Extraction Procedure

To systematize the heterogeneous evidence and answer the research questions, a structured extraction form was designed. This instrument was iterated following a pilot test with 10 studies to ensure its ability to capture modern technical details (e.g., Kotlin Coroutines, Jetpack, or Compose).

The extracted information was organized into a data matrix linking each variable to the RQs, as detailed in Table 3.

Assessment of Study Quality

Given the variability in the rigor of the primary studies, a methodological quality assessment (QA) was performed to weigh the reliability of the evidence. The questionnaire by Kitchenham et al. [1] was adapted,

Table 3: Data Extraction Categories and Their Relationship to the Research Questions

Data Category	**Specific Fields Extracted**	**Relationship to RQs**
Technological context	Target platform (Android, iOS, cross-platform), programming language (Java, Kotlin, Swift), and UI framework (XML, Jetpack Compose).	RQ1 (Evolution): Enables tracing the technological transition from Java to Kotlin and the adoption of modern Android development components and paradigms.
Architectural design	Architectural patterns used for comparison (MVC, MVP, MVI) and supporting components (e.g., LiveData, StateFlow, Hilt).	RQ3 (Challenges): Supports the identification of architectural adaptation issues, including architectural erosion and integration with reactive and declarative paradigms.
Quality assessment	Reported quality metrics (e.g., testability, performance, energy consumption) and type of empirical validation employed.	RQ2 (Impact): Enables the analysis of trade-offs between software quality attributes (e.g., maintainability) and efficiency-related concerns (e.g., performance and energy usage).
Methodological rigor	Research strategy (experiment, case study), sample size, and study context (academic vs. industrial).	RQ4 (Maturity): Provides the basis for assessing the maturity and empirical rigor of existing evidence and for discussing the identified theory–practice gap in the literature.

defining four key criteria evaluated on a ternary scale (Yes=1, Partial=0.5, No=0):

- QA1: Are the architectural objectives and context of the application clearly defined?
- QA2: Is the methodology used appropriate for evaluating the MVVM pattern?
- QA3: Are explicit metrics or robust qualitative evidence reported to support the conclusions?
- QA4: Do the authors discuss threats to validity or limitations of their implementation?

Although base studies were not excluded due to low scores to maximize descriptive coverage, the impact analysis (RQ2) prioritized studies with a QA score ¿= 2.5, ensuring that conclusions about performance are based on the most robust evidence.

Data Synthesis

Due to the mixed nature of the data (objective performance metrics vs. subjective perceptions of developers), a hybrid synthesis strategy was adopted:

Descriptive statistics and bibliometric analysis (using VOSviewer) were applied to map the temporal evolution of key terms (Keywords Co-occurrence), which allowed us to identify the "Specialization and Modernization" phase (2023-2024), dominated by Kotlin and Testability.

To address the benefits and limitations (RQ2, RQ3), Inductive Thematic Analysis was used. Textual excerpts from the "Discussion" and "Conclusions" sections of the primary studies were coded and grouped into emerging categories such as "Cognitive Complexity," "Boilerplate Overhead," and "Decoupling."

Finally, quantitative findings (e.g., memory consumption metrics) were contrasted with qualitative evidence to explain contradictions identified in the literature (e.g., the performance vs. maintainability paradox), thereby generating an integrative narrative of the state of the art.

4 Results

A Taxonomy of Academic and Practical Intentions

The objectives of the studies analyzed can be classified into four main categories, reflecting different research approaches in mobile software architecture.

The first and most frequent category is empirical benchmarking. Numerous studies aim to systematically compare the performance, testability, and maintainability of different architectural patterns. Paradigmatic examples include comparisons of MVC, MVVM, and MVI in iOS [19], evaluations of testability and maintainability between MVC and MVVM in GUI desktop applications [20], and the comparison of MVP, MVI, and MVVM within the context of Clean Architecture [21]. These works seek to generate objective evidence to inform architecture selection.

The second category comprises studies of applicability and demonstration of benefits. Their objective is not to compare, but to validate the feasibility and advantages of applying a specific pattern (predominantly MVVM) in a specific domain. This can be seen in the implementation of MVVM in academic systems [22],

the development of management applications with this architecture [23], and the integration of advanced techniques such as dependency injection within MVVM [24]. These studies are more practical and case-oriented.

The third category focuses on developing innovative products, frameworks, or methods. Here, the architecture (usually MVVM) is a central component of the artifact built. Examples include the creation of a framework for museum applications [25], the development of a web system for real-time 3D object manipulation [26], and the proposal of a specific architecture that combines MVVM with Data Binding for Android [26]. The objective transcends evaluation to focus on construction and proposal.

Finally, a fourth category corresponds to teaching and disseminating knowledge. It includes books, chapters, and courses whose primary purpose is to educate developers on selecting and implementing modern architectural patterns. Works such as Advanced Android App Architecture (Advanced Android App Architecture, s. f.) and Building Xamarin.Forms Mobile Apps Using XAML [27] exemplify this educational objective. Together, these categories reveal a field of research with a dual focus: generating rigorous evidence and transmitting practical knowledge to industry.

The Dominant Technological Ecosystem

The technological ecosystem reveals clear stratification and specialization by platform. For Android development, Android Studio has established itself as the virtually universal integrated development environment (IDE) [21, 28, 23]. At the same time, Xcode maintains its hegemony in the iOS ecosystem [29, 19]. In the field of cross-platform development with .NET, Visual Studio is the central tool [24, 30].

One of the most significant trends identified is the linguistic transition in native development. Kotlin has definitively surpassed Java as the language of choice and object of study for modern Android, being promoted for its conciseness, zero security, and interoperability [31, 32, 16]. In iOS, Swift?and particularly its declarative framework SwiftUI?is establishing itself as the contemporary standard [29, 19]. For .NET solutions, C# remains the fundamental pillar [27, 24].

In terms of architectural support frameworks and libraries, there is a clear correlation with the recommendations of the main providers. Android Jetpack Components (ViewModel, LiveData, Room, Navigation) constitute the canonical basis for implementing MVVM on Android, demonstrating strong adoption driven by Google [33, 34]. On the web front-end, another area of focus in this review, Vue.js emerges as a popular framework that inherently implements the MVVM pattern [26, 35].

Finally, the analysis highlights a set of specialized tools for quality assessment. The Android Profiler is instrumental for quantitative performance measurement (CPU and memory) [36, 37]. For static code analysis, SonarQube is a recurring tool [16]. The importance of testability is reflected in the widespread use of frameworks such as XCTest (iOS) and JUnit (Android), as well as advanced mocking libraries such as Mockative and MockK [19]. Together, these tools form a mature technological ecosystem that supports and, in many cases, drives the adoption of modern, decoupled architectures. To complement the qualitative analysis and validate the observed trends, a bibliometric analysis of keywords in the corpus of studies was conducted. Using VOSviewer, network visualization, temporal overlap, and density maps, the conceptual structure and evolution of the field of research on mobile architectures were revealed.

Figure 2 shows the conceptual core of the research, strongly interconnected around the triad android app (viewmodel) pattern. This core is directly linked to the quality attributes maintainability and testability, confirming that these are the primary evaluative dimensions in the literature, above others such as pure performance or user experience. Terms such as framework, mobile device, and user appear on the periphery of the network, indicating their contextual but not central role in the current academic discussion.

