UNIVERSITY OF BUEA

REPUBLIC OF CAMEROON

PEACE-WORK-FATHERLAND

P.O. Box 63,
Buea, South West Region
CAMEROON
Tel : (237) 3332 21 34/3332 26 90
Fax: (237) 3332 22 72

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

MOBILE APP DEVELOPMENT PROCESS

By:

BERINYUY CLETUS FE21A148
EFUETLATEH JOAN FE22A197
ETAPE NGABE FE22A210
ETIENDEM PEARL FE22A211
TATA GLEN FE22A309

2024/2025 Academic Year

Contents

1. Types of Mobile Apps and Their Differences

Introduction

- Mobile apps are software applications designed to run on mobile devices like smartphones and tablets.
- They can be categorized into **three major types** based on how they are developed and deployed:
    1. Native Apps
    2. Progressive Web Apps (PWAs)
    3. Hybrid Apps
- Understanding these differences helps in selecting the best approach for app development.

1. Native Apps

Definition:

- Apps specifically built for a particular operating system (OS) using platform-specific languages and tools.
- Example: Android apps use Java/Kotlin, while iOS apps use Swift/Objective-C.

Features:

- High performance and responsiveness
- Access to all device features (camera, GPS, notifications)
- Better user experience (UX) with optimized UI
- Expensive to develop (separate codebase for Android & iOS)
- Requires approval from app stores

Examples:

- WhatsApp (Android & iOS)
- Spotify (Android & iOS)

2. Progressive Web Apps (PWAs)

Definition:

- Web applications that function like native apps but run in a browser.
- Built using standard web technologies (HTML, CSS, JavaScript).

Features:

- No installation required; accessed via a browser

- Works across different devices (responsive design)
- Updates automatically without requiring downloads
- Cheaper and faster to develop
- Limited access to device features (e.g., restricted access to Bluetooth, sensors)
- Slower performance compared to native apps

Examples:

- Pinterest
- Google Maps Go

3. Hybrid Apps

Definition:

- A combination of native and web apps, built using web technologies but wrapped in a native shell.
- Can be deployed across multiple platforms with a single codebase.

Features:

- Faster development compared to native apps
- Works on multiple platforms (Android & iOS)
- Access to some native device features
- Slightly slower performance than fully native apps
- May not offer the same level of UX as native apps

Examples:

- Instagram
- X **(**formerly Twitter)

Comparison Table

Feature	**Native Apps**	**Progressive Web Apps**	**Hybrid Apps**
Performance	High	Moderate	Moderate
Device Access	Full	Limited	Partial
Development Cost	High	Low	Moderate
Installation	From App store	No installation required	From App store

Best for	High-performance apps	Simple, lightweight apps	Apps with a mix of Web and native features

Figure 1: Comparison Table

Conclusion

- Native apps are best for high-performance, feature-rich applications.
- Progressive Web Apps are ideal for lightweight, easily accessible applications.
- Hybrid apps offer a balance between native performance and web flexibility.
- Choosing the right type depends on budget, performance needs, and target audience.

2. Mobile app programming languages

Programming languages for Native apps:

a. iOS:

There are two native programming languages for iOS development; Objective-C and Swift

- **Objective-C**

Objective-C was the first programming language by Apple to support mobile applications on its platform. It's an OO (object-oriented programming language). The language isn't very developer-friendly. One of the drawbacks is that the syntax feels clunky(inefficient), and the square brackets can be tough to debug**.**

- **Swift**

Swift was introduced in 2014 as an Apple programming language. This language has quickly become the developers' preferred language when building an iOS app as it offers safety features, modern syntax and seamless integration with apple's hardware. Compared to Objective-C and other programming languages, Swift is easier and more compact.

b. Android

- Java

Since Android was officially launched in 2008, Java has been the default development language to write Android apps. While Java has its fair share of faults, it's still the most popular language for Android development since it runs on a virtual machine. As an object-oriented option for mobile development, Java is commonly used to develop Android apps.

- Kotlin

It's an alternative language to traditional Java for Android development, and it runs on the Java Virtual Machine. Even as a new language, it's very popular. Kotlin and Java are interoperable(compatible), meaning they can make use of the same information. All of your Java libraries can be accessed with Kotlin. From an execution standpoint, the Kotlin language complies with Java Bytecode. Overall, it's considered a neater and cleaner version of Java.

