Republic of Cameroon

Peace-Work-Fatherland

Ministry of Higher Education

University of Buea

Faculty of Engineering and Technology

Department of Computer Engineering

République du Cameroun

Paix-Travaille-Patrie

Ministère de l'enseignement supérieures

Université de Buea

Faculty of Engineering and Technology

Department of Computer Engineering

Comprehensive Analysis of Mobile Application Development

Course Instructor : Dr Nkemeni Valery

Group Members (Group 21)

1- Angi Atangwa Valerine **FE22A148**

2- Av'Nyehbeuhkonh Ezekiel Hosana **FE22A160**

3- Beng Dze Meinoff **FE22A173**

4- Jong Turkson Junior **FE22A228**

5- Nyuyesemo Calglain **FE22A287**

Table of contents

Abstract

This report provides a comprehensive analysis of mobile app development, covering different types of mobile applications (native, hybrid, and Progressive Web Apps), programming languages, development frameworks, architectures, requirement engineering, and cost estimation. The study compares key aspects such as performance, development time, cost, and user experience across different approaches. Additionally, it explores mobile app architectures, design patterns, and best practices for collecting user requirements. Finally, the report discusses methodologies for estimating development costs, helping stakeholders make informed decisions when planning mobile app projects.

I- Introduction

Mobile applications have become an integral part of modern life, serving various purposes such as communication, entertainment, business, and productivity. With the rapid evolution of mobile technology, developers now have multiple approaches to building apps, each with distinct advantages and trade-offs. This report examines the different types of mobile apps, compares programming languages and frameworks, analyses architectures and design patterns, and explores requirement engineering and cost estimation techniques.

The goal of this study is to provide a structured comparison of mobile development methodologies, enabling developers and businesses to choose the most suitable approach based on project requirements, budget, and target audience.

II- Objectives

The objectives of this report are;

1.To understand mobile app development thereby understanding

i. The various types of mobile apps that exist
ii. The different programming language used for mobile app development,
iii. The different frameworks used for mobile app development,
iv. The different architecture and design patterns that exists,
v. How to collect, analyse and use user reauirements,
vi. How to estimate the development cost of a mobile app.

III- Background of Mobile App Development

III.1 History of Mobile Apps

The evolution of mobile applications can be traced back to the early 2000s with the introduction of smartphones. Key milestones include:

- **2007:** Apple launched the iPhone and the App Store, revolutionizing mobile software distribution.
- **2008:** Google introduced Android and the Google Play Store, fostering competition in the mobile OS market.
- **2010s:** The rise of cross-platform frameworks (React Native, Flutter) and Progressive Web Apps (PWAs) provided alternatives to native development.
- **2020s:** Advancements in AI, 5G, and foldable devices further expanded mobile app capabilities.

III.2 Current Trends

- **Cross-platform development** (Flutter, React Native) gaining popularity.
- **Progressive Web Apps (PWAs)** bridging the gap between web and mobile.

- **AI and Machine Learning integration** in mobile apps.
- **5G enabling faster, more responsive applications.**

IV- Review and comparison of major types of mobile apps.

Mobile apps can be broadly categorized into three main types based on their development approach and functionality: **native apps, web apps, and hybrid apps**. Each has distinct advantages and limitations, depending on factors like performance, development cost, and user experience.

A brief explanation of the

1. Native Apps

- **Definition**: Built specifically for a single platform (iOS or Android) using platform-specific languages (Swift/Objective-C for iOS, Kotlin/Java for Android).

- **Pros:**

  - **High performance**: Optimized for the OS, offering fast and smooth UX.

  - **Full device access**: Can utilize device features (camera, GPS, sensors, etc.).

  - **Better security**: Platform-specific security features.

  - **Offline functionality**: Can work without internet.

  - **App store distribution**: Available on Google Play & Apple App Store.

- **Cons**:

  - **Higher cost**: Requires separate development for each platform.

  - **Longer development time**: Two codebases need maintenance.