The temporal evolution of research interests is evident in Figure 3 (Overlay Visualization). Here, colour (from blue to yellow) represents the average year of prominence for each term. A clear transition can be observed:

Establishment phase (2020-2022): Focused on fundamental concepts such as Android app, pattern, MVC, and MVP (blue tones).

Specialization and modernization phase (2023-2024): Dominated by the emergence and centrality of Kotlin (bright yellow) as the hegemonic language, along with a growing focus on testability. Methodological topics, such as stack overflow analysis and specific architectural aspects, including routers (navigation), also emerge, reflecting the influence of single-activity application paradigms and the search for empirical data in community sources.

Finally, Figure 4 identifies the "hot spots" of research concentration. The area of maximum density (yellow) is located precisely at the intersection of Android app, viewmodel, pattern, and testability. This graphical finding corroborates that the epicenter of contemporary research is the evaluation of the testability of the Model-View-ViewModel pattern in the Android ecosystem. Areas of medium density (green) surround

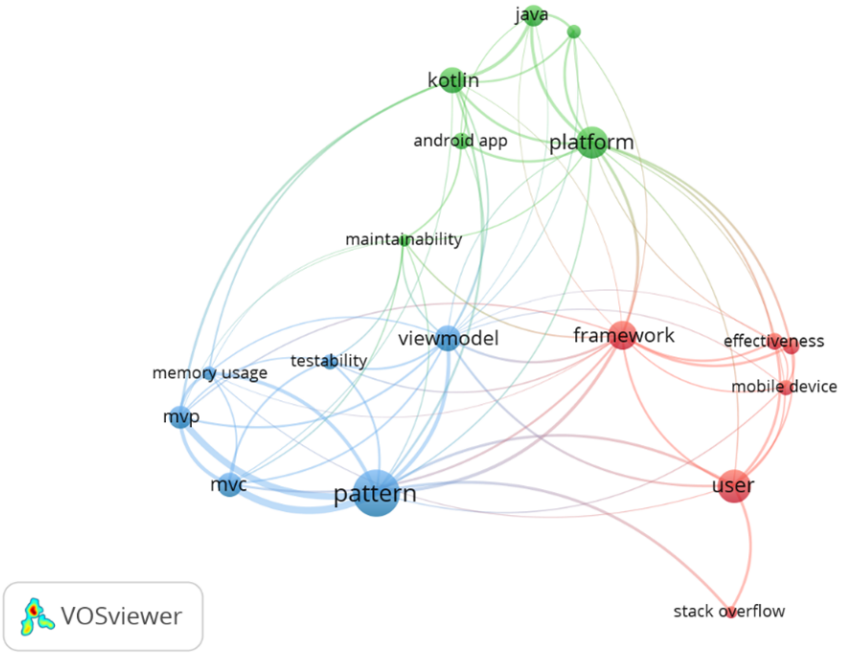

Figure 2: Keyword Co-occurrence Network of MVVM-Related Studies in Mobile Application Development

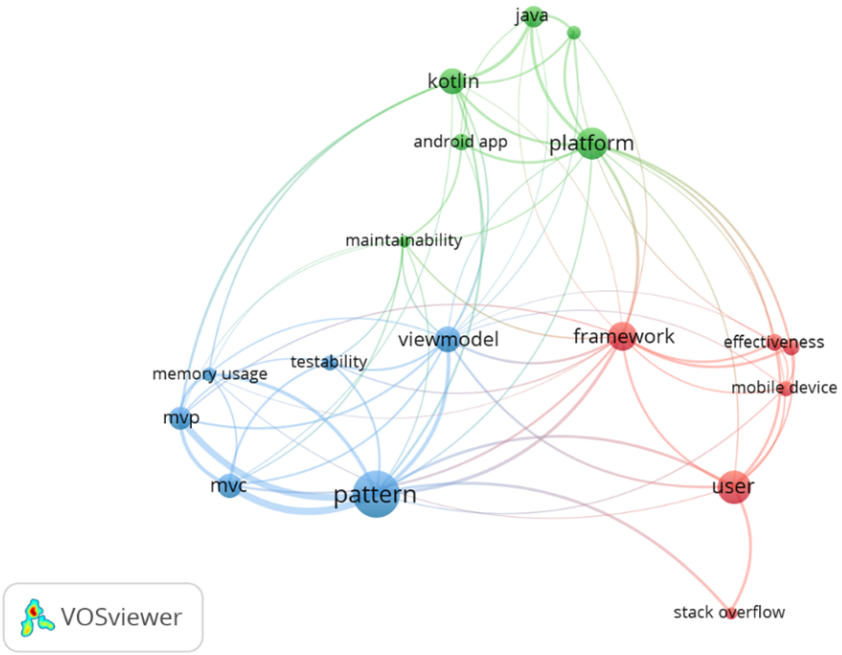

Figure 3: The temporal evolution of research interests

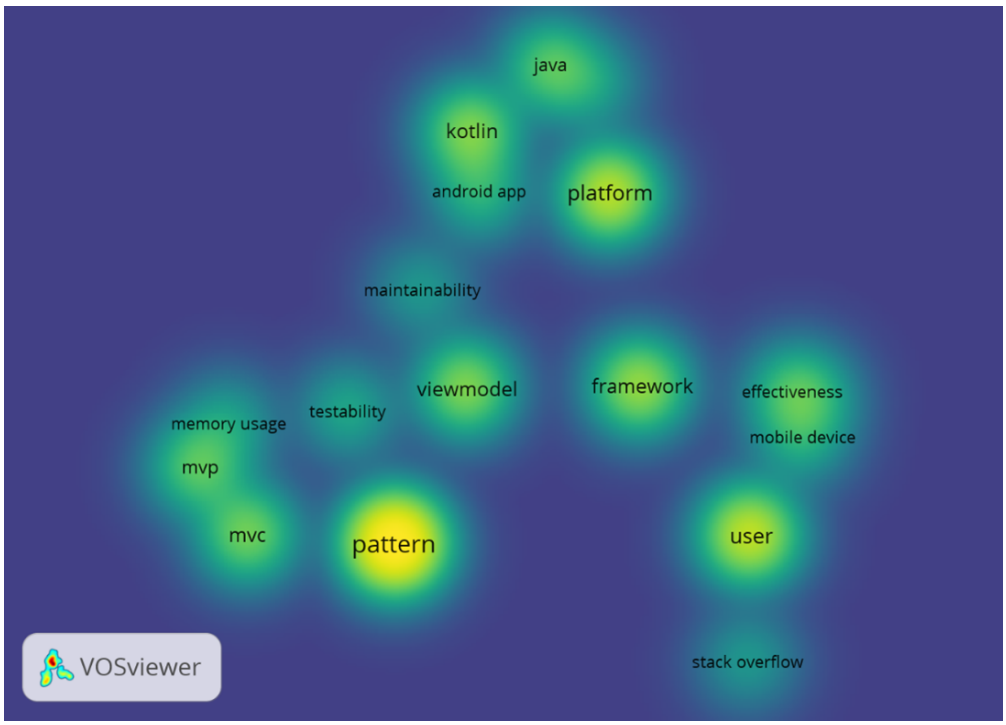

Figure 4: Density Visualization

kotlin/maintainability and the cluster of classic patterns (MVC/MVP/user), while concepts such as platform, message, or framework have low density, indicating that they are less explored topics or of secondary interest.

