REPUBLIQUE DU CAMEROUN	**REPUBLIC OF CAMEROON**
Paix – Travail – Paitrie	**Peace – Work – Fatherland**

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

CEF440: INTERNET PROGRAMMING (J2EE) AND MOBILE PROGRAMMING

COURSE TASK ONE : FUNDERMENTALS OF INTERNET AND MOBILE PROGRAMMING

GROUP 1: MEMBERS

Name	Matricule
MEWOABI NGUEFACK DORE	FE21A239
ALEANU NTIMAEH ENOW	FE21A134
OJONG-ENYANG OYERE	FE21A297
FONGANG KELUAM PAUL DIEUDONNE	FE21A193
TANWIE BRUNO ADEY	FE21A316
TCHUIDJAN JORDAN BRYANT	FE21A320

COURSE INSTRUCTOR: Dr. NKEMENI VALERY

Contents

INTRODUCTION

In today's digital era, mobile applications have become an integral part of our daily lives, facilitating seamless access to information, services, and entertainment. With the ever-growing demand for mobile apps, developers are faced with a myriad of choices, ranging from selecting the right development approach to choosing the most suitable programming languages and frameworks. This report aims to provide a comprehensive analysis of the mobile app development landscape, focusing on key areas such as types of mobile apps, programming languages, development frameworks, architectures, requirement engineering, and cost estimation. Let's define some of the main terms below

Internet programming:

Internet programming involves creating software applications that are accessed through the internet. These applications typically run on web servers and are accessed by users through web browsers. Internet programming can involve different technologies such as; HTML, CSS, JS, server-side scripting languages like; PHP, PYTHON, NODE.JS and databases such as; MYSQL or MongoDB.

Internet programming is used to create websites, web applications and web services that perform a wide range of functions from simple websites to complex e-commerce and social networking sites.

Mobile programming:

Mobile programming on the other hand involves creating software applications specifically designed to run on mobile devices such as; smartphones and tablets. Mobile programming often involves developing applications for popular mobile operating systems like iOS (for Apple devices) and Android (for devices from various manufacturers). Mobile programming can be done using various programming languages such as Swift or Objective-C for iOS development, and Java or Kotlin for Android development. Additionally, cross-platform frameworks like React Native, Flutter, or Xamarin enable developers to write code once and deploy it on multiple platforms.

1) REVIEW AND COMPARISON BETWEEN THE MAJOR TYPES OF MOBILE APPS AND THEIR DIFFERENCES.

The landscape of mobile development is diverse offering developers multiple approaches to create mobile applications to meet different needs and preferences. There exist three major types of mobile apps which dominates the mobile development industry: Native Apps, Progressive Web Apps (PWAs) and Hybrid Apps. Each type having its own set of characteristics, advantages and disadvantages.

Choosing the right approach depends on factors such as performance requirements, targeted audience, budget and development timeline.

1.1) Native Apps:

- Native Apps are platform specific applications developed for either IOS or Android using using languages, tools and platform specific SDKs provided by the respective platforms such as Apple and Google i.e. (using Sift or Objective-C for IOS Applications) and (Java or Kotlin for Android applications).
- Native apps have full access to device features and APIs, providing high performance and smooth user experience.
- They can take advantage of platform specific design guidelines, resulting in a polished look and feel (High performance and smooth experience).
- Update and maintenance need to be separately managed for each platform increasing development time and cost.

1.2) Progressive Web Apps (PWAs):

- Progressive Web Apps are web applications that uses modern web technology (HTML, CSS and JavaScript) to deliver an app like experience for users across all platforms and devices.
- They are designed to work across all platforms and devices with a responsive design approach.
- PWAs can be access through web browsers eliminating the need to install mobile applications.

- They can work both offline and with poor internet connection by utilizing service workers for caching content and improve discoverability through search engines.
- While offering cross platform compatibility, they may have limited access to device features compared to native apps.

1.3) Hybrid Apps:

- Hybrid apps combine elements of both native and web applications. They are developed using web technologies such as HTML, CSS and JavaScript like progressive web apps but are wrapped in a native container to access device features.
- Hybrid frameworks such as React Native, Apache Cordova and Ionic enable developers to build apps using web technologies and access native device features through plugins.
- Hybrid applications offer a single codebase for multiple platforms, reducing development time and cost.
- Maintenance and update are easier compared to the native apps since changes can be deployed across multiple platforms simultaneously. However, they can't match the performance of the native applications.

Feature	Native Mobile App	Progressive Web App (PWA)	Hybrid Mobile App
Development Technology	Specific to platform (e.g., Swift for iOS, Java/Kotlin for Android)	Web technologies (HTML, CSS, JavaScript)	Combination of web technologies (HTML, CSS, JavaScript) and native code
Installation	Downloaded from app store	Accessed via browser and can be optionally installed to home screen	Downloaded from app store
Access to Device Features	Full access to device features (camera, GPS, accelerometer, etc.)	Limited access compared to native apps	Access depends on the framework used and plugins integrated
Performance	Generally faster and smoother performance	Typically slower compared to native apps, but improving	Can be slower than native apps due to the use of web views

Offline Functionality	Can work fully offline	Limited offline functionality with service workers, caching, etc.	Depends on implementation, can vary from full offline functionality to none
App Updates	Updates are pushed through app stores, users need to download updates	Updated in real-time, no need for users to download updates	Updates can be pushed through app stores or via the web
Platform-Specific Features	Can fully utilize platform-specific features and design elements	Limited access to platform-specific features, generally look and feel like a web app	Can access platform-specific features through plugins but may not fully utilize native UI components
Distribution	Distributed through app stores (e.g., Apple App Store, Google Play Store)	Can be accessed via URL or discovered through search engines, can also be distributed through app stores	Distributed through app stores
Development Cost	Higher cost due to separate development for each platform	Generally lower cost due to single codebase, but may require additional development for platform-specific features	Lower cost compared to native apps, but higher than PWAs due to platform-specific development
User Engagement	High user engagement due to app store presence and notifications	Can have lower user engagement compared to native apps, but improving	Moderate user engagement, can vary depending on implementation
Examples	Instagram, Spotify (iOS/Android versions), Pokemon Go	Twitter Lite, Flipkart, Pinterest	Instagram (older versions), Evernote

2) REVIEW AND COMPARISON OF MOBILE APP PROGRAMMING LANGUAGES:

Here, we explore the primary programming languages used in mobile app development, including Swift and Kotlin for native development, and JavaScript for hybrid and PWA development. A comparative analysis is provided, highlighting the strengths, weaknesses, and suitability of each language for various development scenarios. The programming languages shall be listed or classified in order of the mobile app type they are used to build

2.1) Native Android app Development Programming Languages

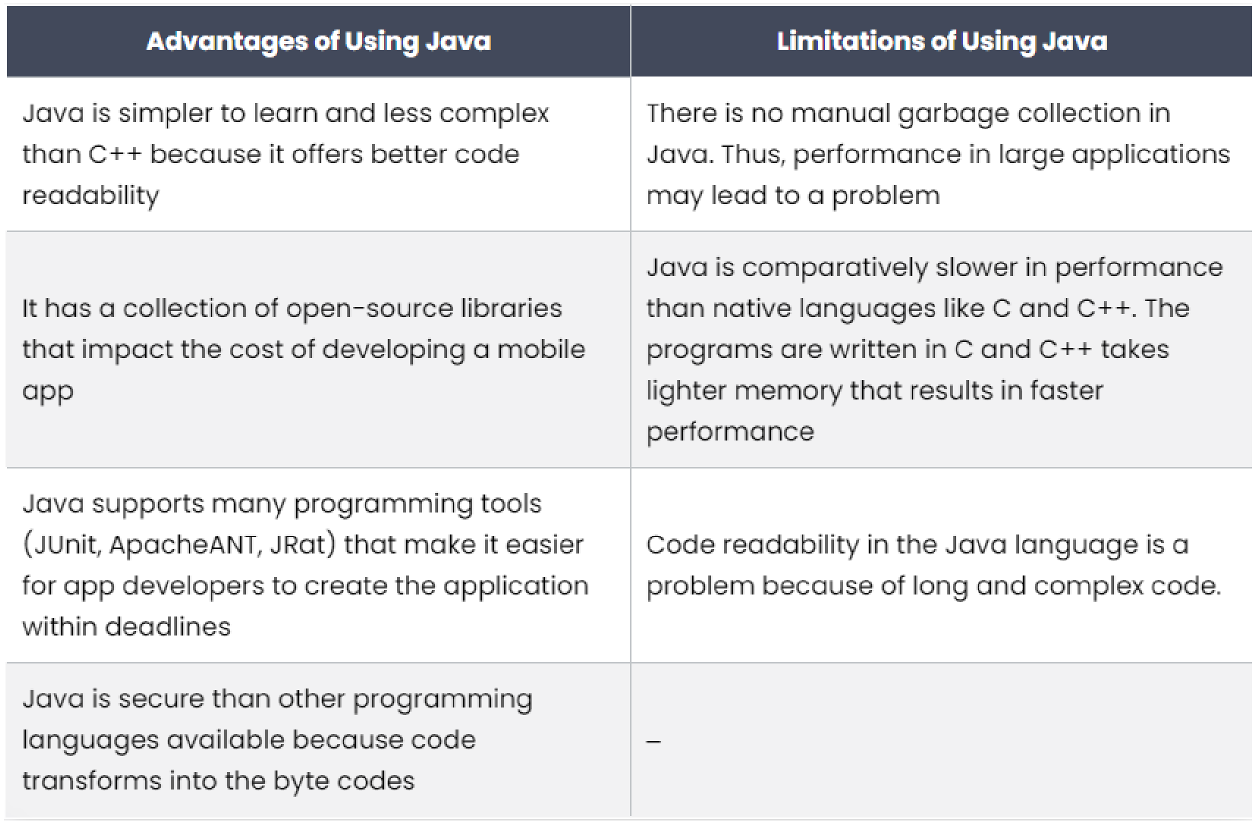
2.1.1) Java

Java stands as a high-level, object-oriented, multipurpose programming language, empowering developers to craft software solutions for desktops and mobile devices alike. Recognized as the default language for mobile app development, it holds a prominent status within the Android Studio IDE, a favored environment for Android application creation.