  - **Updates require approval**: App store reviews delay updates.

Best for: High-performance apps (games, augmented reality/virtual reality, finance, healthcare).

2. Progressive Web Apps

- **Definition**: Browser-based apps accessed via a URL (not installed from an app store). Built with HTML5, CSS, JavaScript.

- **Pros:**

  - **Cross-platform**: Works on any device with a browser.

  - **Lower development cost**: Single codebase for all platforms.

  - **No installation needed**: Runs in a browser.

  - **Easy updates**: Changes reflect instantly.

- **Cons**:

  - **Limited functionality**: No full access to device hardware.

  - **Slower performance**: Depends on browser & internet speed.

  - **No offline mode**: Requires an internet connection.

  - **No app store presence**: Harder to discover.

Best for: Content-driven apps (news, blogs, PWAs).

3. Hybrid Apps

- **Definition**: Combines native and web technologies (built with frameworks like React Native, Flutter, Ionic). Runs inside a native container but uses web views.

- **Pros**:

  - **Single codebase**: Works on both iOS & Android.

  - **Faster development**: Reuses web technologies.

  - **Access to device features**: Plugins enable native-like functionality.

  - **App store distribution**: Can be published like native apps.

  Cons:

  - **Performance trade-offs**: Slower than native but better than web.

  - **Dependency on frameworks**: Limited by third-party tools.

  - **UI inconsistencies**: May not feel fully native.

Best for: Budget-friendly cross-platform apps (e-commerce, social media, productivity tools).

Comparison Table

The table below does a brief comparison of features for the different types of mobile apps that exists;

Table 1:

Features	Native Apps	Progressive web apps	Hybrid Apps
Performance	High	Low	Medium
Development cost	High	Low	Medium
Time to Market	Slow	Fast	Moderate
Offline Support	Full	Partial	Full(with caching)
App store	Available	Not available	Available
Device access	Full	Limited	Partial
Best for	High-end apps	Web-first projects	Cross platform MVPs

V- The Various Programming Languages Used for Mobile App Development

Mobile app development utilizes several programming languages, each with specific characteristics that make them best suited for particular types of projects or problem domains. The primary languages include **Kotlin, Swift, Java, Dart, JavaScript, C#, and Objective-C**, among others. Below is a detailed explanation of each language, including its purpose, strengths, and ideal use cases.

1. Kotlin

Platform: Android (Native), Cross-Platform (KMM - Kotlin Multiplatform Mobile)
Created By: Jet Brains (2011)
Officially Adopted by Google for Android in 2017

Characteristics of kotlin:

- **Modern & Concise:** Reduces boilerplate code compared to Java.
- **Interoperable with Java:** Can seamlessly use Java libraries.
- **Null Safety:** Minimizes runtime crashes due to null references.
- **Coroutines: It** simplifies asynchronous programming (asynchronous programming is a techniques that allows the code to execute multiple tasks concurrently, improving the overall performance and responsiveness of the application).

Best For:

- Modern Android apps (replacing Java).
- Cross-platform projects via Kotlin Multiplatform (shared business logic).

2. Swift

Platform:iOS/macOS,(Native)
Created By: Apple (2014)

Characteristics:

- **Fast & Safe:** Designed for performance with memory safety features.
- **Readable Syntax:** Easier to learn than Objective-C.
- **Interoperable with Objective-C:** Supports legacy code integration.
- **Strong Apple Ecosystem:** Full access to iOS frameworks (ARKit, Core ML).

Best For:

- High-performance iOS/macOS apps.
- AR/VR, AI (Core ML), and Apple-exclusive applications.

3. Java

Platform: Android(Legacy),
Created By: Sun Microsystems **(1995)**

Characteristics:

- **Stable & Mature:** Extensive libraries and community support.
- **Platform Independence:** Runs on JVM (Java Virtual Machine).
- **Verbose Syntax:** More boilerplate than Kotlin.

Best For:

- Maintaining legacy Android applications.
- Enterprise apps requiring robust backend integration.