In summary, this bibliometric analysis visually confirms and quantifies the hegemony of the Android/Kotlin/ViewModel ecosystem, the primacy of testability and maintainability as evaluation criteria, and the evolution of the field from general architectural debates to specialized research using modern empirical methodologies. This conceptual map serves as a frame of reference for the qualitative findings detailed in the following subsections.

Architectural Patterns: The Competitive Landscape

Analysis of the "Comparative Architectures" field reveals a core set of patterns that dominate the comparative literature: Model-View-Controller (MVC), Model-View-Presenter (MVP), and Model-View-ViewModel (MVVM). This triad forms the basis of most experimental studies, with the patterns being compared in various combinations and contexts [38, 20, 39].

Alongside this core, more specialized and modern architectures are emerging. Model-View-Intent (MVI) and VIPER are frequently used in cutting-edge studies, often associated with advantages in testability, unidirectional data flow, and extreme separation of responsibilities, although they are recognized as involving greater initial complexity and a steeper learning curve [19, 40, 31]. On the other hand, Clean Architecture is not presented as an alternative presentation pattern, but as a meta-architecture or set of design principles (such as the Dependency Rule) that can and should be implemented using presentation layer patterns such as MVP, MVVM, or MVI [21, 41, 28].

A specialization by platform is evident in the comparisons. The iOS ecosystem tends to include a wider range, with studies considering MVC, MVP, MVVM, VIPER, and VIP [19]. In contrast, the Android ecosystem, strongly influenced by the official Jetpack libraries, focuses its comparative discussion on MVVM, MVP, MVI, and the application of Clean Architecture.

Consolidated analysis reveals that the most critical and persistent gap is the disconnect between theoretical or laboratory findings and their application, validation, and impact measurement in industrial and large-scale software development environments. A significant gap remains between architectural theory and industrial practice. Many real-world projects still lack a clear structure or rely on 'Obese MVC' patterns despite official guidelines [9, 15]. This suggests that adoption depends not only on technical benefits like testability, but also on team expertise and organizational constraints [8]. Future research should focus on automated tools to help developers maintain architectural standards in large-scale production environments [42, 43].

Quality Attributes

Studies predominantly evaluate software quality attributes, which can be grouped into key dimensions. The performance dimension is the most common in quantitative studies and is broken down into specific metrics: resource consumption, measured mainly through CPU usage (%) and RAM usage (MB) [19, 37]; responsiveness, evaluated through response time or interface latency [37]; and temporal efficiency of processes, measured in execution time (ms) [36].

Testability is a second critical dimension, defined as the ease with which automated tests, particularly unit tests, can be applied to business and presentation logic. This aspect is evaluated using indicators such as code coverage, expressed as the percentage of code executed by a set of tests [19, 20], and testing effort, referring to the modifications necessary in the production code to enable its automated verification.

A third dimension covers the maintainability and structural quality of the code. Here, low coupling and high cohesion resulting from an effective separation of responsibilities are evaluated [20]. Code complexity is another sub-aspect, often approximated by metrics such as the number of lines of code (LOC) or cyclomatic complexity [15, 16]. Finally, the presence of presentation-layer-specific "code smells" (poor design patterns) is a widely used negative indicator of quality [15].

Other dimensions evaluated, albeit less frequently, include human and process factors, such as ease of development and implementation [25, 44], the learning curve associated with a pattern [29], and architectural scalability, i.e., the pattern's ability to accommodate growth in project complexity and size [45]. A clear pattern that emerges is that studies rarely cover all dimensions exhaustively; typically, they focus on a subset, with the performance-testability-maintainability triangle being the most recurrent.

Between Quantitative Objectivity and Qualitative Assessment

Analysis of the metrics reveals a dichotomy between objective quantitative measures, which predominate in experimental studies, and qualitative or subjective assessments.

Among the objective quantitative metrics, those related to system performance stand out: CPU Usage (%), Memory Usage (MB), and Response/Execution Time (ms) are ubiquitous in studies comparing architectures [36, 26, 37]. To evaluate testability, Code Coverage (%) is the most widely adopted indicator [19, 20]. Code quality is often quantified by volume (Lines of Code (LOC)) and the presence of anomalies (Number of Code Smells) [15, 16]. In applications with artificial intelligence components, precision metrics such as Accuracy, Precision, Recall, F1-Score, and AUC-ROC are standard [46, 47].

On the other hand, qualitative or subjective metrics are crucial for evaluating human and perception aspects. Usability is measured using validated scales such as the Mobile App Usability Questionnaire (MAUQ) or the Heuristic Evaluation Severity Rating (scale 0-4) [48, 38]. Developers' perceptions are captured through surveys that ask about the importance of certain "code smells" and the difficulty of implementing them [15]. Finally, many studies resort to descriptive evaluation to judge attributes such as ease of maintenance, clarity, or code modularity [31, 49].

A significant methodological limitation is identified: the lack of standardization in metrics for evaluating complex constructs such as "maintainability" or "ease of development." While performance has universal, directly comparable metrics, maintainability is often described qualitatively or approximated using indirect proxy metrics (such as LOC or coupling), which makes it difficult to aggregate and rigorously compare results across different studies, weakening the ability to perform meta-analyses.

5 Discussion

The review reveals a mature research landscape, but with significant opportunities for empirical evidence, standardization, and adaptation to emerging technologies. The studies present recurring limitations that affect the generalization and practical validity of their findings.

A dominant limitation is validation in controlled environments or with simple prototypes, without testing in industrial-scale projects or with real users [50, 21, 22, 25, 51, 52, 53, 54, 37, 55]. This restriction is also evident in studies that use a single example application [36, 44] or a few devices for testing [19].

There is a lack of extensive empirical validation; numerous studies, particularly those of a conceptual, technical, or formative nature, lack quantitative empirical evaluation. They do not present comparative metrics of performance, maintainability, usability, or impact on productivity [31, 56, 28, 21, 45, 40, 57, 58, 59, 27, 30, 60, 34, 41, 61, 62, 63, 64, 65, 32, 66, 67]. This shortcoming reduces the ability to extrapolate recommendations to real development contexts.

The technological and demographic scope is limited; many studies are restricted to specific technologies, platforms, or populations, which limits their general applicability. Some focus on a single language or

framework [29, 68, 69], while others use geographically or temporally limited datasets [70, 71, 42, 47]. Technological obsolescence is an explicit limitation in studies based on older versions of tools [31, 56, 15].

In terms of partial metrics, evaluations tend to focus on a limited subset of quality metrics. It is common to prioritize performance (CPU, memory) or testability, while leaving aside aspects such as energy consumption, user experience, security, scalability, or long-term maintenance costs [21, 49, 19, 37, 20].

Critical Aspects Not Addressed

Existing research leaves key subject areas unaddressed that are fundamental to a holistic understanding of the impact of mobile architectures.