Many mobile app developers tout Java as the premier choice for Android app development, owing to its versatility and broad compatibility across diverse hardware configurations. The language's platform-agnostic nature allows programs written in Java to seamlessly operate across various devices and operating systems.

Renowned for its scalability, flexibility, and extensive collection of libraries and tools, Java is the preferred language for both mobile app development and the creation of enterprise-level software solutions. Its versatility extends beyond mobile applications, finding integration in consumer electronics like televisions, mobile devices, and digital assistants. Moreover, Java plays a pivotal role in network systems, facilitating the development of server-client architecture-based applications.

The figure below shows the strengths and limitations of using java for mobile programming



Advantages of Using Java	Limitations of Using Java
Java is simpler to learn and less complex than C++ because it offers better code readability	There is no manual garbage collection in Java. Thus, performance in large applications may lead to a problem
It has a collection of open-source libraries that impact the cost of developing a mobile app	Java is comparatively slower in performance than native languages like C and C++. The programs are written in C and C++ takes lighter memory that results in faster performance
Java supports many programming tools (JUnit, ApacheANT, JRat) that make it easier for app developers to create the application within deadlines	Code readability in the Java language is a problem because of long and complex code.
Java is secure than other programming languages available because code transforms into the byte codes	–

fig 2.1 advantages and limitations of using java as programming language. Src: https://www.spaceo.ca/blog/best-mobile-app-development-languages/ date : 29th march 2024

2.1.2) kotlin

Kotlin emerged as a response to the shortcomings observed in Java for mobile app development, offering a robust alternative that is both powerful and simpler

to use. This language is specifically tailored for mobile app development, providing developers with a streamlined and efficient programming experience compared to Java.

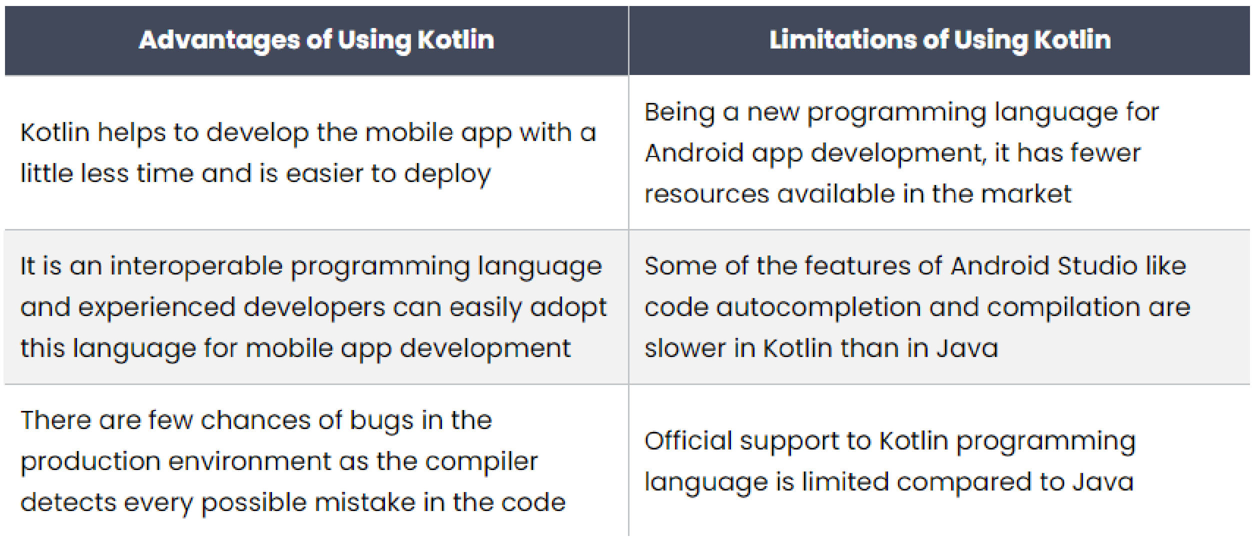
The primary objectives of Kotlin center around promoting interoperability, code clarity, safety, and robust tooling support, particularly for Android app development. With Kotlin, developers can easily craft functional programs and conduct smooth testing of mobile applications, enhancing productivity and facilitating the development process.

In addition, Kotlin introduces the Multiplatform Mobile SDK, enabling developers to build cross-platform apps effortlessly. Consequently, when confronted with the choice between leading frameworks like Flutter and the Kotlin language, developers often find themselves grappling with a dilemma. For a comprehensive understanding of this comparison, refer to our detailed guide on Flutter versus Kotlin for cross-platform mobile app development.

Employing Kotlin for mobile app development not only boosts team productivity but also elevates the overall quality of mobile applications. For instance, HeadSpace experienced a notable 20% increase in monthly active users after adopting Kotlin in their mobile application development. Recognizing the potential for improvement, HeadSpace's Android app development team embarked on migrating the entire application to the Kotlin language.

Furthermore, it is imperative to examine the advantages and drawbacks of utilizing Kotlin for Android app development.

The figure below shows the strengths and limitations of using kotlin for mobile programming



Advantages of Using Kotlin	Limitations of Using Kotlin
Kotlin helps to develop the mobile app with a little less time and is easier to deploy	Being a new programming language for Android app development, it has fewer resources available in the market
It is an interoperable programming language and experienced developers can easily adopt this language for mobile app development	Some of the features of Android Studio like code autocompletion and compilation are slower in Kotlin than in Java
There are few chances of bugs in the production environment as the compiler detects every possible mistake in the code	Official support to Kotlin programming language is limited compared to Java

fig 2.2 advantages and limitations of using kotlin as programming language. Src: https://www.spaceo.ca/blog/best-mobile-app-development-languages/ *date : 29th march 2024*

2.2) Native iOS App Development Programming Languages

2.2.1) Swift

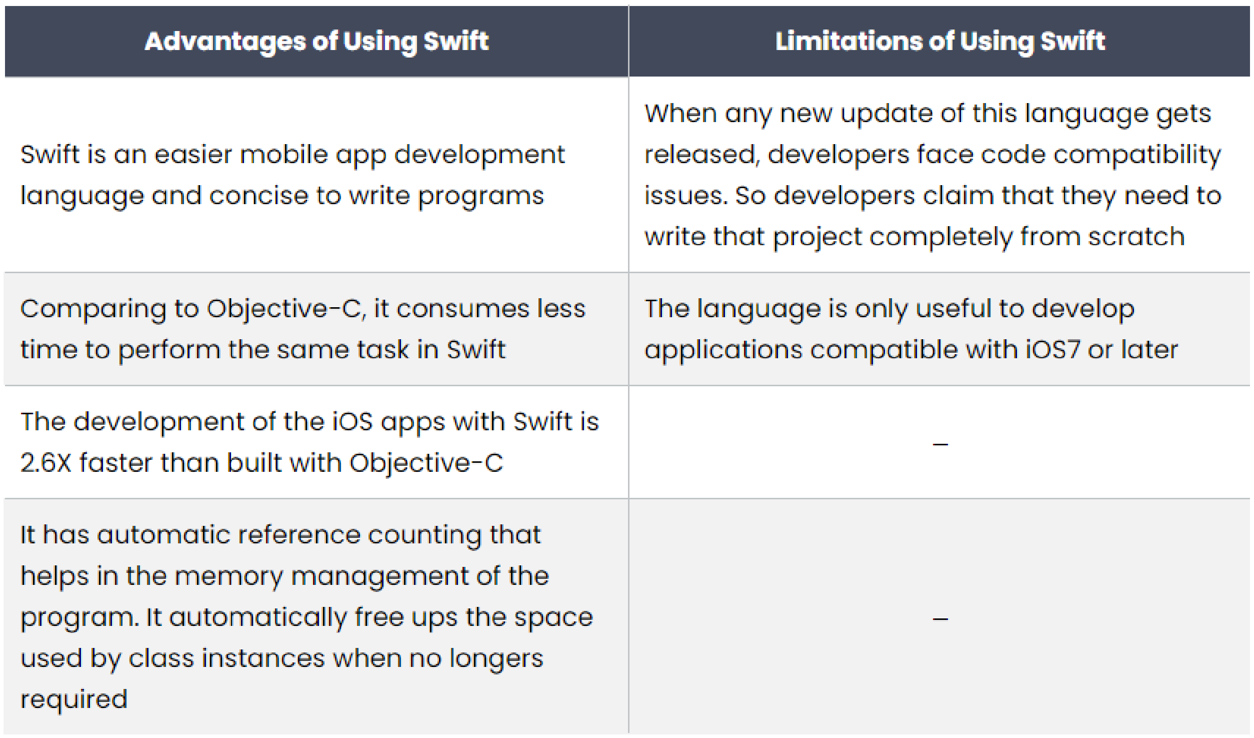
For developers seeking to create applications across various Apple platforms such as iOS, iPadOS, macOS, tvOS, or watchOS, Swift stands out as the premier programming language of choice. With its concise syntax, Swift facilitates enhanced readability and maintenance of application code, empowering developers to craft robust and efficient apps that capitalize on the capabilities of the underlying hardware.

Moreover, Swift extends its utility beyond mobile app development, finding application in the creation of modern server applications as well. When utilized for server-side development on the iOS platform, Swift offers comprehensive support, including runtime safety, optimized performance, and a minimal memory footprint.

One notable advantage of Swift is its interoperability with Objective-C, enabling seamless integration with existing mobile applications developed using Objective-C. This interoperability grants developers access to the full spectrum of Objective-C APIs, facilitating the extension and enhancement of projects built with Objective-C.

As with any programming language, it is essential to consider both the advantages and limitations of Swift in order to make informed decisions regarding its usage.