4. Dart (with Flutter)

Platform: Cross-Platform(iOS,Android,Web,Desktop)
Created By: Google (2011)

Characteristics:

- **Optimized for UI:** Flutter's framework uses Dart for smooth animations.

- **Hot Reload:** Speeds up development with instant code updates.
- **Single Codebase:** Build for multiple platforms simultaneously.

Best For:

- Cross-platform apps with custom UIs (e.g., MVPs, start-up apps).
- Projects needing rapid prototyping.

5. JavaScript (with React Native)

Platform: Cross-Platform,(iOS,Android)
Created By: Brendan Eich (1995)

Characteristics:

- **Web Technology Reuse:** Leverages existing JS skills.
- **Large Ecosystem:** npm (node package model) offers thousands of libraries.
- **Bridge Architecture:** Can impact performance vs. native.

Best For:

- Web developers transitioning to mobile.
- Apps requiring frequent updates (e.g., social media).

6. C# (with Xamarin/.NET MAUI)

Platform:

Cross-Platform,(iOS,Android,Windows)
Created By: Microsoft (2000)

Characteristics:

- **Strong Typing:** Reduces runtime errors.
- **Microsoft Ecosystem:** Integrates with Azure, Visual Studio

- **Code Sharing:** Shared logic across platforms.

Best For:

- Enterprise apps in .NET environments.
- Teams already using Microsoft tools.

7. Objective-C (Legacy)

Platform: iOS/macOS

(Native)
Created By: Brad Cox and Tom Love (1984)

Characteristics of Objective-C:

- **C with OOP Features:** Adds Smalltalk-style messaging.
- **Gradually Phased Out:** Replaced by Swift.

Best For:

- Maintaining older iOS apps.

8. Python

Used for mobile app development with frameworks like kivy, and Beeware. It's access to native features are limited.

Summary Table
The table below gives the key differences between the programming languages

Table 2 :

Language	**Platform**	**Strengths**	**Best For**
Kotlin	Android, Cross-Platform	Concise, null-safe, interoperable	Modern Android apps, shared logic
Swift	iOS/macOS	Fast, safe, Apple-optimized	High-performance iOS apps
Java	Android (Legacy)	Stable, vast ecosystem	Legacy Android, enterprise apps
Dart	Cross-Platform	Hot reload, single codebase	Flutter apps, MVP development
JavaScript	Cross-Platform	Web-friendly, large community	React Native apps, web-to-mobile
C#	Cross-Platform	Strong typing, .NET integration	Enterprise apps with Microsoft tools
Objective-C	iOS/macOS (Legacy)	Early Apple development	Maintaining old iOS codebases

In order to determine the best language to use for your project you have to consider the following;

- **Project Requirements** (performance, platform support).
- **Team Expertise** (existing knowledge of Java vs. Dart).
- **Long-Term Goals** (maintenance, scalability).

For **native performance**, Swift/Kotlin are ideal. For **cross-platform** needs, Flutter (Dart) or React Native (JavaScript) offer balanced trade-offs. Legacy systems may still rely on Java or Objective-C.

VI- Mobile App Development Frameworks Key Features Comparison

Key features considered when compairing the different frameworks: language, performance, cost & time to market, UX & UI, complexity, community support and where they can be used.

The table below compares the key features of each mobile app development framework

Table 3;

Framework	**Language**	**Performance**	**Cost & Time to Market**	**UX & UI**	**Complexity**	**Community Support**	**Use Cases**
React Native	JavaScript	Near-native	Fast, cost-effective	Good	Moderate	Strong	Cross-platform, MVPs
Flutter	Dart	Excellent	Quick, cost-effective	High-quality UI	Moderate to high	Growing	Complex UIs, animations
Xamarin	C#	Native	Slower, higher cost	Good	Moderate	Solid	Enterprise apps, deep integration
Ionic	HTML/CSS/JS	Lower	Fast, cost-effective	Good	Low to moderate	Strong	PWAs, simple mobile apps

Native Dev	Swift/Kotlin	Best	Long, higher cost	Superior	High	Strong	High-performance, complex apps

VII- Architecture and Design Patterns in Mobile Development

In modern mobile application development, architecture serves as the foundational blueprint that determines an application's scalability, maintainability, and long-term viability. As mobile projects grow in complexity—with features like real-time synchronization, offline capabilities, and AI integrations—the choice of architecture becomes increasingly consequential.