Real and Longitudinal Impact Assessment: There is a significant gap in measuring the practical impact of architectures beyond the laboratory. Few studies address their effectiveness in real clinical workflows [48], their integration into industrial development pipelines [71, 72], their influence on the productivity of large teams, or their long-term effect on technical debt and maintenance costs [42, 41]. Energy consumption, a critical factor in mobile devices, is also not evaluated [21, 37].

Integration with Emerging Technologies and Paradigms: There is a disconnect between architectural research and current technological trends. Many studies do not consider the impact of Jetpack Compose, Kotlin Flow, Coroutines, Hilt, or declarative development with SwiftUI on the architectures evaluated [31, 56, 21, 15, 29]. The transition to Jetpack Compose requires a shift in how we apply MVVM. Because declarative UIs are a direct function of state, Model-View-Intent (MVI) has emerged as a strong alternative [29]. Unlike MVVM, MVI uses a Unidirectional Data Flow (UDF) to centralize state management [21]. Although MVI is more complex to implement initially, it offers better predictability and testability in modern reactive environments using Kotlin Flows [19, 40]. Likewise, integration with artificial intelligence, machine learning, IoT, or cloud computing is not sufficiently explored [25, 44, 73, 67].

Human and Organizational Perspectives: Research tends to be technically focused, neglecting human and organizational dimensions. Studies on the learning curve of different architectures [19], developer experience, end-user usability [51, 38, 64, 16], adoption in institutions with limited resources [25, 74], and economic and cost-effectiveness aspects are scarce [47].

Comprehensive Comparisons and Measurement Standards: There is no consensus or standard set of metrics and methodologies for comparing architectures fairly and comprehensively. There is a lack of studies comparing a broad spectrum of patterns (MVC, MVP, MVVM, MVI, VIPER, Clean Architecture) under the same conditions and using multidimensional metrics (performance, testability, maintainability, complexity, development effort) [2, 29, 49, 20].

Proposals for Future Research

The authors' proposed future research agendas converge in several main directions, indicating paths to be explored.

Expanding Empirical and Validation Scope: The most frequent proposal is to extend validation to more realistic and diverse contexts. This includes testing in production environments [48, 47], with larger user and application samples [21, 71, 37, 16], and in additional application domains [75, 25].

Exploration of Hybrid, Modern, and Comparative Architectures: The authors urge research into architectures beyond traditional ones, including modern patterns such as MVI, VIPER, Clean Architecture, and their combinations [31, 21, 29, 45, 19, 37]. They also suggest studying the impact of new declarative UI paradigms (Compose, SwiftUI) on architectural structure (Advanced Android App Architecture, s. f.; Android Development Essential Training, 2019; Furjan et al., 2025).

Integration with Advanced Technologies and Automation: There is a call to investigate the synergy between mobile architectures and innovations such as AI/ML [76, 47], IoT [73], cloud computing [30] and automation techniques for migration, architectural inference, or code smell detection [77, 71, 42, 43].

Deepening Quality Metrics and Human Factors: It is proposed to enrich the evaluation by incorporating underestimated metrics such as energy consumption, usability, security, and long-term maintenance costs [21, 37, 16]. At the same time, it is suggested that human factors such as developer experience, learning curve, and end-user satisfaction be investigated [70, 29, 78].

Identified Gaps

The gaps identified reinforce and complement the limitations and opportunities already described, highlighting the disconnect between academic theory and industrial practice.

Disconnect between Architecture and Practical and Educational Contexts: There is a lack of research that translates architectural principles into practical guidelines applicable to educational or self-learning

environments [19], and that links technical frameworks (such as MVVM) with the real needs of institutions with limited resources [25]. Likewise, there is a lack of educational systems that integrate collaborative reading, data analysis, and automated assessment [35].

Lack of Solid, Comparative Empirical Evidence: There is recurring criticism of the lack of robust quantitative studies that compare architectures in real projects, measure production performance, or validate the impact on productivity and maintainability [21, 22, 28, 49, 60, 34, 37]. This gap is especially notable in specific ecosystems such as C#/.NET/WPF [20] and in comparisons between native and cross-platform approaches [16].

Lack of Synchronization with the Modern Technology Ecosystem: The obsolescence of many studies in light of the current Android (Kotlin, Compose, Coroutines) [15, 79] and iOS is highlighted. The absence of AI integration for personalization [44] and the lack of practical validation of models in security [46, 80] and health [81] are also noted.

Need for Support Tools and Methods: Gaps are identified in the development of automated tools for architectural inference [42], detection of missing information in forums [77], mapping of cross-platform libraries [82], and evaluation of the real impact of multi-platform technologies [27].

6 Conclusions

There is notable consensus in the literature regarding the superiority of modern patterns, especially MVVM and MVI, in two key dimensions. First, in testability, multiple studies find that MVVM and MVI consistently outperform MVC and, often, MVP in code coverage and ease of writing unit tests. This is attributed to the clear separation of presentation logic (contained in ViewModel/Presenter/Intent) from the view, which allows it to be isolated and tested without dependencies on the user interface [19, 20]. Second, in maintainability and separation of responsibilities, MVVM is consistently associated with code that is more decoupled, modular, and easier to extend and correct in the long term [21, 44, 66].

However, the picture is ambiguous and contradictory regarding performance. There is no unanimous verdict, suggesting that the impact on performance is strongly mediated by the specific implementation, the application's complexity, and the metric considered. Some studies report that MVVM can introduce significant overhead in CPU and memory usage compared to MVC, attributable to the additional infrastructure required for data binding and state observation [36]. Others, however, find that more modern architectures such as MVI can offer better performance than MVC or MVVM, possibly due to more efficient and predictable state management. A third group of results indicates insignificant or mixed differences, where one pattern wins on one metric (e.g., CPU) but loses on another (e.g., response time), as seen in the direct comparison between MVVM and MVP [37].

Regarding specific patterns, the literature highlights that MVI excels at exceptional testability and strictly unidirectional data flow, which favor debugging and predictability, at the cost of greater implementation complexity. MVP is positioned in some studies as a classic balance, sometimes showing better response times than MVVM with a simpler structure than MVI [21, 37]. For its part, traditional MVC is the subject of recurring criticism for its tendency to degenerate into "Massive View Controllers" with low testability and high coupling, being considered suitable mainly for applications of trivial complexity [15].

The overall conclusion that emerges from this synthesis is that there is no universal optimal architecture. Empirical evidence points to the existence of inherent trade-offs. MVVM emerges as the predominant balanced recommendation for most medium-scale projects, offering a favorable combination of testability, maintainability, and support for modern tools, even though it may incur a resource overhead in certain scenarios [19]. The final decision should be based on a contextual analysis that prioritizes the quality attributes most critical to the specific project: maximum performance (favoring lighter options), long-term maintainability and testability (favoring MVVM/MVI), or scalability in large-scale projects (considering multi-layer architectures such as Clean Architecture or VIPER).

Research is evolving from basic comparisons (MVC vs. MVVM) to more sophisticated analyses that include MVI, Clean Architecture, and the impact of declarative frameworks. However, empirical validation has not advanced at the same pace as conceptual exploration.