The figure below shows the strengths and limitations of swift programming language



Advantages of Using Swift	Limitations of Using Swift
Swift is an easier mobile app development language and concise to write programs	When any new update of this language gets released, developers face code compatibility issues. So developers claim that they need to write that project completely from scratch
Comparing to Objective-C, it consumes less time to perform the same task in Swift	The language is only useful to develop applications compatible with iOS7 or later
The development of the iOS apps with Swift is 2.6X faster than built with Objective-C	–
It has automatic reference counting that helps in the memory management of the program. It automatically free ups the space used by class instances when no longers required	–

fig 2.3 advantages and limitations of using swift as programming language. Src: https://www.spaceo.ca/blog/best-mobile-app-development-languages/ *date : 29th march 2024*

2.2.2) Objective-C

Objective-C serves as an object-oriented programming language that inherits the syntax of the C language while incorporating elements of Smalltalk to streamline the process of problem-solving during programming endeavors. Predominantly utilized during the 1980s and 1990s, Objective-C emerged as the language of choice for software development tasks.

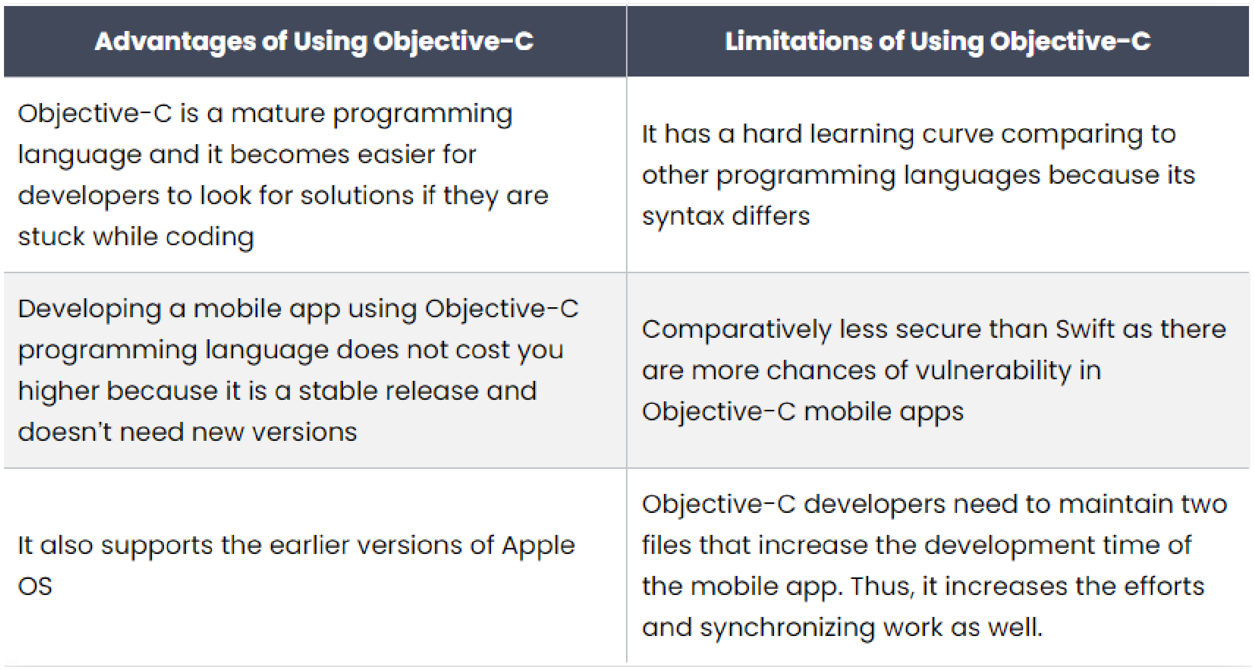
When juxtaposed with the Swift programming language, Objective-C distinguishes itself as the most mature and extensively updated language employed in the realm of app development. Its longevity and continuous evolution position it as a stalwart within the development community, offering a robust foundation for crafting high-quality applications.

On the other hand, Swift, despite its advancements and popularity, lacks ABI (Application Binary Interface) stability. This absence of ABI stability entails that Swift does not establish reliable communication with machine code. Consequently, if one intends to develop a software development kit (SDK) for their mobile application, Objective-C emerges as the preferable choice due to its established compatibility and reliability.

Delving deeper into the concept of ABI stability, it signifies the ability of a programming language to maintain consistent communication with the underlying machine code across different versions and platforms. As Swift currently lacks ABI stability, developers may encounter challenges in ensuring seamless interoperability and compatibility with existing codebases and system architectures.

In essence, while both Objective-C and Swift offer distinct advantages and capabilities for developing applications for Apple products, it is crucial to weigh the benefits and limitations of each language to determine the most suitable option for a given development project.

The figure below shows the strengths and limitations of swift programming language



Advantages of Using Objective-C	Limitations of Using Objective-C
Objective-C is a mature programming language and it becomes easier for developers to look for solutions if they are stuck while coding	It has a hard learning curve comparing to other programming languages because its syntax differs
Developing a mobile app using Objective-C programming language does not cost you higher because it is a stable release and doesn't need new versions	Comparatively less secure than Swift as there are more chances of vulnerability in Objective-C mobile apps
It also supports the earlier versions of Apple OS	Objective-C developers need to maintain two files that increase the development time of the mobile app. Thus, it increases the efforts and synchronizing work as well.

fig 2.4 advantages and limitations of Objective-C as programming language. Src: https://www.spaceo.ca/blog/best-mobile-app-development-languages/ *date : 29th march 2024*

2.3) Hybrid And PWA Mobile App Development

2.3.1) Javascript

JavaScript is a versatile, high-level programming language commonly used for web development.

It is interpreted, allowing for dynamic runtime behavior, and supports both functional and object-oriented programming paradigms.

JavaScript serves as the primary language for developing hybrid mobile applications using frameworks like Ionic, PhoneGap, and React Native.

Its ubiquity in web development makes it accessible to a broad range of developers, facilitating code reuse and skill portability.

Hybrid mobile app development using JavaScript is ideal for projects requiring cross-platform compatibility, rapid development cycles, and a balance between development cost and performance.

It is particularly well-suited for startups and small to medium-sized businesses looking to build cost-effective mobile applications with a faster time-to-market.

Advantages of using JavaScript	Limitations of using JavaScript
Cross-Platform Compatibility: Hybrid mobile apps written in JavaScript can run on multiple platforms, including iOS and Android, with minimal modifications.	Performance Overhead: While hybrid apps can approach native performance, they may still exhibit slight performance overhead compared to fully native apps due to the reliance on webviews for rendering UI.
Rapid Development: JavaScript's dynamic nature and extensive ecosystem of libraries and frameworks accelerate the development process, enabling rapid prototyping and iteration.	Limited Access to Native Features: While hybrid frameworks provide access to native device features through plugins, certain advanced functionalities may be challenging to implement or may require additional development effort.
Native-Like Performance: With advancements in hybrid app frameworks like React Native, JavaScript-powered apps can achieve near-native performance by leveraging platform-specific APIs and components.	Dependency on Frameworks: Hybrid app development relies heavily on third-party frameworks and libraries, leading to potential compatibility issues and reliance on framework updates.

fig 2.5 advantages and limitations of JavaScript as programming language.

2.3.2) Dart

Dart, developed by Google, is a modern and object-oriented programming language designed for building web, server, and mobile applications. It serves as the primary language for developing mobile apps with the Flutter framework, offering a unified solution for cross-platform development. Dart's clean syntax, performance optimizations, and compatibility with Flutter make it a popular choice among developers for building high-quality mobile applications. Dart is the language of choice for developing mobile apps with the Flutter framework. Flutter is a UI toolkit from Google that enables developers to build natively compiled applications for mobile, web, and desktop from a single codebase. Dart's integration with Flutter provides a seamless development experience and allows developers to create high-performance, visually stunning apps.

Dart is optimized for performance, making it well-suited for mobile app development. Flutter apps built with Dart can achieve high performance and smooth animations, comparable to native apps. Dart's ahead-of-time (AOT) compilation and just-in-time (JIT) compilation contribute to the speed and efficiency of Flutter apps.

Dart features a clean and expressive syntax that is easy to read and write. It offers modern language features such as async/await for asynchronous programming, optional typing for increased flexibility, and strong support for object-oriented programming paradigms. Dart's syntax makes it accessible to developers of all skill levels, from beginners to seasoned professionals.

Dart and Flutter enable developers to target multiple platforms with a single codebase. This includes iOS, Android, web, and desktop applications. With Flutter, developers can write once and deploy everywhere, reducing development time and effort while ensuring a consistent user experience across different platforms.

Advantages of using Dart	Limitations of using Dart
Flutter Framework: Dart is the language of choice for developing mobile apps with Flutter, a UI toolkit from Google for building natively compiled applications for mobile, web, and desktop from a single codebase.	Learning Curve: While Dart's syntax is relatively straightforward, developers may encounter a learning curve, especially if they are new to reactive UI frameworks like Flutter.
Hot Reload: Dart and Flutter offer a hot reload feature, allowing developers to make changes to the code and see the results instantly on the emulator or device without restarting the app.	Ecosystem Maturity: Compared to more established languages like JavaScript, Dart's ecosystem is still evolving, with fewer third-party libraries and resources available.
Performance: Dart's optimized ahead-of-time (AOT) compilation and just-in-time (JIT) compilation enable Flutter apps to achieve high performance and smooth animations, comparable to native apps.	Limited Platform Support: While Flutter enables cross-platform development for iOS, Android, web, and desktop, there may be limitations in terms of platform-specific features and integrations.
Expressive Syntax: Dart features a clean and expressive syntax with features such as optional typing, async/await for asynchronous programming, and strong support for object-oriented programming paradigms.	

fig 2.6 advantages and limitations of Dart as programming language.

2.3.3) C# (C-sharp)

C# (pronounced as "C sharp") is a versatile and robust programming language developed by Microsoft as part of its .NET initiative. It has gained significant popularity and is widely used for various software development purposes, including hybrid mobile app development. When utilized in conjunction with frameworks like Xamarin, C# offers a comprehensive solution for building cross-platform mobile applications.

