REPUBLIQUE DU CAMEROON
PAIX-Travail-Patrie
MINISTRE DE L'ENSEIGNEMENT SUPERIEUR

FACULTE DE L'ENGINERIE ET TECHGNOLOGIE

REPUBLIC OF CAMEROON
Peace-Work-Fatherland
MINISTER OF HIGHER EDUCATION

FACULTY OF ENGINEERING AND TECHNOLOGY

********************* UNIVERSITY OF BUEA ************************

DEPARTMENT OF COMPUTER ENGINEERING

COURSE: INTERNET PROGRAMMING (J2EE) AND MOBILE PROGRAMMING

COURSE CODE: CEF 440

TASK 1

GROUP MEMBERS

S/N	**Name**	Matricule Number
1	DUESENBERRY MBIKANG AGBORTAR AKO	FE21A175
2	FOTABONG FUALEFAC PUISSANCE	FE21A195
3	ABANGMA ARRAH JESUS MCLOUIS	FE21A124
4	TABOT JOEL EBANGHA	FE21A308
5	EBOMESUMBE NGOLE DANDY BRADLEY	FE21A177

INSTRUCTOR: DR. VALERY

JANUARY 2024

Table Of Contents

SECTION 1: REVIEW AND COMPARE THE MAJOR TYPES OF MOBILE APPS AND THEIR DIFFERENCES (NATIVE,PROGRESSIVE WEB APPS, HYBRID APPS)

Here's a breakdown of the

1.1. Major mobile app types

1.1.1. Native Apps:

Built specifically for a particular operating system (OS) like Android or iOS.

Advantages:

- Fastest performance and smoothest user experience.
- Full access to device features like camera, GPS, etc.
- Generally more visually appealing and intuitive.

Disadvantages:

- Requires separate development for each platform (Android and iOS), increasing cost.
- Updates must be submitted to the respective app store (Google Play Store or Apple App Store) for approval.

1.1.2. Progressive Web Apps (PWAs):

Essentially websites designed to function like native apps.

Advantages*:*

- Work across any mobile device and operating system with a web browser.
- No installation required, users access through a web address.
- Updates happen automatically without needing app store approval.

Disadvantages:

- May not have the same level of performance or functionality as native apps, especially regarding offline use.
- Limited access to some device features compared to native apps.

1.1.3. Hybrid Apps:

A combination of native app and web app elements.

Built using web technologies like HTML, CSS, and JavaScript but wrapped in a native app container.

Advantages:

- Faster and more cost-effective development compared to building separate native apps.
- Can access some device features like native apps.
- Can work across multiple platforms with some adjustments.

Disadvantages*:*

- Performance may not be as smooth as native apps.
- Limited access to some device features compared to fully native apps.

1.2. Differences between the different types

1.2.1. Web apps vs. Native apps

The term web app indicates an app you can access from the browser of a desktop or mobile device. The term native app suggests an app you can download and install on your device. A native mobile app is developed specifically for a mobile device. The terms native app, native mobile app, and mobile app are often used interchangeably to refer to the same type of software.

Some key differences between native apps and web apps are given below.

1.2.1.1 Functionality

Web apps only give users access to interactions supported by web browsers. Even though a web application has rich design elements, it cannot access device features. Native mobile apps, on the other hand, let users interact with their devices' internal hardware and operating systems. You can grant users access to native features like:

- Device location tracking
- Device microphone and cameras
- User contact lists
- Touch gestures, device tilt, and other user interactions
- Device security features like a fingerprint scan or face recognition

1.2.1.2. User experience

Web apps lack consistency in user experience due to their heavy dependency on browsers. Certain features or images may look different on different browsers. Buttons and menu bar features may be challenging to access from mobile browsers. Browser window resizing may impact the look, feel, and functionality of the web application.

Users tend to have a better experience on native mobile apps. For instance, the native app fills the screen and takes control of the entire device. Users get more out of the native app because they are comfortable with the interactions. The native app can also send push notifications to users and get them to re-engage.

1.2.1.3. Performance

Native applications give better performance when compared to web apps. They are faster, more responsive, and more interactive. However, the onus is on the user to maintain the native app performance. The user must download and install regular software updates to keep the app running optimally. Web applications are slower and less responsive, but they give you more control over performance. Software updates benefit all users immediately.