While studies such as "A Comparison of Architectural Patterns for Testability and Performance Quality for iOS Mobile Applications Development" [19] find that MVI offers the best testability and performance, others, such as "Comparative Analysis of MVVM and MVP Patterns Performance on Android Dashboard System" [37], show that MVP can have better response times. This suggests that the "optimal" architecture critically depends on the prioritized metrics and the application's context, highlighting the need for multifactorial studies.

Future proposals clearly point to a convergence of research in mobile architectures with domains such as artificial intelligence, digital health (mHealth), education, cybersecurity, and IoT. Architecture is no longer

studied in isolation, but as a key enabler for complex applications in these domains.

Consolidated analysis reveals that the most critical and persistent gap is the disconnect between theoretical or laboratory findings and their application, validation, and impact measurement in industrial and large-scale software development environments. Closing this gap requires academia-industry collaboration and research methods that can capture the complexity of real-world projects.

7 Acknowledgment

7.1 Funding

This research was supported by the Universidad Autnoma de Bucaramanga under the internal research grant: "Teaching programming in the AI era: design and evaluation of pedagogical strategies based on Pair Programming and Learning Analytics in university education" (Pure ID: 36105068).

7.2 Competing Interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

7.3 Credits

Fabian-Enrique Suarez-Carvajal: Conceptualization, Methodology, Investigation, Validation, Data curation, Writing, original draft, Writing review & editing, Project administration. Rene Alejandro Lobo Quintero: Methodology, Validation, Data curation, Writing review & editing. Julian Santiago Diaz Santoyo: Methodology, Validation, Data curation, Writing review & editing. Yamid Grabriel Gamba Gonzalez: Methodology, Validation, Data curation, Writing review & editing. Javier Pinzon Cantellanos: Methodology, Investigation, Writing review & editing. Declaration of Generative AI and AI-assisted Technologies in the Writing Process Declaration of Generative AI and AI-assisted Technologies in the Writing Process During the preparation of this work, the authors utilised Grammarly in order to check for grammatical errors and improve sentence structure. Subsequent to the application of this tool/service, the authors reviewed and edited the content as necessary and accept full responsibility for the final content of the publication.

References

[1] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," vol. 2, Jan. 2007.

[2] M. Fuksa, S. Speth, and S. Becker, "MVVM Revisited: Exploring Design Variants of the Model-View-ViewModel Pattern," 2025, vol. 15409, pp. 163–181, arXiv:2504.18191 [cs]. [Online]. Available: https://doi.org/10.1007/978-3-031-78338-8_9

[3] N. C. Ramachandrappa, "MVVM Design Pattern in Software Development," *International Journal of Computer Trends and Technology*, vol. 72, no. 9, pp. 114–119, Sep. 2024. [Online]. Available: https://doi.org/10.14445/22312803/IJCTT-V72I9P117

[4] P. S. P. Akarte, "Evaluation of Cross-Platform Technology Flutter Using Different System Environments," *International Journal for Research in Applied Science and Engineering Technology*, vol. 12, no. 4, pp. 1330–1335, Apr. 2024. [Online]. Available: https://doi.org/10.22214/ijraset.2024.60066

[5] W. Sun, H. Chen, and W. Yu, "The Exploration and Practice of MVVM Pattern on Android Platform," Jan. 2016.

[6] D. Dobrean and L. Diosan, "A Comparative Study of Software Architectures in Mobile Applications," *Studia Universitatis Babe-Bolyai Informatica*, vol. 64, pp. 49–64, Dec. 2019.

[7] M. Martinez, "Two Datasets of Questions and Answers for Studying the Development of Cross-Platform Mobile Applications using Xamarin Framework," in *2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, May 2019, pp. 162–173. [Online]. Available: https://doi.org/10.1109/MOBILESoft.2019.00032

[8] A. Ahmad, K. Li, C. Feng, S. M. Asim, A. Yousif, and S. Ge, "An Empirical Study of Investigating Mobile Applications Development Challenges," *IEEE Access*, vol. 6, pp. 17 711–17 728, 2018. [Online]. Available: https://doi.org/10.1109/ACCESS.2018.2818724

[9] A. Daoudi, N. Moha, G. ElBoussaidi, and S. Kpodjedo, "An exploratory study of MVC-based architectural patterns in android apps," in *Proc ACM Symp Appl Computing*, vol. Part F147772. Association for Computing Machinery, 2019, pp. 1711–1720, journal Abbreviation: Proc ACM Symp Appl Computing.

[10] R. Solingen and E. Berghout, "The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development," Jan. 1999.

[11] A. Birn-Hansen, T.-M. Grnli, and G. Ghinea, "A Survey and Taxonomy of Core Concepts and Research Challenges in Cross-Platform Mobile Development," *ACM Comput. Surv.*, vol. 51, no. 5, pp. 108:1–108:34, Nov. 2018. [Online]. Available: https://doi.org/10.1145/3241739

[12] R. Rua and J. Saraiva, "A large-scale empirical study on mobile performance: energy, run-time and memory," *Empirical Software Engineering*, vol. 29, no. 1, p. 31, Dec. 2023. [Online]. Available: https://doi.org/10.1007/s10664-023-10391-y

[13] D. Li, S. Hao, J. Gui, and W. G. Halfond, "An Empirical Study of the Energy Consumption of Android Applications," in *2014 IEEE International Conference on Software Maintenance and Evolution*, Sep. 2014, pp. 121–130, iSSN: 1063-6773. [Online]. Available: https://doi.org/10.1109/ICSME.2014.34

[14] F. Palomba, G. Bavota, M. D. Penta, F. Fasano, R. Oliveto, and A. D. Lucia, "On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1188–1221, Jun. 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9535-z

[15] S. G. Carvalho, M. Aniche, J. Verssimo, R. S. Durelli, and M. A. Gerosa, "An empirical catalog of code smells for the presentation layer of Android apps," *Empirical Software Engineering*, vol. 24, no. 6, pp. 3546–3586, Dec. 2019. [Online]. Available: https://doi.org/10.1007/s10664-019-09768-9

[16] I. S. R. Ugli, J. Cheon, and G. Woo, "Code Smells and Development Efficiency in Native and Cross-Platform Mobile Applications: A Study of Java, Kotlin, and Flutter," *Journal of Information Processing Systems*, vol. 21, no. 4, pp. 401–412, Aug. 2025. [Online]. Available: https://jips-k.org/digital-library/2025/21/4/401

[17] L. Cruz and R. Abreu, "Catalog of energy patterns for mobile applications," *Empirical Software Engineering*, vol. 24, no. 4, pp. 2209–2235, Aug. 2019. [Online]. Available: https://doi.org/10.1007/s10664-019-09682-0

[18] J. R. Landis and G. G. Koch, "The Measurement of Observer Agreement for Categorical Data," *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977, publisher: International Biometric Society. [Online]. Available: https://doi.org/10.2307/2529310

[19] H. Magics-Verkman, D. R. Zmaranda, C. A. Győrödi, and R.-Ş. Győrödi, "A Comparison of Architectural Patterns for Testability and Performance Quality for iOS Mobile Applications Development," in *2023 17th International Conference on Engineering of Modern Electric Systems (EMES)*, Jun. 2023, pp. 1–4. [Online]. Available: https://doi.org/10.1109/EMES58375.2023.10171619