Advantages of using C#	Limitations of using C#
Cross-Platform Compatibility: C# allows developers to write code once and deploy it across multiple platforms, including iOS, Android, and Windows. This cross-platform compatibility streamlines the development process and reduces the need for separate codebases.	Learning Curve: While C# offers a relatively straightforward syntax, developers may encounter a learning curve when getting started with Xamarin and mobile app development. Familiarity with platform-specific nuances may also be required for optimal results.
Integration with Xamarin: Xamarin is a popular framework for building cross-platform mobile applications using C#. It provides access to native APIs, allowing developers to create native-like user experiences while leveraging their existing C# skills.	Platform-Specific Features: Despite Xamarin's support for cross-platform development, there may be limitations in accessing platform-specific features and functionalities. Developers may need to implement workarounds or platform-specific code to address these limitations.
Rich Ecosystem: C# benefits from a rich ecosystem of tools, libraries, and resources, supported by a vibrant community of developers. This extensive ecosystem facilitates rapid development and provides solutions for various development challenges.	
Performance: Hybrid mobile apps built with C# and Xamarin can achieve high performance and efficiency, thanks to Xamarin's ahead-of-time (AOT) compilation and just-in-time (JIT) compilation. This ensures smooth user experiences and responsive interfaces.	

fig 2.7 advantages and limitations of C# as programming language

3) REVIEW AND COMPARISON OF MOBILE APP DEVELOPMENT FRAMEWORKS:

This section evaluates popular mobile app development frameworks such as React Native, Flutter, and Ionic. Key features such as programming language, performance, cost & time to market, UX & UI capabilities, complexity, and community support are compared to assist developers in choosing the most suitable framework for their projects. We Shall do the evaluation of these frame works in a tabular form based on all the properties mentioned above. The table is as shown in the figure below

	React Native	**Flutter**	**Ionic**	**NativeScript**	**Xamarin**
Language	React Native utilizes JavaScript.	Flutter uses Dart as its programming language.	Ionic is based on web technologies such as HTML, CSS, and JavaScript/TypeScript.	NativeScript supports JavaScript and TypeScript.	Xamarin utilizes C# as its programming language.
Performance	React Native generally delivers good performance, thanks to its ability to render components directly to the native platform. However, performance can occasionally suffer due to the bridge between JavaScript and native components, especially when handling complex UI interactions or animations.	Flutter excels in performance, thanks to its compiled code, which eliminates the need for a JavaScript bridge. This results in smooth animations, fast rendering, and near-native performance.	Ionic may encounter performance issues at times due to its reliance on web technologies wrapped in a native shell. While performance has improved with recent updates, complex UI interactions or heavy processing tasks may still pose challenges.	NativeScript delivers good performance by directly accessing native APIs without the need for a bridge. This results in fast rendering, smooth animations, and near-native performance.	Xamarin offers near-native performance, as apps are compiled to native code. This results in fast execution, efficient memory usage, and smooth user experiences.

Cost and Time to Market	React Native significantly reduces development time and cost by allowing code reuse across platforms. This means developers can write a single codebase for both iOS and Android, saving resources and accelerating time-to-market.	Flutter reduces development time and cost by enabling code sharing between platforms. Developers can write a single codebase for iOS and Android, leading to faster development cycles and reduced expenses.	Ionic offers a relatively quick and cost-effective development process, thanks to its use of familiar web technologies. Developers can leverage existing web development skills to build cross-platform apps efficiently.	NativeScript can be cost-effective and quick, especially for teams with web development experience. Its ability to share code across platforms accelerates development cycles and reduces expenses.	Cost & Time to Market: Xamarin may incur higher costs due to licensing fees for certain features, such as Xamarin.Forms. However, its ability to share code between platforms reduces development time and accelerates time-to-market.
UI/UX	React Native provides a native-like experience to users and supports easy integration with native components. Developers can create visually appealing interfaces with smooth transitions and animations.	Flutter offers highly customizable and consistent UI across platforms, allowing developers to create beautiful and intuitive interfaces. Its widget-based architecture facilitates building complex UI designs with ease.	Ionic provides a native-like experience through its extensive library of UI components, allowing developers to create visually appealing interfaces. Its UI components are customizable and offer a consistent look and feel across platforms.	NativeScript provides native UI components for iOS and Android, ensuring a seamless and native-like user experience. Developers can customize UI elements to match platform-specific design guidelines and user expectations.	Xamarin provides access to native UI components and allows for extensive customization to meet design requirements. Developers can create visually appealing and user-friendly interfaces tailored to specific platform guidelines.
Complexity	React Native has a moderate level of complexity. While developers need a solid understanding of JavaScript,	Flutter has a moderate complexity level. While Dart might require some learning for developers unfamiliar with it, Flutter's	Ionic has a low to moderate complexity level. Developers with web development experience can quickly adapt to Ionic's framework and architecture, making it accessible to a broad audience.	NativeScript has a moderate complexity level. Developers need to have knowledge of JavaScript/TypeScript and native development concepts to build efficient and	Xamarin has a moderate to high complexity level. Developers need to be proficient in C# and have a solid understanding

	they also need to grasp some native development concepts to optimize performance and handle platform-specific features.	extensive documentation and rich set of tools help streamline the development process.		optimized applications.	of native development concepts to leverage Xamarin effectively.
Community Support	React Native boasts a robust and vibrant community, offering a vast array of libraries, plugins, and resources. This strong community support ensures developers have access to valuable tools and assistance when building React Native applications.	Flutter's community is rapidly growing, with enthusiastic developers and increasing adoption by companies like Google. This growing community ensures developers have access to support, tutorials, and plugins to enhance their Flutter projects.	Ionic boasts strong community support, with a large number of plugins, themes, and resources available. This active community ensures developers have access to valuable tools and assistance when building Ionic applications.	NativeScript benefits from strong community support, with a decent number of plugins, libraries, and resources available. This active community helps developers overcome challenges and enhances the development experience.	Xamarin benefits from strong community support backed by Microsoft, with a large number of resources available, including documentation, forums, and tutorials. This robust community ensures developers have access to valuable tools and assistance.
Where to use	React Native is ideal for cross-platform development of applications requiring frequent updates or rapid prototyping. It's suitable for a wide range of app types, including social media	Flutter is suitable for building cross-platform apps with high-performance requirements and complex UI designs. It's ideal for applications requiring a custom look and feel, such as gaming apps, media streaming platforms, and	Ionic is suitable for developing cross-platform apps with a focus on content-based applications or those requiring a rapid development cycle. It's ideal for building apps such as news platforms, e-commerce websites, and educational tools.	NativeScript is suitable for cross-platform development of apps requiring close-to-native performance and access to native APIs. It's ideal for applications such as productivity tools, utility apps, and business applications.	Xamarin is suitable for enterprises invested in the Microsoft ecosystem and requiring high-performance cross-platform apps. It's ideal for applications such as enterprise tools, financial applications, and healthcare solutions.

	platforms, e-commerce apps, and business tools.	productivity tools.			

fig 3.1 Review and comparison of mobile app development frameworks

4) STUDY OF MOBILE APPLICATION ARCHITECTURES AND DESIGN PATTERNS:

An in-depth examination of mobile application architectures and design patterns is conducted in this section. Various architectural paradigms such as monolithic, layered, microservices, and serverless architectures are explored, along with design patterns like MVC, MVVM, and Singleton. This knowledge equips developers with the necessary insights to design scalable, maintainable, and efficient mobile applications.

4.1) Mobile application Architectures

Mobile app architecture is the structural design and organization of a mobile application, outlining how various components and modules of the app are interconnected and work together to achieve its functionality. It serves as the blueprint for the development of a mobile app, defining the framework for building, maintaining, and expanding the application.

Mobile app architecture includes various layers or components, each with specific responsibilities, and it determines how data is processed, presented to users, and interacted with. It plays a critical role in the app’s performance, scalability, maintainability, and security.

Below are listed the most commonly use mobile app architectures

4.1.1) The Three Layered Architecture (layered architecture)

Layered architecture, also known as n-tier architecture, divides the application into multiple layers, with each layer responsible for specific tasks or functionalities. Common layers include presentation layer, business logic layer, and data access layer. In a layered architecture, each layer communicates only with the adjacent layers, following a strict hierarchy. This separation of concerns improves modularity, maintainability, and scalability of the application.

Key Components of Mobile App Architecture

1. User Interface (UI) Layer

The UI layer is responsible for the presentation of the app to the user. It includes the visual elements and components that users interact with, such as screens, buttons, forms, navigation menus, and any graphical elements. It manages the layout and appearance of the app. Common technologies used in the UI layer include UI frameworks, user interface libraries, and design tools that help create a visually appealing and responsive user experience.

2. Application Logic Layer (Business Layer)

The application logic layer, also known as the business logic layer, houses the core functionality of the app. It includes algorithms, business rules, and processes that control the app's behavior. This layer processes user input, orchestrates data retrieval and storage, and ensures the correct operation of the app's features.The application logic layer uses programming languages, software libraries, and frameworks specific to the platform (e.g., Android, iOS) to implement the app's functionality.

3. Data Layer

The data layer manages data storage, retrieval, and communication with external data sources. It includes databases, server APIs, and any data repositories that the app interacts with. This layer ensures data integrity, security, and availability. In the data layer, technologies like databases (SQL or NoSQL), RESTful or GraphQL APIs, and caching mechanisms are commonly used. Data management frameworks, Object-Relational Mapping (ORM) libraries, and data synchronization tools may also be part of this layer.