1.2.1.4. App development

Web apps are comparatively simpler, cheaper, and faster to develop. Time to market is shorter because of a straightforward app development process. They are also easier to maintain because you only have to test and update a single codebase. Native apps require a heavier financial investment. They also need development teams with cross-platform development experience. For instance, a developer specializing in native iOS apps may not be the best choice for building native Android apps.

1.2.1.5. Customer reach

Web apps have limited customer reach, as users require an internet connection to access the app. In the case of mobile web apps, there is a multi-step access process because users first have to open the mobile browser then find the app. Conversely, you can design native apps to work offline on the user's device. Native apps also give more discoverability because they are on app stores. You can run marketing campaigns within the app store to reach a wider or newer customer base.

1.2.2. Native apps vs. hybrid apps

A hybrid app is a particular type of native app. Like native apps, users can download and install a hybrid app from app stores. However, the internal structure of native and hybrid apps is very different. Internally, hybrid apps are more like web apps. Hybrid apps lie somewhere between native and web apps.

1.2.2.1. App development

In a native app, your developers have to rewrite and redesign all the app functionality in the native development language. A hybrid app lets you write the app functionality in a single codebase. You can then wrap your code in a lightweight native app shell or container. The container enables you to take advantage of native features in your mobile devices, like hardware, calendars, and notifications.

1.2.2.2. Cost efficiency

Hybrid apps achieve the same performance and user experience as native apps at a lower cost. Your developers can build them using commonly used app development languages and technologies like JavaScript, CSS, and HTML5. They can then integrate them with hybrid app development frameworks like

Ionic, Cordova, or React Native. Both time and cost of development are lower, but you can still upload them to an app store to enjoy the same reach and discoverability.

1.2.3. hybrid apps vs. progressive web apps

Progressive web apps are the result of advancements in browser technologies. Modern browsers let you give your users a native-app-like experience from the web app itself. You can achieve this by integrating a JavaScript framework around your existing web app. Your progressive web app can send notifications via the mobile browser, track user location, and so on. Like hybrid apps, progressive web apps also lie between native and web apps. However, there are some key differences.

1.2.3.1. Organic reach

You can deliver both progressive web apps and hybrid apps from app stores. However, progressive apps rank higher in search engine results with no additional effort. In addition, you get better search results than hybrid apps with the same keyword targeting.

1.2.3.2. Performance

In most cases, progressive web apps tend to be lighter in size than hybrid apps. They utilize less mobile storage and memory. However, the underlying technologies are non-native. Browser dependency could result in increased mobile battery consumption for users.

1.2.3.3. Maturity

Progressive web app technology is relatively new compared to hybrid or web apps. Consequently, developer and community support for progressive apps are still evolving. Hybrid app technology is more mature, and development remains less expensive.

SECTION 2: REVIEW AND COMPARE MOBILE APP PROGRAMMING LANGUAGES

2.1. Definition of Mobile App Programming Language:

A mobile app programming language is a programming language used to develop applications specifically for mobile devices such as smartphones and tablets. There are multiple programming languages used to develop mobile apps, some of which are android specific, some iOS specific and some which can be used for both.

2.2. Programming Languages which are iOS Specific:

The iOS platform is a proprietary platform made by Apple. You can develop apps for the iOS platform, and then target the same app to both an iPhone and an iPad. Apps can be built for iOS devices either by using the native iOS SDK with Objective-C and Swift or with the various cross platform technologies that are written against the SDK of that framework, but targeted for iOS.

2.2.1. Objective C:

Was the first language supported by Apple for developing iOS mobile apps. It is an objected-oriented (OO) language that derives the language syntax from C and the OO-ness from SmallTalk. One of the common criticisms of the language is that its syntax seems clunky being verbose and the square brackets are hard to debug. It's a stable and mature language, however, as it has had several years of wide usage. Since Apple introduced Swift, the popularity of Objective-C has declined considerably for new iOS mobile development.