Native Programming Pros:

1. Most control over the device
2. Low-level coding for cutting edge technologies that are added on to the device
3. Fastest access to latest and greatest features through your language
4. Fastest in execution bottom line

Native Programming Cons:

1. Slowest to develop
2. Most costly development method
3. Takes highest skilled and specialized mobile app developers to build for iOS and Android

Programming languages for Hybrid apps:

a. C#

Developed by Microsoft, C# (pronounced C sharp) is another object-oriented programming language. It's a popular programming language for game development and command line scripting for Android operating systems.

b. Xamarin

Technically, Xamarin isn't a language. It's an open-source development platform for iOS, Android, and Windows applications. It's a .NET platform that uses C# as its core language.

c. Javascript

Used in the React Native framework enables cross-platform app dev. It's ideal for web developers and offers a low barrier to entry.

Programming languages for PWA apps:

a. Ruby

Ruby is a general-purpose programming language that can be used for a wide range of use cases, including PWAs. Lots of developers rely on Ruby for web applications because of its simplicity. While shipping code with Ruby is easy, finding bugs and debugging errors is not always as simple.

b. JavaScript

If you have web development experience, using JavaScript to create a PWA might be the best option for you. You can use JS on top of HTML and alongside CSS to create your web application from scratch.

Compared to other options for PWA, this has a lower barrier to entry for those of you who have some basic technical knowledge and coding experience.

Key Considerations

When choosing a programming language for mobile app dev consider factors like:

- Native languages generally provide better performance but cross platform languages can offer acceptable performance with proper optimization.
- Cross platform languages can reduce dev time, but may require additional setup and configuration
- Languages like swift, kotlin and javascript have a relatively gentle learning curves, while Java and C# may require more time and effort to master

3. Overview of Mobile App Development Frameworks

a. Definition:

A mobile app development framework is a software library that provides developers with tools, APIs, and structures to create mobile applications efficiently. They provide pre-written code, templates, and tools to allow developers to focus on the application's functionality and user experience rather than starting everything from scratch.

b. Types:

- **Native Frameworks**

Frameworks designed for a specific platform (e.g. Swift for iOS, Kotlin for Android)

- **Cross-Platform Frameworks**

Frameworks that allow developers to write code once and deploy it on multiple platforms (e.g., Flutter, React Native, Xamarin).

- **Hybrid Frameworks**

Combine web and mobile app development using technologies like HTML5, CSS, and JavaScript (e.g., Apache Cordova, Ionic) with native capabilities.

c. **Key Components of Mobile App Frameworks**
   - **UI:** Prebuilt components such as buttons, sliders, and navigation bars that developers can use to create the App's UI.
   - **APIs:** Frameworks often provide access to a wide range of APIs and libraries for functionalities like networking, databases and analytics (Eg, Firebase for backend services, axios for HTTP requests)
   - **Dev Tools:** IDEs, code editors and debugging tools that assist in writing and testing code.

d. Popular Frameworks
   - **React Native:** Developed by Facebook, it allows building apps using JavaScript and React components, enabling native performance and a richer UI.
   - **Flutter:** Developed by Google, it utilizes Dart to create natively compiled applications for mobile, web, and desktop from a single codebase.
   - **Xamarin**
   - **Ionic:** Uses web technologies to build hybrid mobile apps, often with Angular or React. It provides a rich library of UI components.

e. Key Benefits Frameworks
   - Provide pre-built components and templates which saves development time and efforts increasing development speed.
   - Frameworks often support modular development, making it easier to scale applications as needed.
   - Most Frameworks have active communities that contribute to documentation, plugins, and troubleshooting.
   - Ensure better performance and UI consistency.

4. Mobile Application Architectures and Design Patterns

a. Introduction

Mobile applications are an essential part of modern technology, and their success depends on how well they are structured. Choosing the right

mobile architecture helps ensure that the app is efficient, scalable, and easy to maintain. Similarly, design patterns provide reusable solutions to common problems, making development faster and more reliable. We will discuss three major mobile architectures and four important design patterns.

b. Mobile Application Architectures

What is Mobile Application Architecture?

A mobile application architecture defines how different parts of an app interact with each other. A well-designed architecture helps developers build secure, maintainable, and high-performing applications.

c. Common Mobile Architectures

1. **Monolithic Architecture**
   - In a monolithic architecture, the entire application is built as one single unit where all components (UI, database, logic) are tightly connected.

Advantages

- Simple to develop and deploy.
- Easy debugging since everything is in one place.