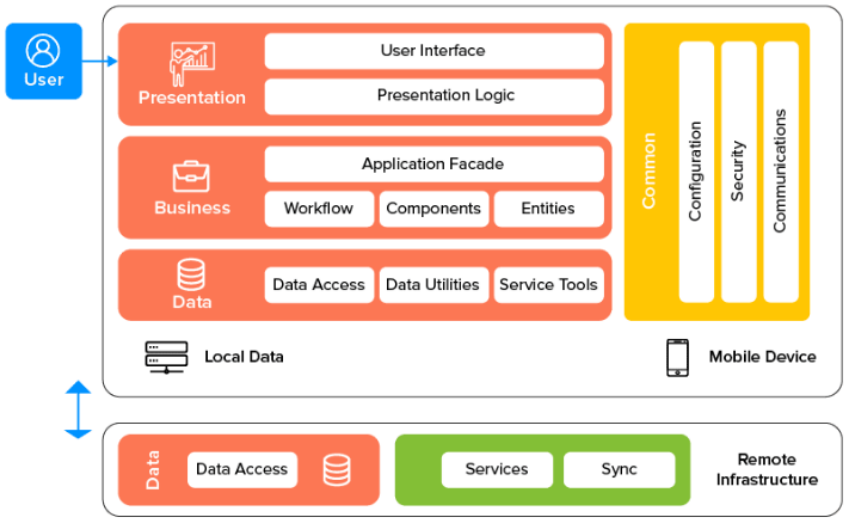


Fig 4.1: Representation of interaction between user and mobile app architecture in the three layered architecture src: https://www.octalsoftware.com/blog/mobile-app-architecture-guide *date: 30th march 2024*

Importance of Mobile App Architecture

The layered architecture matters in terms of efficiency and for ensuring the following:

Scalability

Scalability is essential because it ensures that an app can manage the growing number of users, a growing amount of data, and evolving requirements. As an app gains popularity, it must be able to expand without a significant loss in performance or user experience.

Maintainability

Maintainability plays an important role in the long-term success of a mobile app. As an app evolves, it must be easy to update, fix bugs, and adapt to new technologies. A maintainable architecture reduces the cost and effort required for ongoing development and support.

Performance

Slow or inefficient apps can deter users and lead to negative reviews. High-performing apps respond quickly to user interactions and provide a smooth and responsive experience. The architecture can impact performance through factors such as data retrieval methods, caching strategies, and the organization of code. A well-optimized architecture, combined with efficient algorithms and data management, leads to superior app performance.

Security

Security must be integrated into the architecture from the ground up. Secure communication protocols, data encryption, and access control mechanisms should be incorporated. The choice of architecture can also impact security; for example, microservices architectures may require additional measures to secure communication between services.

4.1.2) Monolithic Architecture

In monolithic architecture, all components and modules of an application are tightly integrated into a single, unified unit. In a monolithic architecture, the entire application, including the user interface, application logic, and data storage, is bundled together as a single codebase and runs within a single process.

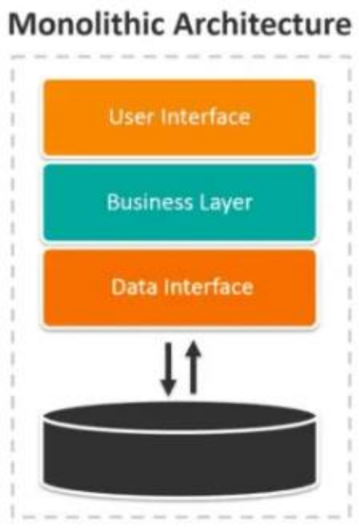


Fig 4.2: Block Representation monolithic mobile app architecture src: https://www.octalsoftware.com/blog/mobile-app-architecture-guide *date: 30th march 2024*

Key Characteristics of Monolithic Architecture:

Single Codebase:

In a monolithic architecture, all the code that makes up the application is part of one codebase.

Tight Coupling:

Components and modules within the monolith are tightly interconnected, meaning they are interdependent and often share resources and libraries. Changes to one part of the code can have ripple effects on other parts, making maintenance and updates more challenging.

Single Deployment Unit:

Monolithic applications are typically deployed as a whole. When you need to make changes or updates to the application, you redeploy the entire monolith, which can result in downtime during updates.

Shared Data Storage:

Data storage is centralized, and all components of the application share access to the same database or data store.

Advantages of Monolithic Architecture:

- Easier to develop, test, and initially deploy
- A wealth of knowledge and tools available for monolithic development
- Does not require communication between separate services

4.1.3) Microservices Architecture

In a microservices architecture, an application uses small, independent, and loosely coupled services that operate simultaneously to provide its functionality.

Instead of building a monolithic application where all features and functions are tightly integrated into a single codebase, a microservices architecture breaks down the application into a collection of individual services, each responsible for a specific set of tasks or functionalities.

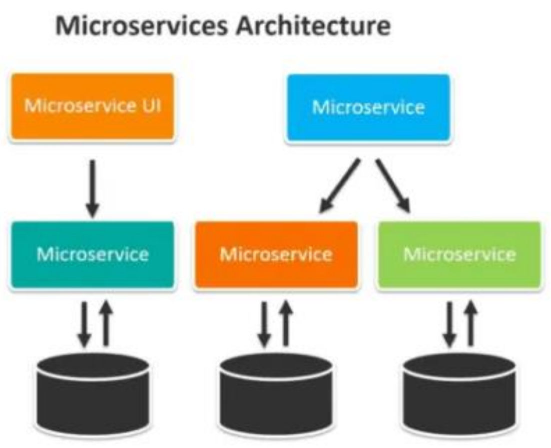


Fig 4.3: Block Representation microservices mobile app architecture src: https://www.octalsoftware.com/blog/mobile-app-architecture-guide *date: 30th march 2024*

Key Characteristics of Microservices Architecture:

Service Independence:

Each service is a standalone unit with its codebase, database, and possibly even its technology stack.

Loose Coupling:

Services communicate with each other through well-defined, lightweight interfaces such as APIs (Application Programming Interfaces).

Small and Focused:

Microservices are generally small in size and focused on a specific piece of functionality. Each service should perform a single task well.

Distributed Deployment:

Microservices are often deployed independently. This means that different services can be hosted on separate servers or containers.

Polyglot Technology:

In a microservices architecture, different services can use different programming languages and technologies based on the specific requirements of the service.

Independent Data Management:

Services may have their own data stores, which can be a database, key-value store, or other data storage solutions.

Advantages of Microservices Architecture:

- Microservices can be individually scaled up or down as needed
- Flexibility to choose the most appropriate technology for each service
- Smaller, focused teams can work on individual services, leading to faster development cycles and quicker time-to-market for new features.
- Failure in one service is less likely to impact the entire application
- Easier Maintenance

4.1.4) Model-View-Controller (MVC)

Model-View-Controller (MVC) is a widely used architectural design pattern for developing software applications, particularly in web and mobile applications.

It divides the application into three interconnected components, each with a specific role and responsibility. The primary purpose of the MVC pattern is to separate the concerns of an application, making it easier to develop, maintain, and extend.

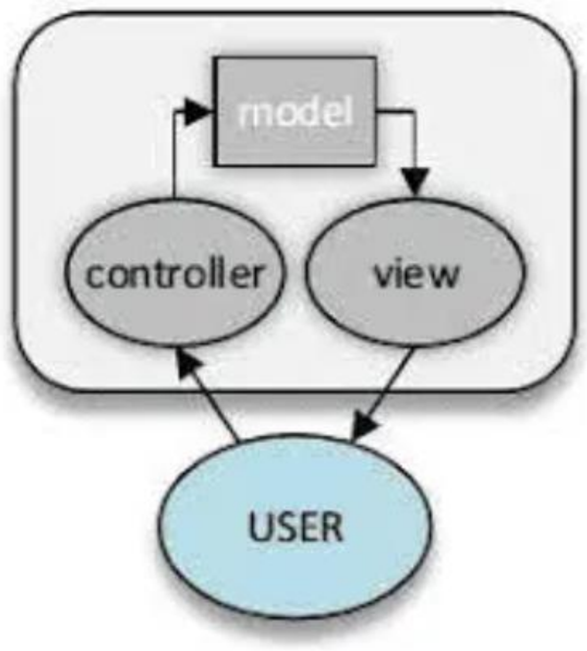


Fig 4.4: Block Representation MVCmobile app architecture src: https://www.octalsoftware.com/blog/mobile-app-architecture-guide date: 30th march 2024

Key components of the MVC architecture

1. Model:

It represents the data and business logic of the app. It encapsulates the core functionality and data management of the application. This component is responsible for data storage, retrieval, validation, and processing. It responds to requests from the Controller and notifies the View when the data changes.

2. View:

Its responsibility is to render the user interface and present data to the user. It displays the information from the Model to the user in a visually appealing and understandable format.

3. Controller:

It works like an intermediary between the Model and the View. It receives user input, processes it, and communicates with the Model to retrieve or update data. The

Controller decides which View should be displayed to the user based on the user's actions.

4.1.5) Model-View-ViewModel (MVVM)

Model-View-ViewModel (MVVM) is an architectural design pattern used primarily in software development for building user interfaces. MVVM is especially popular in modern mobile and web applications. It's an evolution of the Model-View-Controller (MVC) pattern, designed to enhance the separation of concerns and improve the testability and maintainability of code. MVVM consists of three main components: Model, View, and ViewModel, each with distinct responsibilities:

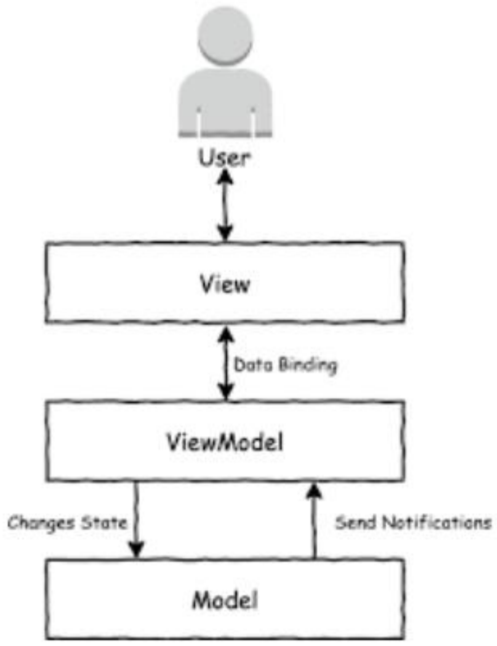


Fig 4.5: Block Representation MVVM mobile app architecture src: https://www.octalsoftware.com/blog/mobile-app-architecture-guide *date: 30th march 2024*

Key components of the MVC architecture

1. Model:

The Model represents the application's data and business logic. It is responsible for data storage, retrieval, validation, and processing.