2.2.1.1. Key Features of Objective C

- Original iOS language
- Stable and reliable programming language
- Strong developer community and documentation
- Has complex syntax that is more difficult to learn

2.2.1.2. Advantages of Objective C

- Interoperability with C and C++: Objective-C is a superset of the C programming language, which means it can directly use C and C++ libraries and frameworks. This interoperability allows Objective-C developers to leverage existing codebases, libraries, and tools written in C or C++.
- Mature and Stable: Objective-C has a long history and has been used for iOS and macOS app development for many years. It is a mature and stable language with a large community of developers and extensive documentation. This stability can be advantageous when working on large-scale projects or maintaining legacy codebases.
- Access to Apple's APIs: Objective-C has been the primary language for iOS app development prior to the introduction of Swift. As a result, it has extensive support for Apple's APIs and frameworks. Objective-C developers have access to a vast ecosystem of libraries and documentation built around the iOS platform.
- Easy Integration with Swift: Objective-C and Swift are interoperable, which means you can write new code in Swift while still using existing Objective-C code. This flexibility enables developers to gradually migrate their codebase from Objective-C to Swift if desired, rather than having to rewrite everything from scratch.
- Availability of Learning Resources: Objective-C has been around for a long time, and as a result, there are numerous learning resources available, including books, tutorials, and online courses. This wealth of resources can be beneficial for developers who are new to iOS app development and want to learn from established materials.

2.2.1.3. Limitations of Objective C

- Verbose syntax: Compared to modern programming languages, the syntax of Objective-C is verbose and complex to use. Standards such as square bracket notation for a method call and manual memory management reduce code readability and increase code verbosity.
- Less Industry Trend: With the arrival of Swift as Apple's preferred language for new development, the current industry trend is shifting away from Objective-C. This could potentially

result in job opportunities and career growth for those developers who specialize solely in Objective-C.

- Manual Memory Management: Objective-C relies on manual memory management via retain-release cycles, which can be error-prone and result in memory leaks and crashes if not handled carefully. Developers are forced to manage memory explicitly while using Objective-C, which increases complexity.
- Limited Safety Features: Objective-C doesn't have the safety features that modern languages such as Swift have. The absence of nullable checks compiles time results in runtime application crashes. Developers may face type-related issues due to its dynamic support feature.
- Learning Curve: Objective-C has a deep learning curve, especially for developers who are new to its syntax and concepts such as pointers and manual memory management. Objectice-C's dynamic nature may require additional time and effort for developers to understand. *"source (geeks)"*

2.2.2. Swift:

Swift is a beginner-friendly yet powerful iOS programming language designed by Apple to replace Objective C for its native app development. It was introduced by Apple in 2014 as a modern, safe, and powerful language to replace Objective-C. Swift is designed to work seamlessly with Apple's Cocoa and Cocoa Touch frameworks, making it an excellent choice for developing iOS, macOS, watchOS, and tvOS applications. Swift offers a clean syntax, type safety, memory safety, and performance optimizations.

2.2.2.1. Key Features of Swift

- Loved more than Objective C in iOS development
- Better speed and performance than Objective C
- Easy to learn and read language
- Error handling mechanisms
- Capable for creating complex apps

2.2.2.2. Advantages of Swift

- Scale Your Staff and Product with Ease: You may create a product that is future-proof with Swift. It may be enhanced with a variety of additional features. You'll discover that Swift projects scale much more easily. Swift would let you expand your Swift development company or development team by a number of programmers. Onboarding becomes easy because of the clear and straightforward codebase. Swift uses natural Language and is very simple to read.
- Compatibility with Objective-C: Swift is designed to work alongside Objective-C, so developers may utilize both languages in the same project. This is advantageous since it allows developers to reuse Objective-C code and libraries in Swift applications and vice versa. This simplifies and lessens the transition from Objective-C to Swift.
- Security: Swift is not only a very effective and quick solution, but it also provides great security. Programmers may quickly and easily identify and fix problems with this new approach to coding. As a consequence, the code is clear and readable. Its typing interface reduces the likelihood that errors would go undetected.
- Improved Memory Management: Swift provides an automated reference counting (ARC) system that automatically controls memory allocation and deallocation. This eliminates the need for

developers to manually manage memory, which may be time-consuming and error-prone. ARC also aids in the reduction of memory leaks and crashes caused by memory-related problems.

- Rapid Development: Swift is a clear language with a straightforward syntax that is easy to understand and write in. Moreover, semicolons are not required. International languages and emojis are supported. The management of memory utilization requires a lot of time and effort from developers (There are memory leaks that need to be handled). As a result, iOS app development is made quicker. Moreover, Swift is a value type for frequently used types like arrays and dictionaries, so there is no concern that it will be updated elsewhere, increasing your development speed.