[20] A. Wilson, F. Wedyan, and S. Omari, "An Empirical Evaluation and Comparison of the Impact of MVVM and MVC GUI Driven Application Architectures on Maintainability and Testability," in *2022 International Conference on Intelligent Data Science Technologies and Applications (IDSTA)*, Sep. 2022, pp. 101–108. [Online]. Available: https://doi.org/10.1109/IDSTA55301.2022.9923083

[21] F. F. Anhar, M. H. P. Swari, and F. P. Aditiawan, "Analisis Perbandingan Implementasi Clean Architecture Menggunakan MVP, MVI, Dan MVVM Pada Pengembangan Aplikasi Android Native," *Jupiter: Publikasi Ilmu Keteknikan Industri, Teknik Elektro dan Informatika*, vol. 2, no. 2, pp. 181–191, Jan. 2024. [Online]. Available: https://doi.org/10.61132/jupiter.v2i2.155

[22] M. S. Arif, A. Musthafa, and D. Muriyatmoko, "Implementation of Model-View-ViewModel (MVVM) Architecture Pattern in the Sistem Informasi Akademik UNIDA Gontor Mobile Application," *Proceeding International Conference on Science and Engineering*, vol. 3, pp. 283–289, Apr. 2020. [Online]. Available: https://doi.org/10.14421/icse.v3.514

[23] M. K. Islam, S. J. Boshra, M. Rahman, M. M. Islam Jony, and I. A. Rahat, "Android Based Smart Appointment System (SAS) for Booking and Interacting with Teacher for Counselling," in *2023 International Conference on Emerging Smart Computing and Informatics (ESCI)*, Mar. 2023, pp. 1–5. [Online]. Available: https://doi.org/10.1109/ESCI56872.2023.10099495

[24] S. Hoefling, "Dependency Injection and Logging," in *Getting Started with the Uno Platform and WinUI 3: Hands-On Building of Cross-Platform Desktop, Mobile, and Web Applications That Can Run Anywhere*, S. Hoefling, Ed. Berkeley, CA: Apress, 2022, pp. 207–238. [Online]. Available: https://doi.org/10.1007/978-1-4842-8248-9_9

[25] T. Baumgrtner, *A Software Framework for Mobile Apps in the Museum Application Domain*, ser. Gabler Theses. Wiesbaden: Springer Fachmedien, 2024. [Online]. Available: https://doi.org/10.1007/978-3-658-44367-2

[26] N. Kentaro and F. Satoshi, "Browser-Based Manipulation of Virtual Objects Through MVVM Architecture with Data Binding," in *2021 Ninth International Symposium on Computing and Networking (CANDAR)*, Nov. 2021, pp. 134–140, iSSN: 2379-1896. [Online]. Available: https://doi.org/10.1109/CANDAR53791.2021.00026

[27] D. Hermes and N. Mazloumi, *Building Xamarin.Forms Mobile Apps Using XAML: Mobile Cross-Platform XAML and Xamarin.Forms Fundamentals*. Berkeley, CA: Apress, 2019. [Online]. Available: https://doi.org/10.1007/978-1-4842-4030-4

[28] "Clean Android Architecture | Programming | Paperback." [Online]. Available: https://www.packtpub.com/en-ca/product/clean-android-architecture-9781803234588?type=print

[29] K. Furjan, F. Kisi, and D. Bele, "Declarative iOS programming and architectural patterns," in *Security Issues in Communication Devices, Networks and Computing Models*. CRC Press, 2025, num Pages: 10.

[30] S. Hoefling, *Getting Started with the Uno Platform and WinUI 3: Hands-On Building of Cross-Platform Desktop, Mobile, and Web Applications That Can Run Anywhere*. Berkeley, CA: Apress, 2022. [Online]. Available: https://doi.org/10.1007/978-1-4842-8248-9

[31] "Advanced Android App Architecture: Real-world app architecture in Kotlin 1.3." [Online]. Available: https://dokumen.pub/advanced-android-app-architecture-real-world-app-architecture-in-kotlin-13.html

[32] A. Patel, K. Kumar, and D. B. Kumar Pandey, "Object-Oriented Programming with Kotlin," in *Kotlin Mastery: A Comprehensive Guide for Java Developers and New Programmers*, A. Patel, K. Kumar, and D. B. Kumar Pandey, Eds. Berkeley, CA: Apress, 2025, pp. 73–126. [Online]. Available: https://doi.org/10.1007/979-8-8688-1618-5_4

[33] M. Fuksa, S. Speth, and S. Becker, "MVVM Revisited: Exploring Design Variants oftheModel-View-ViewModel Pattern," in *Enterprise Design, Operations, and Computing*, J. Borbinha, T. Prince Sales, M. M. Da Silva, H. A. Proper, and M. Schnellmann, Eds. Cham: Springer Nature Switzerland, 2025, pp. 163–181.

[34] P. Kaczmarek and Z. Piotrowski, "The Architecture of the Mobile Application for Android with the Use of the MVVM Pattern and the HILT, Livedata, Room, Retrofit and Kotlin Coroutines Libraries," in *Commun. Comput. Info. Sci.*, vol. 2101 CCIS. Springer Science and Business Media Deutschland GmbH, 2024, pp. 296–307, journal Abbreviation: Commun. Comput. Info. Sci.

[35] L. Liu, H. Cao, B. Lin, and L. Zhang, "A System Framework to Support the Collaborative Reading for Students," in *2021 IEEE International Conference on Engineering, Technology & Education (TALE)*, Dec. 2021, pp. 1–3, iSSN: 2470-6698. [Online]. Available: https://doi.org/10.1109/TALE52509.2021.9678828

[36] H. Epiloksa, D. Kusumo, and M. Adrian, "Effect Of MVVM Architecture Pattern on Android Based Application Performance," *JURNAL MEDIA INFORMATIKA BUDIDARMA*, vol. 6, p. 1949, Oct. 2022.

[37] F. Pradana, R. I. Langundi, D. Pramono, and N. I. Iriani, "Comparative Analysis of MVVM and MVP Patterns Performance on Android Dashboard System," *Jurnal Nasional Teknik Elektro dan Teknologi Informasi*, vol. 14, no. 2, pp. 87–95, May 2025. [Online]. Available: https://doi.org/10.22146/jnteti.v14i2.18985

[38] M. I. S. B. Khairat, Y. Priyadi, and M. Adrian, "Usability Measurement in User Interface Design Using Heuristic Evaluation & Severity Rating (Case Study: Mobile TA Application based on MVVM)," in *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, Jan. 2022, pp. 0974–0979. [Online]. Available: https://doi.org/10.1109/CCWC54503.2022.9720876

[39] A. Yusuf, R. Fitri, A. S. B. Nugroho, and A. Rozaq, "Evaluating the Performance of Architectural Patterns in Web Application Development," in *2024 Ninth International Conference on Informatics and Computing (ICIC)*, Oct. 2024, pp. 1–6. [Online]. Available: https://doi.org/10.1109/ICIC64337.2024.10957712

[40] R. F. Garca, "Other Architecture Patterns," in *iOS Architecture Patterns: MVC, MVP, MVVM, VIPER, and VIP in Swift*, R. F. Garca, Ed. Berkeley, CA: Apress, 2023, pp. 365–384. [Online]. Available: https://doi.org/10.1007/978-1-4842-9069-9_7

[41] K. Lano and S. Yassipour Tehrani, "Mobile Application Architectures," in *Introduction to Software Architecture: Innovative Design using Clean Architecture and Model-Driven Engineering*, K. Lano and S. Yassipour Tehrani, Eds. Cham: Springer Nature Switzerland, 2023, pp. 99–136. [Online]. Available: https://doi.org/10.1007/978-3-031-44143-1_6

[42] D. Dobrean and L. Dioan, "Validating HyDe: Intelligent Method for Inferring Software Architectures fromMobile Codebase," in *Evaluation of Novel Approaches to Software Engineering*, R. Ali, H. Kaindl, and L. A. Maciaszek, Eds. Cham: Springer International Publishing, 2022, pp. 3–28.