2. View:
The View represents the user interface and is responsible for displaying data to the user. In MVVM, the View is the visual component of the application, such as a screen in a mobile app or a web page in a web application.

3. ViewModel:
The ViewModel serves as an intermediary between the Model and the View. It encapsulates the presentation logic, transforms the data from the Model into a format that is suitable for the View, and handles user interactions.

4.1.6) Clean Architecture
Clean Architecture is another popular mobile app architecture that emphasizes the separation of concerns, maintainability, and testability in software development.

It was introduced by Robert C. Martin, also known as Uncle Bob, and it provides a structured approach to building robust and adaptable software systems.

The core idea of Clean Architecture is to decouple different parts of the system and organize them in a way that prioritizes business logic and minimizes dependencies on external frameworks and technologies.

4.1.7) Model-View-Presenter (MVP)
MVP. The Model-View-Presenter architecture has been long used in other software development areas not only on mobile platforms. The principles behind this pattern were not designed from scratch and it came as a tour of MVC bringing in some advancements. This pattern can also be adapted to a large set of applications such as client/server or multi-tier applications [13]. The whole pattern is built with the idea that the actions in the application should be driven by the user interaction, by the view layer rather than by the controller. It is composed of three major types of components: the model that handles all the data, the view that takes care of the interaction with the user, the presenter that is responsible for connecting elements, MVC and MVP look very much alike and the di refences are subtle, that is why MVP comes as a avour of MVC and not as a new concept, both of them being presentational patterns. While they might look alike, there are differences and advantages in using one or another. In MVP

the presenter is responsible for manipulating the views and they communicate through interfaces; the views being decoupled from the presenters and vice versa. In the world of MVC, all the communication between the views and models is done through controllers, the elements are more tightly coupled. The controller receives an event from the view layer, it does some processing, it might manipulate the models and updates the views accordingly. Another diffrerence is the fact that in MVC the views are dumb objects, they do not contain processing code as contrary to what happens in the MVP pattern where the views have to communicate with the presenter.

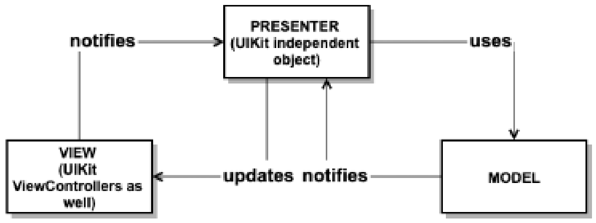

PRESENTER (UIKit independent object)
notifies
uses
VIEW (UIKit ViewControllers as well)
updates
notifies
MODEL

Fig 4.6: Block Representation MVP mobile app architecture src: A COMPARATIVE STUDY OF SOFTWAREARCHITECTURES IN MOBILE APPLICATIONS, STUDIA UNIV. BABES{BOLYAI, INFORMATICA, VolumeLXIV, Number 2, 2019DOI: 10.24193/subbi.2019.2.04 page 52

4.1.8) View Interactor Presenter Entity Routing (VIPER)

it is a software architecture used in large mobile applications. VIPER does not come from MV family [10], [16], [5], [18]. As shown in Figure 4 it uses layers of abstraction for separating concerns in the application. It does that for solving problems that come with using a classical MVC architecture where there is no clear layer where the business logic should be placed. VIPER respects the principles of a Clean Architecture [9] and it can be considered pattern for the whole application (not only a presenter pattern, like the previously described ones). software systems built using this architecture resemble a game of LEGO.A complete application is built from multiple VIPER modules, the size of those modules depending on the granularity sought. Each component has a well de- ned and single concern, this architecture is built on the Single Responsibility Principle. The view is only responsible for displaying the items it receives from the presenter. The presenter works closely with the interactor and prepares the content it receives from the interactor for the view so that this component can display it. The presenter is also responsible for reacting to events from the view and requesting new data from the interactor. The business logic is contained in the interactor; its responsibility is to manipulate the entity objects. All the logic should be independent on any UI components and all its behavior should be portable to other platforms. The entity layer contains the items with which the business logic works and it is related to the model in the MVC. The navigation from one view to another is shared between the presenter and an object which handles the navigation

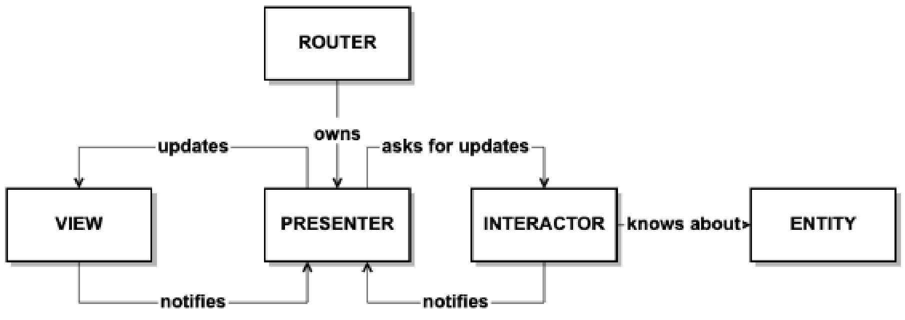


Fig 4.7: Block Representation VIPER mobile app architecture src: A COMPARATIVE STUDY OF SOFTWAREARCHITECTURES IN MOBILE APPLICATIONS, STUDIA UNIV. BABES{BOLYAI, INFORMATICA, VolumeLXIV, Number 2, 2019DOI: 10.24193/subbi.2019.2.04 page 54

4.2) Mobile application Design Patterns

Design patterns are general and reusable solutions to a common problem in software design. A pattern design is a description of the solution or a template that can be applied to solve the problem, not a piece of code that can be applied directly. In general, object-oriented patterns show relationships and interactions between classes or objects without specifying the final shape of the classes or objects involved.

4.2.1) Singleton Pattern:

The Singleton pattern ensures the existence of only one instance of a class throughout the application, providing a global point of access to that instance. This pattern proves useful in scenarios that require a single, shared resource. For example, a logger or a database connection should have a single instance to avoid redundancy and conflicts. To implement the Singleton pattern, we create a class with a private constructor and a static method that controls the creation and access to the instance.   Instead of global variables, a singleton is preferred to global variables because, among other things, it does not pollute the global namespace with unnecessary variables.

When to use

- Ensuring that a class has only one instance.
- Providing a global point of access to that instance.
- Guarantees that all code refers to the same instance.
- Useful when you want to limit resource usage, maintain consistency, or provide a shared service throughout the application.
- Can be applied to scenarios like logging services, database connections, or configuration managers.

4.2.2) Observer Pattern:

Design The Observer Pattern defines a relationship of dependency 1 to n between objects so that when an object changes its status, all its addicts are notified and updated automatically. Using this pattern involves the existence of a topic object that has a list of dependent objects as an observer, which it automatically calls every time an action takes place.

This pattern is behavioral (Behavioral), because it facilitates a better organization of classroom communication according to their roles/behavior.

The observer is used if several classes (observers) depend on the behavior of another class (subject) in situations like:

A class implements/represents logic, the basic component, and other classes only use its results (monitoring). A class performs actions that can then be represented in several ways by other classes. Practically in all of these situations, the Observer classes observe the changes/actions of the Subject class. The observation is implemented through notifications initiated from the methods of the Subject class.

When to use

- Establishing a one-to-many dependency between objects.

- Notifying dependents when the state of an object changes.
- Ensures automatic updates and synchronization between objects.
- Useful in scenarios such as real-time stock market updates, event-driven systems, or UI components that need to respond to changes in data.

4.2.3) Factory Pattern:

Factory patterns are used for objects that generate instances of related classes (implement the same interface, inherit the same abstract class). These are used when we want to isolate the object that needs a court of a certain type, to actually create it.

Additionally, the class that uses the instance does not need to specify exactly the subclass of the object to be created, so you do not have to know all the implementations of that type, but just what features the object has to create. For this reason, Factory is part of the Creational Patterns category because it provides a solution for creating objects.

Applicability:

-In libraries / APIs, the user is separate from the actual type of implementation and must use factory methods to obtain certain objects.

-When creating objects is more complex (several stages must be done, etc.), it is more useful to separate the logic needed to instantiate the class sub-type that needs that instance.

When to use

- Creating different instances of related objects based on conditions or parameters.
- Allows subclasses to decide which class to instantiate.
- Useful when you want to abstract the object creation process and decouple it from the client code.
- Can be used in scenarios such as creating different character classes in a game or generating different types of documents based on user input.

4.2.4) Dependency Injection (DI)

Dependency Injection is a powerful technique for managing dependencies between objects. It aims to decouple classes from their dependencies, making code more flexible, testable, and maintainable. Instead of creating their own dependencies internally, objects receive them from an external source. Dependency Injection is widely used in mobile app development. It helps manage dependencies between components, making code more modular and testable.

The Key Roles here include :

- Client: The component or class that depends on services provided by another class.
- Service: The component or class providing specific functionality.
- Injector: Creates instances of services and injects them into the client.
- Interface: Defines the contract that services must implement.

When to use

- Loose coupling and reusability.
- Testability (inject mock or test doubles).
- Maintainability and flexibility.
- E.g in java frameworks like Spring or JavaScript frameworks like Nest, In Android development, frameworks like Dagger use DI to inject dependencies into activities, fragments, and services.
- Decoupling components.
- Enhancing testability.

4.2.5) Adapter pattern

The Adapter pattern acts as a bridge between two incompatible interfaces, making them work together. It allows an existing class to be used as another interface. Imagine translating messages between friends who speak different languages. You act as an adapter, enabling smooth communication despite the language difference. The Adapter pattern is relevant in mobile apps when integrating with external libraries, APIs, or legacy code. It bridges the gap between incompatible interfaces.. Useful for integrating legacy code or third-party libraries into modern architectures.

The Key Roles here include :

- Target Interface: Defines the interface expected by the client.
- Adaptee: The existing class with an incompatible interface.
- Adapter: Implements the target interface and internally uses an instance of the adaptee.
- Client: Interacts with objects via the target interface.