This technical deep dive examines four pivotal architectural patterns: **MVC (Model-View-Controller), MVVM (Model-View-View Model), VIPER (View-Interactor-Presenter-Entity-Router), and MVP (Model-View-Presenter)**. Through comparative analysis, implementation case studies, and empirical performance data, we demonstrate how each pattern addresses specific development challenges while exposing their inherent limitations.

The explanation of each of the design patterns is given below;

1. MVC: The Foundational Pattern

Architectural Breakdown

MVC is a foundational design patterns which partitions the application logic into three interconnected components:

1. **Model**: The data layer handling:
    - Business logic validation
    - Persistence operations (CoreData, Room)
    - API service integration
2. **View**: The presentation layer responsible for:
    - UI rendering (UIKit, XML layouts)
    - Input event detection
    - Basic formatting logic
3. **Controller**: The mediation layer that:
    - Receives user input from Views
    - Orchestrates Model updates
    - Controls View state changes

When to Consider MVC

- Prototyping phases requiring rapid iteration

- Legacy system maintenance
- Applications with limited business logic complexity

2. MVVM: Also known as the Reactive Paradigm

It is an architectural innovation particularly used in modern android and cross-platform development.

MVVM introduces critical improvements over MVC and is partitioned into:

a. **ViewModel Layer**:
   - Abstracts business logic from Views
   - Provides observable data streams (Combine, LiveData)
   - Handles data formatting and state management

b. **Data Binding**:
   - Automatic View updates via observers
   - Declarative UI frameworks (SwiftUI, Jetpack Compose)

Stress testing demonstrated:

- **Memory Efficiency**: 18% improvement over MVC
- **Render Consistency**: Stable 60FPS with complex lists
- **Network-State Handling**: 25% fewer race conditions

Implementation Challenges

- Learning curve for reactive programming
- Debugging complexity with chained observables
- 15-20% initial setup overhead

3.VIPER: Enterprise-Grade Modularity

It is an architecture based on clean architectural principles, focusing on a strong separation of concerns into distinct layers. Each layer has a specific responsibility, leading to highly testable and maintainable code, but it can be too complex for smaller applications.

VIPER's rigorous separation of concerns introduces:

1. **Interactor**: Pure business logic container
   - Isolated unit testing
   - Thread management
   - Repository coordination
2. **Router**: Navigation abstraction
   - Deep linking handling
   - Transition animations
   - Dependency injection

Case Study for the VIPER design pattern: Banking Application

A multinational bank's mobile app migration to VIPER yielded:

- 92% unit test coverage
- 70% reduction in merge conflicts
- 40% faster onboarding for new developers

Performance Tradeoffs

Benchmark comparisons showed:

- **Cold Start Time**: 12% slower due to DI overhead
- **Memory Footprint**: 25MB baseline vs MVVM's 18MB
- **Build Times**: 15% increase from modular compilation

4. MVP: The Transitional Architecture

Evolutionary Context

MVP emerged as a stopgap between MVC and MVVM, featuring:

1. **Passive View**:
    - Zero business logic
    - Interface contracts for test mocking
2. **Presenter**:
    - 1:1 mapping to Views
    - Manual data binding
    - Navigation coordination

VIII-Collecting and Analysing Use Requirements for a Mobile app Development (Requirement engineering)

I.Requirements Collection

When collecting user requirements, the following methods are used;

1. Define the apps idea and purpose: Clearly understand the core concept and the problem the app aims to solve.

2. Stakeholder interviews: Talk to stakeholders, including project managers, product owners, and end-users. This is done inorder to identify the goals and objectives from both the app's functionalities and the business strategy.