[43] L. Wijerathna, A. Aleti, T. Bi, and A. Tang, "Mining and relating design contexts and design patterns from Stack Overflow," *Empirical Software Engineering*, vol. 27, no. 1, p. 8, Oct. 2021. [Online]. Available: https://doi.org/10.1007/s10664-021-10034-0

[44] S. Kamdi and A. Liu, "A study of the development framework followed to integrate the Shopify SDK in an iOS app, built using SwiftUI," in *2025 International Conference on Software, Knowledge, Information Management & Applications (SKIMA)*, Jun. 2025, pp. 1–7. [Online]. Available: https://doi.org/10.1109/SKIMA66621.2025.11155241

[45] R. F. Garca, *iOS Architecture Patterns: MVC, MVP, MVVM, VIPER, and VIP in Swift*. Berkeley, CA: Apress, 2023. [Online]. Available: https://doi.org/10.1007/978-1-4842-9069-9

[46] K. Ma, B. Lu, S. Yin, C. Zheng, and H. Zhu, "CADroid: A cross-combination attention based framework for android malware detection," *Expert Systems with Applications*, vol. 297, p. 129446, Feb. 2026. [Online]. Available: https://doi.org/10.1016/j.eswa.2025.129446

[47] Y. Yue, X. Zeng, H. Lin, J. Xu, F. Zhang, K. Zhou, L. Li, and Z. Li, "A deep learning based smartphone application for early detection of nasopharyngeal carcinoma using endoscopic images," *npj Digital Medicine*, vol. 7, no. 1, p. 384, Dec. 2024. [Online]. Available: https://doi.org/10.1038/s41746-024-01403-2

[48] E. Asadiara, H. Emami, M. Safari, and R. Rabiei, "Development and usability evaluation of a mHealth application for prehospital triage," *Scientific Reports*, vol. 15, no. 1, p. 21576, Jul. 2025, publisher: Nature Publishing Group. [Online]. Available: https://doi.org/10.1038/s41598-025-06952-4

[49] R. F. Garca, "MVVM: ModelViewViewModel," in *iOS Architecture Patterns: MVC, MVP, MVVM, VIPER, and VIP in Swift*, R. F. Garca, Ed. Berkeley, CA: Apress, 2023, pp. 145–224. [Online]. Available: https://doi.org/10.1007/978-1-4842-9069-9_4

[50] K. D. Abrahamsen and J. Mauro, "Mocking Classes inKotlin withMockative," in *Software Engineering and Advanced Applications*, D. Taibi and D. Smite, Eds. Cham: Springer Nature Switzerland, 2026, pp. 23–31.

[51] H. A. alikan and H. Osmanolu, "GitHubShare: A Mobile Application for Sharing Software Projects Via GitHub," in *Computing, Internet of Things and Data Analytics*, F. P. Garca Mrquez, A. Jamil, A. A. Hameed, H. Wang, Y. Zhang, and J. Yang, Eds. Cham: Springer Nature Switzerland, 2025, pp. 659–670.

[52] E. T. Ewane and M. Mazzara, "Development of a tool for gaining relevant skills for a solid career in native Android development, focusing on todays industry," in *2021 International Conference "Nonlinearity, Information and Robotics" (NIR)*, Aug. 2021, pp. 1–6. [Online]. Available: https://doi.org/10.1109/NIR52917.2021.9665811

[53] X. He, C. Liang, M. Shi, B. Shi, Y. Zhao, L. Yao, and J. Yang, "Design and Implementation of a Software Architecture for Portable Recording Equipment for Power Supply Service," in *Internet of Things ICIOT 2025*, B. Tekinerdogan, H. A. Hamid, H. Li, and L.-J. Zhang, Eds. Cham: Springer Nature Switzerland, 2026, pp. 1–14.

[54] M. Maghalian, S. Veisi, M. Mirghafourvand, M. Rezaei, T. Samad-Soltani, A. Naghizadeh, and S. Ghanbari-Homaie, "Developing and testing a mobile application for imparting knowledge: the positive birth experience guideline to midwifery students," *BMC Medical Education*, vol. 25, no. 1, p. 577, Apr. 2025. [Online]. Available: https://doi.org/10.1186/s12909-025-07147-1

[55] A. Pratama and F. Ardiani, "Optimization of Model-View-ViewModel (MVVM) Architecture Pattern and RESTfull API on Android-based E-Learning Application," *International Journal of Computer Applications*, vol. 185, pp. 4–11, Nov. 2023.

[56] *Android Development Essential Training: App Architecture with Kotlin.* linkedin.com, 2019, google-Books-ID: 52ARzgEACAAJ.

[57] L. Guo, "Explore New Language: A Quick Introduction to Kotlin," in *The First Line of Code: Android Programming with Kotlin*, L. Guo, Ed. Singapore: Springer Nature, 2022, pp. 33–86. [Online]. Available: https://doi.org/10.1007/978-981-19-1800-1_2

[58] ——, "Real Project Practice: Creating a Weather App," in *The First Line of Code: Android Programming with Kotlin*, L. Guo, Ed. Singapore: Springer Nature, 2022, pp. 641–709. [Online]. Available: https://doi.org/10.1007/978-981-19-1800-1_15

[59] T. Herceg, "Migration of ASP.NET MVC and Web Pages," in *Modernizing .NET Web Applications: Everything You Need to Know About Migrating ASP.NET Web Applications to the Latest Version of .NET*, T. Herceg, Ed. Berkeley, CA: Apress, 2024, pp. 569–616. [Online]. Available: https://doi.org/10.1007/979-8-8688-0617-9_9

[60] P. Johnson, *Using MVVM Light with your Xamarin Apps.* Berkeley, CA: Apress, 2018. [Online]. Available: https://doi.org/10.1007/978-1-4842-2475-5

[61] S. Lawrence, "Advanced UI Concepts," in *Introducing .NET MAUI: Build and Deploy Cross-Platform Applications Using C# and .NET 9.0 Multi-Platform App UI*, S. Lawrence, Ed. Berkeley, CA: Apress, 2025, pp. 251–294. [Online]. Available: https://doi.org/10.1007/979-8-8688-1189-0_9

[62] ——, "Testing," in *Introducing .NET MAUI: Build and Deploy Cross-Platform Applications Using C# and .NET 9.0 Multi-Platform App UI*, S. Lawrence, Ed. Berkeley, CA: Apress, 2025, pp. 419–450. [Online]. Available: https://doi.org/10.1007/979-8-8688-1189-0_13

[63] A. Mishra, "The MVVM Architectural Pattern," in *iOS Code Testing: Test-Driven Development and Behavior-Driven Development with Swift*, A. Mishra, Ed. Berkeley, CA: Apress, 2017, pp. 43–60. [Online]. Available: https://doi.org/10.1007/978-1-4842-2689-6_3

[64] A. S. Negi, H. Vaidya, and R. Chauhan, "Design and Development of a LMS with MVVM Architecture," in *2023 International Conference on Sustainable Emerging Innovations in Engineering and Technology (ICSEIET)*, Sep. 2023, pp. 820–823. [Online]. Available: https://doi.org/10.1109/ICSEIET58677.2023.10303333

[65] D. Panjuta and L. Nwokike, *Tiny Android Projects Using Kotlin.* New York: Chapman and Hall/CRC, Feb. 2024.