When to use

- Adapting data from a REST API (with a specific format) to display in a RecyclerView.
- Converting data from a database query to match the expected format in a ListView.

4.2.6) Decorator pattern

The Decorator pattern dynamically adds functionality to an existing object without altering its original class. It wraps the original object, providing additional behavior. Abstract decorator classes or interfaces are used to implement the wrapper. Decorator classes keep the original class methods' signature unchanged. Promotes single responsibility principles by dividing functionality into classes with unique areas of concern. Imagine adding features (e.g., color, border) to shapes (circle, rectangle) without modifying their core behavior. The Decorator pattern is useful for adding features or behaviors to UI elements without modifying their core functionality.

5) REQUIREMENTS ENGINEERING

5.1) What is Requirement Engineering;

It is the process of eliciting, documenting, analyzing, validating, and managing the needs and expectations of stake holders for a software system. It is a crucial phase in software development that lays the foundation for the entire project.

To guarantee the effective creation of a software product, the requirements engineering process entails several tasks that help in understanding, recording, and managing the demands of stakeholders.

5.2) Tasks Involved In Requirement Engineering;

5.2.1) Feasibility study:

A feasibility study focuses on assessing the viability(ability to be successful and sustainable) and practicality of implementing the requirements of a proposed software project. Feasibility studies are conducted during the early stages of project planning to help stakeholders make informed decisions about whether to proceed with the project or not.

i. *Technical feasibility*: In Technical Feasibility current resources both hardware software along required technology are analyzed/assessed to develop the project. This technical feasibility study reports whether there are correct required resources and technologies that will be used for project development. Along with this, the feasibility study also analyzes the technical skills and capabilities of the technical team, whether existing technology can be used or not, whether maintenance and up-gradation are easy or not for the chosen technology, etc.

ii. *Operational Feasibility*: In Operational Feasibility, degree of providing service to requirements is analyzed along with how easy the product will be to operate and maintain after deployment. Along with this, other operational scopes are; determining the usability of the product, determining whether suggested solution by the software development team is acceptable or not, etc.

iii. *Economic Feasibility*: In the Economic Feasibility study, cost and benefit of the project are analyzed. This means under this feasibility study, a detailed analysis is carried out. The cost of development of the project which includes all required costs for final development; hardware and software resources required, design and development costs, operational costs, and so on. After that, it is analyzed whether the project will be beneficial in terms of finance for the organization or not.

iv. *Legal and Regulatory Feasibility*: This aspect assesses whether the proposed project complies with relevant laws, regulations, and industry standards. It involves identifying any legal or regulatory constraints that may affect the implementation or operation of the project and determining how they can be addressed.

v. *Schedule Feasibility*: Schedule feasibility evaluates whether the project can be completed within the desired timeframe. It involves developing a realistic project timeline, identifying critical milestones, and assessing potential risks and uncertainties that could impact the schedule.

In summary, a feasibility study is important in requirements engineering because it helps stakeholders identify and address feasibility issues early in the project lifecycle, make informed decisions, allocate resources effectively, assess the economic viability of the project, ensure alignment with organizational goals, and mitigate legal and regulatory risks. By conducting a feasibility study, organizations can increase the likelihood of project success and achieve better outcomes for their software development initiatives.

5.2.2) Requirements Elicitation:

It is the process of gathering information about the needs and expectations of the stake holders for a software system. This is the first step of the requirements engineering process and it is critical to the success of the software development project. The goal of this step is to understand the problem that the software system is intended to solve and the needs and expectations of the stakeholders who will use the system. The various sources of domain knowledge include; customers, existing software of the same type, stakeholders, business manuals etc. The techniques used for requirements elicitation include; interviews, brainstorming, task analysis, prototyping, Delphi techniques etc.

5.2.3) Documentation:

Once requirements are gathered, they need to be documented in a clear and unambiguous manner. This documentation serves as a reference for all stakeholders involved in the project and provides a basis for agreement and communication. Common documentation artifacts include requirement specifications, use cases, user stories, and prototypes. It is important to document, organize and prioritize requirements gathered from all these techniques to ensure that they are complete, consistent and accurate.

5.2.4) Requirements specification:

It is the process of documenting the requirements identified in the elicitation step in a clear, consistent, and unambiguous manner. The goal of this step is to create a clear and comprehensive document that describes the requirements for the software system. This document should be understandable by both the development team and the stakeholders. This activity is used to provide formal software requirement models. All the requirements including the functional as well as non-functional requirements and the constraints are specified by these models in totality. During specification, more knowledge about the problem may be required which can again trigger the elicitation process. The models used at this stage include; ER diagrams, data flow diagrams, function decomposition diagrams, data dictionaries, etc. Several types of requirements are commonly specified in this step, including;

i. *Functional Requirements:* This describes what the software system should do. They specify the functionality that the system must provide, such as input validation, data storage, and user interface.
ii. *Non-Functional Requirements:* This describes how well the software system should do it. They specify the quality attributes of the system, such as performance, reliability, usability, and security.
iii. *Constraints:* This describes any limitations that must be considered when developing the system.

iv. *Acceptance Criteria*: This describes the conditions that must be met for the software system to be considered complete and ready for release.

Once these requirements are specified, they must be reviewed and validated by the stakeholders and development team to ensure that they are complete, consistent and accurate.

5.2.5) Requirements Verification and Validation:

Requirements verification and validation is the process of checking that the requirements for a software system are complete, consistent and accurate and that they meet the needs and expectations of the stakeholders. The goal is to ensure that the software system being developed meets the requirements and that it is developed on time, within budget and to the required quality.

Verification: It refers to the set of tasks that ensures that the software correctly implements a specific function.

Validation: It refers to a different set of tasks that ensures that the software that has been built is traceable to customer requirements. If requirements are not validated, errors in the requirement definitions would propagate to the successive stages resulting in a lot of modification and rework.

Verification and Validation is an iterative that occurs throughout the software development life cycle.

5.2.6) Requirements Management:

Requirement management involves organizing, tracking, and controlling changes to the requirements throughout the software development lifecycle. This includes establishing baselines, maintaining traceability between requirements and other project artifacts, and handling change requests effectively. Requirement management ensures that; the project stays on track, the software meets the stakeholders' needs and that it is developed on time, within budget and to the required quality.

5.3) Advantages of Requirements Engineering

- Helps ensure that the software meets the needs and expectations of the stakeholders
- Can help identify potential issues or problems early in the development process, allowing for adjustments to be made before significant errors at successive stages
- Helps ensure that the software is developed in a cost-effective and efficient manner
- Provides a solid foundation for the development process, which helps to reduce risk of failure

5.4) Disadvantages of Requirements Engineering Process

- Can be time-consuming and costly, particularly if the requirements gathering process is not well-managed
- Changes in requirements can lead to delays and increased costs in the development process.

6) STUDY OF ESTIMATING MOBILE APP DEVELOPMENT COST:

6.1) Introduction

The final section of the report focuses on the challenging task of estimating the cost of mobile app development. Factors influencing development costs, estimation techniques, and strategies for managing and optimizing project budgets are explored, providing valuable insights for project managers and stakeholders.

App development cost estimation is what sits at the very bottom of any mobile app development budget. It is not easy to jump on a specific number during the estimation, but here we will guide you towards a better way of summing up all the costing aspects of mobile app development.

Mobile app development is a challenging yet exciting venture. Entering the app development endeavor comprehends a thoughtful plan about your organization, your business goals and your app features. The formula for app development is the number of hours multiplied by the hourly rate.

The majority of mobile app development companies lend an app development cost estimate so that you have a rough idea of the overall cost of getting an app developed.

The simple formula for calculating mobile app development cost is as follows:

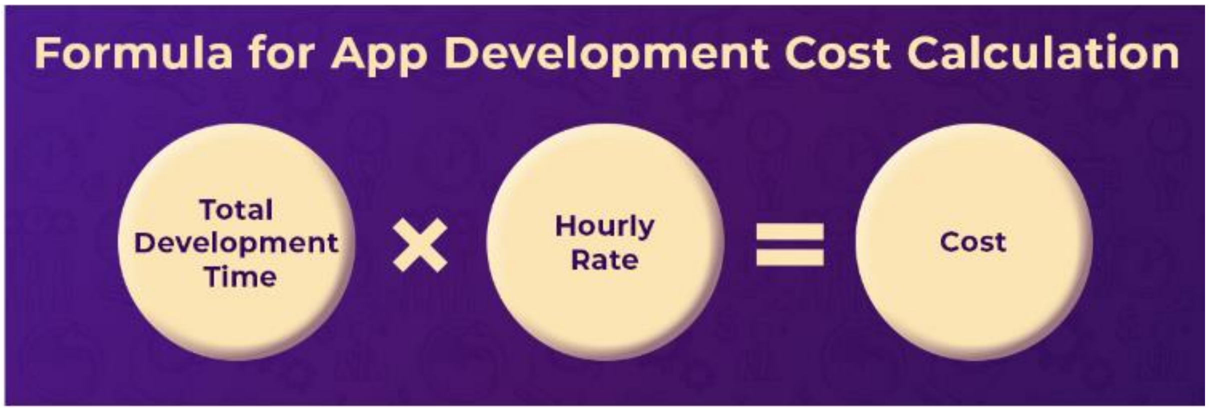


Fig 6.1: formula for app development cost src: https://www.topdevelopers.co/blog/app-development-cost/#whom-should-you-hire-for-mobile-app-development-project-requirement *date: 30th march 2024*

6.2) Factors influencing mobile app development

6.2.1) Choosing Your Technology Partner

Business owners are often in turmoil while choosing a technology partner for their app development project. Since it heavily impacts the cost of application development, it should be a well-thought decision. Choosing the best mobile app development company gives you an edge over freelancers as they have much more to offer in terms of technology and experience. On the other hand, hiring freelancers can cut down your typical app development costs to a certain extent.

Pro tip: Establish a communication process that works with both organizations' processes. Based on this information, you can determine the cost as well as the best allocation of project resources.