3. Surveys and questionnaires: Distribute surveys to gather information from a larger group of stakeholders.

4. Market and Competitor analysis: Research competing apps to identify features and functionalities.

5. User research: Conduct user interviews, usability testing, and A/B testing to understand user behavior.

6. Technical research: Research technical feasibility, infrastructure, and integration

Requirements.

7. Identify functional and non-functional requirements: Define what the app should do (functional) and how it should perform (non-functional).

After all this done it is important to document your requirements, this can be done using the following documentations

II. Requirements Documentation

1. Business Requirements Document (BRD): Outline business goals, objectives, and key performance indicators (KPIs).

2. Software Requirements Specification (SRS): A detailed document that outlines all the requirements, including functional and non-functional requirements.

3. Detailed Requirements Specification (DRS): A document that provides a detailed description of each requirement, including inputs, processing, outputs, and constraints.

4. Mobile app Requirement Documentation (MRD)

III. Requirements Analysis

1. Categorize requirements: Group requirements into functional, non-functional, and technical categories.

2. Prioritize requirements: Prioritize requirements based on business value, user needs, and technical feasibility.

3. Identify dependencies: Identify dependencies between requirements and components.

4. Estimate complexity: Estimate the complexity of each requirement.

IV. Requirements Validation

1. Review and revise: Review requirements with stakeholders and revise as necessary.

2. Prototype and test: Create prototypes to test and validate requirements.

3. Gather feedback: Gather feedback from stakeholders and end-users.

V. Requirements Management

1. Requirements tracking: Track requirements using a requirements management tool.

2. Change management: Establish a change management process to handle requirement changes.

3. Version control: Maintain version control to track changes to requirements.

IX- Estimating Mobile App Development Costs

After deciding to create a mobile app, you must carefully evaluate your project. How much will it cost? What are the time frames? A well-done estimation can provide correct answers to these problems. Below we discussed on how to estimate mobile app development cost, what you will need to do beforehand, and which price models are most typically utilized by developers.

How to Estimate the Cost of App Development

The price of building an app depends highly on its functionality and the wages of those who are going to build it. Every feature takes several hours to program, and the more complex the app, the higher you should estimate the cost of building and maintaining it after release.

When estimating the cost to develop a mobile app, the following are taking into consideration;

i- Features

The number of hours needed to program each app feature determines its price, which is more or less fixed. The more sophisticated the feature, the more expensive it will be.

ii- Supported Platforms

If we plan to develop an app specifically for one platform, native development (Kotin for Android and Swift for iOS) is the best option in terms of performance, stability, and cost. However, if we want your app to be available on both Google Play and Apple Store, we will consider using a cross-platform framework like Flutter, React Native, Xamarin, and others which is slightly more expensive than native development.

iii- Maintenance

Every app requires updates to ensure complete compatibility with new devices and operating systems (OSes), and regular bug fixing, because bugs are always present. Typically, these charges account for roughly 15% of the initial development cost each year.

Development Team Size

The complexity of the project should determine how big your app development team will be. Typically, a team has:

2 developers (iOS and Android, or cross-platform).

The QA (Quality Assurance) Engineer reviews all specifications and technical documents as well as creates and plans all testing activities.

The backend developer oversees data storage, payment systems, and the logic of app operation.

The UX/UI Designer creates a user-friendly interface for the app.

The project manager manages the development process and communicates directly with the client.

Team Locations

The team's geographical location influences the development costs. For example, software engineers in North America typically charge more than $100 per hour, whereas developers in Western Europe deliver code of comparable quality for half the price. When looking for developers, consider hiring an offshore development company or freelancers, as this can significantly impact the ultimate pricing of the project. To learn more about talent acquisition, read our article on where to hire developers.

The Process of App Cost Estimation

Before estimating mobile app development costs, we must analyze all ideas and concepts and convert them into a precise technical specification. The more thorough the information, the easier it will be to estimate expenditures. Software requirements and specifications typically include the following aspects.