Disadvantages

- Difficult to scale—if one part fails, the whole app is affected.
- Harder to update and maintain as the app grows.

Used for: Small apps and early-stage prototypes.

2. **Microservices Architecture**
   - This architecture divides the application into smaller independent services, each responsible for a specific function. These services communicate using APIs.

Advantages

- Highly scalable—services can be updated or expanded independently.
- Fault-tolerant—if one service fails, others continue working.

Disadvantages

- Complex to develop and manage multiple services.
- Requires strong API design for smooth communication.

Used for: Large-scale applications like Netflix, Uber, and Amazon

3. **Model-View-ViewModel (MVVM) Architecture**
   - MVVM is a structured approach where the application is divided into three parts:

1. **Model**: Handles data and business logic.

2. **View**: Represents the user interface (UI).
3. **ViewModel**: Acts as a bridge between the Model and View, ensuring they don't directly depend on each other.

Advantages

- Improves code reusability and maintainability.
- Easier testing because UI and business logic are separate.

Disadvantages

- Can be complex for beginners.

Used for: Android and iOS apps (e.g., Jetpack Compose, SwiftUI).

d. Mobile Design Patterns

What are Design Patterns?

Design patterns are reusable solutions to common software development challenges. They improve code organization, efficiency, and maintainability.

Common Design Patterns

1. **Singleton Pattern**
   - Ensures only one instance of a class exists across the entire application.

   Advantages
   - Saves memory by preventing multiple unnecessary instances.
   - Useful for shared resources like database connections.

   Used for: Managing a database connection or app settings

2. **Factory Pattern**
   - Dynamically creates objects instead of manually instantiating them.

   Advantages
   - Reduces dependencies and makes code more flexible.
   - Helps manage complex object creation processes.

   Used for: Generating different types of UI components dynamically.

3. **Observer Pattern**
   - Allows multiple components to react to a change in data.

   Advantages
   - Improves efficiency in event-driven applications.
   - Keeps UI updated with minimal manual intervention.

   Used for: Push notifications and real-time updates (e.g., chat apps).

4. **Repository Pattern**
   - Provides an abstraction layer for managing data sources (local database, cloud, API).

Advantages

- Makes data handling easier and more scalable.
- Helps synchronize offline and online data efficiently.

Used for: Fetching data from multiple sources in apps.

e. Why Mobile Architectures Matter?

- **Performance**: Ensuring the app runs smoothly and efficiently.
- **Scalability**: Allowing the app to grow without breaking functionality.
- **Maintainability**: Making it easier to update and add new features.

5. Collecting and Analyzing User Requirements for a Mobile Application (Requirement Engineering)

a. What is Requirement Engineering?

Requirement engineering is the process of **gathering, analyzing, validating, and documenting** the needs and requirements for a mobile application.

b. Steps to Collect and Analyze User Requirements:

1. **Stakeholder Identification:** Identify the key people (clients, users, developers) involved.
2. **Requirement Gathering Techniques:**
   - **Interviews:** Direct discussions with stakeholders.
   - **Surveys/Questionnaires:** Collect feedback from a larger group.
   - **Observation:** Understand how users interact with current solutions.
   - **Workshops/Focus Groups:** Group discussions to define needs.
3. **Requirement Analysis:**
   - **Categorize Requirements:** Functional (what the app should do) and Non-Functional (performance, usability).
   - **Prioritize:** Rank requirements based on importance and feasibility.
   - **Validate:** Ensure requirements align with stakeholders' expectations.
4. **Documentation:**
   - Use tools like **JIRA, Confluence, or Google Docs** for documentation.
   - Create **user stories** and **use case diagrams**.

5. **Review and Approval:** Get feedback from stakeholders and refine as needed.

6. Estimating Mobile App Development Cost

a. Factors Influencing Cost:

1. **Complexity:** Simple apps (basic features) vs. complex apps (advanced features, integrations).
2. **Platform:** Android, iOS, or cross-platform (React Native, Flutter).
3. **Development Team:** Freelancers, in-house developers, or agencies.
4. **Design Requirements:** Simple UI vs. custom, complex designs.
5. **Maintenance:** Regular updates, bug fixes, and improvements.

b. Cost Estimation Techniques:

1. **Analogous Estimation:** Use data from similar projects.
2. **Parametric Estimation:** Calculate based on the number of hours multiplied by the hourly rate.
3. **Bottom-Up Estimation:** Estimate each feature or module separately and add up the total.
4. **Expert Judgment:** Get insights from experienced developers.