6.2.2) Demographic Location of an App Development Company for Hire

The demographic location of the offshore mobile app development company, where your app project is going to be developed, plays a vital role in understanding how much it will cost to develop an app. Moreover, the location of the IT outsourcing company and remote app developers also matters in the overall budget of your app. This is because every country or region has its own terms and legal policies for doing business. Their currency expenses differ and so does their market. Hence, how much it costs to make an app, will also differ. Moreover, the packages offered by a mobile app development agency and freelancing development team also differ. Most of them charge according to the hourly rates, convey how many pages or app functionalities they will build or integrate.

6.2.3) Hiring Models for Mobile App Development Companies

Generally, the mobile app development clients are charged based on the below-mentioned models for mobile app development:

Time and material

Time and material is an elaborate pricing structure that is set on the basis of time and material required for an app development project. It is the best bet for ongoing and complex projects as it gives clients the flexibility of calculating pricing on hourly rates too.

Fixed cost

A fixed charge is a payment structure that is determined by the cost charged for a particular time period. It works best for small app development projects with a well-defined and clear scope of work.

Hourly rate

This hiring model for mobile app development services includes cost based on dedicated app developers' time taken to complete the project requirements. It can vary depending on various regions and the requirements of the mobile application.

6.2.4) Scope and Technology of your app

The scope of app development simply apprises you of the things that should be done to get certain desired outcomes. In other words, if the project scope increases, it leads to increased price as well as time. Ask yourself what are the necessary elements you want to have in your Minimum Viable Product (MVP) and how much of the product you still have to get developed if it's a project on upgrading an application. The technologies used in mobile application development have grown so immensely not because of the number of development platforms but because of the nearly limitless ways, it magnifies our lives.

6.2.5) Features, Functionalities and Complexity of an App

The cost of app development varies widely based on its app features and functionalities. The simpler and smaller the application, the less time you will need for its development, and the cost to develop such mobile apps will be lower. You need to decide on the scale of the upcoming work when it comes to the timing of the app development. The complexity level of your app depends on the features and functionalities you choose to integrate. Here are the three basic levels of app complexity and their estimated app development quote:

Simple apps- Simple app development costs approx $20,000 to $50,000 for one platform,

Mid-level apps- Mid-level apps cost around $50,000 to $150,000 per platform, For instance, Shopify, McDonald’s

Complex apps- Complex apps cost more than $150,000, For instance, Uber eats, Snapchat

6.2.6) App Development Platform to Cater

When it comes to app development, there are two options native and hybrid. The development platform for mobile applications can be chosen based on the business model. It has to be made available if the service demands an app on major platforms such as iOS and Android. Although these days there is no difference in application development cost based solely on the platform, it varies depending on the features and complexity of the app.

Android apps

Android apps cover the Google play store keeping in mind devices such as android smartphones, tablets, smart TV and much more. Android app development cost differs as expert Android app developers will need to adapt to different screen sizes when it comes to Android OS.

iOS apps

iOS apps on the other hand are made solely for Apple devices such as Apple iPhone, iPad, Macintosh, iWatch and iPod. iOS development cost differs as they are developed keeping a predetermined audience in mind. With Apple backed solution, iOS app development provides a variety of bug-fixing tools and supports the entire range of iOS devices.

Cross-platform apps

In order to cash out on both Android and iOS markets, business owners often go for a single cross-platform application instead of two individual platforms. While it seems a jack of all trades, it does reduce development costs with reusable code and a unified interface for both platforms.

6.3) App Development Cost Based on Type of App

The cost of app development is directly affected by the app type and the features that are included in the application. For instance, if you are developing a simple app with subtle functions, the app pricing will be less expensive than an app with extensive modern features.

Moreover, based on the services and the features, there are various classifications of mobile applications as follows:

- Cost to develop On-Demand Apps – $70,000 – $150,000
- E-Commerce App Development Cost – $60,000 – $300,000
- Cost to develop Data-Driven Apps – $15,000 – $35,000
- Authentication App Development Cost – $40,000 – $80,000
- Cost to develop Social Media Apps – $50,000 – $300,000

6.4) Mobile App Development Process with Cost Breakdown

Apart from the key aspects, the subsidiary factors that account for adding mobile app development expenses are in the app development procedure itself. Here is the detailed cost breakdown of the mobile app development process:

6.4.1) Discovery Phase

At the very beginning of mobile app development, there is always the discovery phase that includes in-depth research regarding the application market. The discovery stage includes all the groundwork such as analyzing the market, studying the target audience and gathering industry insights.

You need to decide on the app project requirements and the number of funds allocated for its development. Along with that, the desired development timeline, pricing models as well as the app building platforms to run it and much more. Within a timeline of two to four weeks, the average cost of the discovery phase for app development processes ranges from $10,000 to $15,000.

6.4.2) Mobile App Design Phase

The mobile app competition is fierce with over a million apps in the market. The first and foremost aspect people notice about an app is how it looks. Even before the user downloads the application to test it and see how it functions, they certainly see its design. Additionally, UI (User Interface) and UX (User Experience) are essential to make a unique application. Moreover, when it comes to the app designing process, minimalism is the key trend attracting users currently. Designing for simplicity makes the app look modern and chic. It is tough to create a subtle yet unique UI/UX design and it will have certain additional costs. Its components such as graphic images, icons, push notifications and other design customizations need to be added to your mobile app. The app design costs around $5,000 for an average of 20 basic app screens.

6.4.3) Mobile App Development Phase

The most important step in the entire application development process is the development phase itself. The mobile app development cost estimation varies based on its purpose and the value proportion with the core business function including the steps to create a mobile app. The app development phase includes demonstrating a supportable, producible, interoperable and affordable system in an intended environment.

This phase has two major efforts, system integration and system demonstration. It includes a lot of documentation built upon the necessary details to design the proposed system. The app development project enters a phase when an affordable increment of useful capability has been identified, demonstrated in the relevant environment, and can be developed within a short timeframe.

6.4.4) Mobile App Testing and Deployment Phase

Testing and deployment is just another hurdle that every application has to go through. Testing the app and its features simply ensures a hassle-free service. As the complexity of the app increases, the time taken to test and debunk extends further, thus, the price of mobile app development increases as well.

For instance, a payment application lends various security features along with a navigation interface, a food delivery app comes along with multiple service integrations such as real-time status, payment gateway, communication and location. Therefore, the price entailed in the app development process increases as the feature advances in an application.

If we calculate mobile app development cost, it takes up around 30% of the overall cost of building a mobile app spent on the client side and around 10% on the admin panel.

6.4.5) Mobile App Maintenance and Support

It is no secret that mobile applications require time to time maintenance and application updates. Along with that, the app developers need adequate technical assistance for the application to function smoothly. Enterprises need to revamp the app functionalities and integrations as the app platform transforms into advanced technologies.

The periodic app maintenance and support is a lifelong process that is an important factor to refine mobile apps persistently. It also improves the user experience as the applications need to maintain their APIs regularly too. Along with support and maintenance, the cost for app security never goes in vain.

6.5) Whom you should Hire for Mobile App Development Project Requirement

One of the most significant factors of app development cost estimation is choosing whom to hire to create your brainchild. At this point, you can choose between hiring an IT company or freelancers or hiring an in-house team altogether. You can choose between the options as follows:

6.5.1) Hire In-house App Development Team

An expert in-house app development team to create your application is a great solution. A viable option in mobile apps for business owners is that if they already have a thriving business organization or can afford to have an experienced in-house IT team to do the job right.

The costs of assembling such a team can be prohibitive as recruiting and onboarding new hires is time-consuming. On the other hand, you do get to have sole control over the entire process of app development.

6.5.2) Hire Freelance App Developers

Hiring freelance mobile app developers are a great fit when it comes to small-scale app development projects, product fixes or minor upgrades. You save up to 40-50% of the overall cost for app development by hiring freelancers from major online platforms such as TopDevelopers.co, Upwork, Linkedin and many more.

Although hiring freelance app developers is a bit of a gamble, you can easily monitor the progress of their work and impact on the app development project at each stage.

6.5.3) Hiring Outsourcing Mobile App Development Companies

Between standing up your own in-house team and taking a risk working with a freelancer, outsourcing to a mobile app development company offers a middle ground. You are not limited to the project size with mobile app development companies. Not to mention the added benefits of access to new tools and technologies as well as professional expertise.

When you outsource your app development requirements to an experienced mobile app development company it benefits in a lot of ways such as:

- Brings wide range of skills
- Puts forward wealth of experience
- Reduces project costs
- Enhances product quality
- Shortens time to market
- Gives expert insights
- Integrates modern technologies
- Provides qualified assistance

Closing Thoughts on App Development Costs

Last but not least, you will need a credible estimate of the cost to develop an app for the sake of your budget whether you are just a startup or a major enterprise. An experienced mobile app development service provider will be able to customize app development costs according to your needs. Although, it is worth noting that each project is unique and hence, mobile app development cost differs

Conclusively, the best way to estimate the app development cost based on your project requirements is to reach out to the top app development agencies and discuss your project.

REFERENCES

1) https://en.m.wikipedia.org/wiki/Mobile_app# date: 29th march 2024
2) https://www.cs.ubbcluj.ro/~studia-i/journal/journal/article/view/43/43 date: 29th march 2024
3) https://www.mvps.net/docs/design-patterns-singleton-factory-observer/ date: 29th march 2024
4) https://medium.com/@stheodorejohn/exploring-javascript-design-patterns-c257aa261550 date: 29th march 2024
5) https://www.geeksforgeeks.org/dependency-injectiondi-design-pattern/ date: 29th march 2024
6) https://www.topdevelopers.co/blog/app-development-cost/#whom-should-you-hire-for-mobile-app-development-project-requirement date: 29th march 2024
7) https://www.octalsoftware.com/blog/mobile-app-architecture-guide date: 29th march 2024
8) https://www.spaceo.ca/blog/best-mobile-app-development-languages/ date: 29th march 2024
9) "Requirements Engineering in Software Engineering" geeksforgeeks, https://www.geeksforgeeks.org/software-engineering-requirements-engineering-process/ date 29th march 2024