[66] E. Vennaro, "Model View View-Model (MVVM)," in *iOS Development at Scale: App Architecture and Design Patterns for Mobile Engineers*, E. Vennaro, Ed. Berkeley, CA: Apress, 2023, pp. 277–298. [Online]. Available: https://doi.org/10.1007/978-1-4842-9456-7_8

[67] N. Vermeir, "MAUI," in *Introducing .NET 6: Getting Started with Blazor, MAUI, Windows App SDK, Desktop Development, and Containers*, N. Vermeir, Ed. Berkeley, CA: Apress, 2022, pp. 153–176. [Online]. Available: https://doi.org/10.1007/978-1-4842-7319-7_6

[68] C. Liu, Z. Suo, Q. Mao, and Y. Zhu, "Practice and Application of Wiki Open Source Document Platform Based on Vue," in *2023 3rd International Symposium on Computer Technology and Information Science (ISCTIS)*, Jul. 2023, pp. 819–822. [Online]. Available: https://doi.org/10.1109/ISCTIS58954.2023.10213203

[69] Y. W. Syaifudin, N. Funabiki, and I. Liem, "Comparisons of Student's Self-Learning Performances Using Java and Kotlin Languages in Android Programming Learning Assistance System," in *Conf. Online Teach. Mob. Educ., OT4ME*. Institute of Electrical and Electronics Engineers Inc., 2021, pp. 93–97, journal Abbreviation: Conf. Online Teach. Mob. Educ., OT4ME.

[70] G. Alves, D. Jannach, R. F. de Souza, D. Damian, and M. G. Manzato, "Digitally nudging users to explore off-profile recommendations: here be dragons," *User Modeling and User-Adapted Interaction*, vol. 34, no. 2, pp. 441–481, Apr. 2024. [Online]. Available: https://doi.org/10.1007/s11257-023-09378-7

[71] M. J. de Dieu, P. Liang, M. Shahin, and A. A. Khan, "How do users revise architectural related questions on stack overflow: an empirical study," *Empirical Software Engineering*, vol. 30, no. 6, p. 171, Oct. 2025. [Online]. Available: https://doi.org/10.1007/s10664-025-10719-w

[72] L. G. Marticorena, L. A. Morales, L. Antonelli, G. Rossi, and D. Firmenich, "Development iterations based on web augmentation and context tasks," *Multimedia Tools and Applications*, vol. 82, no. 8, pp. 11 793–11 817, Mar. 2023. [Online]. Available: https://doi.org/10.1007/s11042-022-13694-2

[73] P. M. Mah, I. Skalna, T. Peech-Pilichowski, G. N. Amuzang, M. B. S. Itoe, and N. F. Tah, "Integration of Internet of Things (IoTs) with Wireless Sensor Networks (WSNs) for a Transformative Secure Community Mindset Applied Deep Learning Models and Natural Language Processing Techniques," in *Artificial intelligence and Machine Learning*, K. S. Soliman, Ed. Cham: Springer Nature Switzerland, 2024, pp. 3–19.

[74] G. Dlamini, U. Ahmad, L. R. Kharkrang, and V. Ivanov, "Detecting Design Patterns inAndroid Applications withCodeBERT Embeddings andCK Metrics," in *Analysis of Images, Social Networks and Texts*, D. I. Ignatov, M. Khachay, A. Kutuzov, H. Madoyan, I. Makarov, I. Nikishina, A. Panchenko, M. Panov, P. M. Pardalos, A. V. Savchenko, E. Tsymbalov, E. Tutubalina, and S. Zagoruyko, Eds. Cham: Springer Nature Switzerland, 2024, pp. 267–280.

[75] T. Baumgrtner, "Prototype Development," in *A Software Framework for Mobile Apps in the Museum Application Domain*, T. Baumgrtner, Ed. Wiesbaden: Springer Fachmedien, 2024, pp. 189–266. [Online]. Available: https://doi.org/10.1007/978-3-658-44367-2_7

[76] Y. W. Syaifudin, N. Funabiki, A. B. Kaswar, A. Sunandar, T. Fatmawati, P. Y. Saputra, D. C. Wijaya, and M. F. Irawanto, "Implementing Data Integration Self-Study: A Curriculum Development for Android Programming Learning Assistance System," *SN Computer Science*, vol. 6, no. 6, p. 559, Jun. 2025. [Online]. Available: https://doi.org/10.1007/s42979-025-04114-x

[77] M. N. Alatawi, "Forecasting the software engineering models effort estimation using constructive cost estimation models," *Iran Journal of Computer Science*, vol. 7, no. 4, pp. 735–754, Dec. 2024. [Online]. Available: https://doi.org/10.1007/s42044-024-00194-9

[78] L. Longo and L. M. Barbosa, "Gamified Mobile App for Financial Education: An Innovative Experience in a Brazilian Public University," in *Proceedings of the Future Technologies Conference (FTC) 2025, Volume 2*, K. Arai, Ed. Cham: Springer Nature Switzerland, 2026, pp. 80–96.

[79] D. Schultes, L. Schneider, T. Heymann, and F. Wild, "Native cross-platform app development using the SequalsK transpiler," *Inf. Softw. Technol.*, vol. 179, no. C, Mar. 2025. [Online]. Available: https://doi.org/10.1016/j.infsof.2024.107626

[80] A. Maqsood, H. T. Mirza, F. Iqbal, R. I. Alkanhel, N. Hussain, A. Afzaal, A. Altaf, and I. Ashraf, "Feature Fusion-Based Ensemble Approach for Robust Malware Detection with Reduced False Positives," *Journal of Network and Systems Management*, vol. 34, no. 1, 2026, publisher: Springer.

[81] Z. Li, Y. Huang, Q. Li, Y. Sun, C. Li, J. Wu, H. Zheng, and R. Zeng, "TreeQNet: a webserver for Treatment evaluation with Quantified Network," *BMC Bioinformatics*, vol. 23, no. 1, p. 473, Nov. 2022. [Online]. Available: https://doi.org/10.1186/s12859-022-05024-y

[82] A. A. Muhammad, A. Soliman, H. Zayed, A. H. Yousef, and S. Selim, "Automated library mapping approach based on cross-platform for mobile development programming languages," *Software - Practice and Experience*, vol. 54, no. 5, pp. 683–703, 2024, publisher: John Wiley and Sons Ltd.