Overview

This part briefly outlines the product and app concept. The team involved in the project creates lists of all app features, different user types, their roles, and classes. An overview also includes

information about the app's hardware and software requirements, design and implementation constraints, and a list of user guides that will be distributed.

Estimation Process

When we have the technical specs in hand, it is time to make estimates. To be precise, the process should include numerous stages, each of which clarifies the development needs more and more.

Stage 1. Breakdown

First, the provided technical specifications are decomposed into small and easy-to-estimate parts of the app. Usually, project teams break the specs down by screens or by functions.

By screens. The project team analyzes the wireframes of every screen for present functions (login, record audio) and UI elements (buttons, text fields).

By functions. The team analyzes every function of the app: how a function works, its design in the UI, and on what screens it is present.

Stage 2. Developer Estimation

Next, developers estimate how long it will take to program each function of the app and present their assessment as a range (minimum to maximum) of hours.

Stage 3. Project Manager Estimation

At this stage, the main priority of the project manager is to find the balance between price and quality. The PM reviews the development team estimates and includes the hours required for design, internal and external communications (team meetings, client demonstrations), focus group tests, code testing, and bug fixing.

Stage 4. Final Estimation

The project manager confirms his estimation with other team members, makes necessary corrections, and presents the results.

Pricing Models

The two most used pricing models for estimating app development costs are fixed price and time & materials.

Fixed Price

A fixed-price model determines a precise list of features, their implementation, and costs defined during the estimating process. If development grows to require any more works, it incurs additional costs and necessitates a thorough agreement procedure with the customer, resulting in development team downtime and an increase in development cost and time. If any

planned works are shown to be unneeded, the client's final payment remains the same. The set price model lacks flexibility, carries significant financial and time risks, and is primarily suitable for small and short-term projects.

Time & Material

Time & material is a flexible model that focuses not on building the exact list of features but rather on the result. The development process is reflected in monthly reports, and developers are allowed to build additional features without going through the full agreement process.

The development team provides a rough top-level estimation divided into stages (sprints). The client agrees on the scale of work.

The team starts building the app, presenting the work done to the client at the end of every sprint. The client pays only for what was done during the sprint.

If necessary, the team suggests additional work and, upon the client's agreement, includes them in the next sprint.

Finally, the client receives the finished product, adapted to the current technical requirements and market demands (which may have changed since the estimation).

Since the majority of apps are complex projects, some small details (an additional button on a screen or the logic for an uncommon user flow) are inevitably neglected during the planning process, and the time & material model enables developers to include them in the process quickly and easily. Furthermore, if the fixed-price model estimates each feature with potential issues that may not occur, the time and material model reduces such risks for the development team, allowing it to offer more comfortable prices to the client.

Finally,

Mobile app estimation is a process that takes into account all aspects of the future project and is based on a detailed technical specification. The process is conducted in several stages, beginning with the breakdown of the project by screens or features, followed by an appraisal by developers, and finally by a project manager. Although the development team may provide the client with a precise cost for app development and work along the guidelines of the fixed-price model, a top-level prediction based on the time & material price model bears fewer risks for both parties. It is also more flexible to make essential modifications after work has begun.

X- Conclusion

Mobile app development offers multiple approaches, each with its own advantages and trade-offs. Native development provides the best performance and user experience but at higher cost. Cross-platform frameworks like Flutter and React Native offer good compromises between performance and development efficiency. PWAs are excellent for reach and simplicity but lack some native capabilities.

The choice of technology stack should be based on project requirements, budget, timeline, and target audience. Proper requirement engineering and architecture selection are crucial for long-term maintainability and scalability of mobile applications. Cost estimation should consider both initial development and ongoing maintenance expenses.

As mobile technologies continue to evolve, developers must stay informed about new frameworks, tools, and best practices to deliver high-quality mobile applications that meet user needs and business objectives.