2.2.2.3. Disadvantages of Swift

- Cross-Platform Support in Full: Swift may only be used in applications that run on iOS 7 and later. Nevertheless, because of its design, Swift cannot be utilized in legacy applications that employ earlier operating system versions. Yet, fewer than 5% of Apple devices now use iOS 8 or an older version, according to the findings of a recent survey. Cross-platform support that is lacking Swift is compatible with Linux, Windows, and all Apple systems, as we already said.
- Limited Third-Party Libraries:Swift is a relatively young programming language that is not as well-known as other high-level programming languages such as Java or Python. This implies that Swift has fewer third-party libraries and frameworks available, which might make development more difficult. But, this is changing as Swift’s popularity grows and more developers and businesses embrace the language.
- Issues with the Version Compatibility: Due to frequent language changes in its more recent versions, Swift is prone to version compatibility issues. Consequently, upgrading to a more recent version of Swift might provide serious difficulties, such as the need to fully rewrite the developer’s project’s code. To solve this issue and substantially simplify code transfer from one version to another, Swift developers created the Swift Conversion Tool for XCode. *“Source (directions)”*

2.2.3. Comparison between objective C and Swift

Comparison Criteria	**Objective-C**	**Swift**
Syntax	Syntax in Objective-C can appear to be verbose compared to other programming languages. It requires method calls to be enclosed in square brackets, which might feel unfamiliar to developers.	Swift code is cleaner and more concise compared to Objective-C. Method calls can be made using dot operators, the same as in most modern programming languages.
Readability	Pointers and memory management require explicit handling, which improves the code quality.	Features such as optionals, type interfaces, and closures help in enhancing code and reducing boilerplate code.
Performance	The performance here is reliable, benefiting from years of optimization and refinement. Features such as dynamic typing	It’s designed to be fast and efficient, leveraging modern compilers and the latest language features. Offers

	and message passing can result in an overhead in performance	improved performance over Objective-C in many cases.
Safety	Objective-C developers need to manage memory manually through the retain-release cycle, which increases memory leaks and crash occurrences. The absence of nullability enforcement by the compiler can result in a runtime crash.	It prioritizes safety and reliability by incorporating features like optionals and strong typing. The compiler performs rigorous type checks, which helps catch errors at compile time itself.
Interoperability	Code written in Objective-C can be easily integrated into Swift projects by using bridging headers, enabling incremental migration from Objective-C to Swift.	Swift can make calls to Objective-C APIs directly and vice versa, allowing easy collaboration between legacy and modern codebases.
Community and Ecosystem	Due to its longer history, Objective-C has a larger existing codebase and ecosystem, but new development is shifting rapidly towards Swift.	Benefits from a modern and enthusiastic community and continued support and updates from Apple. Also, Swift's open-source nature has led to cross-platform adoption beyond Apple's ecosystem.

2.3. Programming Languages which are Android Specific

2.3.1. JAVA:

Java was the default language to write Android apps since the Android platform was introduced in 2008. Java is an object-oriented programming language that was originally developed by Sun Microsystems in 1995 (now, it is owned by Oracle). It was a very popular language as a pure object-oriented language (as compared to C++) and was quickly adopted by the Android platform. Java compiles to "bytecode" that is interpreted at runtime by the underlying Java Virtual Machine (JVM) that's running on the OS. You write the mobile apps in Java and program against the Android SDK. The critics of Java say that Java needs a lot of "boilerplate" code to do a simple task, and the concepts like exceptions are difficult to understand. Thus far, this is the most widely used language for Android app development.

2.3.1.1 Advantages of JAVA

- Straightforward Java: It is anything but difficult to program, compose, gather, investigate, and learn than elective programming dialects. Java might be a more modest sum convoluted than C++; therefore, Java utilizes programmed memory portion and trash assortment.
- Item Oriented: It grants you to make standard projects and reusable code.
- Stage Independent Java code:It runs on any machine that needn't bother with any unique programming to be introduced, however, the JVM should be available on the machine.
- Java is a disseminated language: Java is a dispersed language as it gives an instrument for dividing information and projects between numerous PCs that improve the presentation and

proficiency of the framework. The RMI(Remote Method Invocation) is something that bolsters the dispersed handling in Java.

- Secure Java: It has no unequivocal pointer. Besides this, it is a security administrator that characterizes the entrance of classes.
- Memory distribution: In Java, memory is part into two sections one is stored and another is stack. At whatever point we pronounce a variable JVM gives memory from one or the other stack or pile space. It assists with remaining the information and reestablish it without any problem.
- Multithreaded: It is the potential for a program to perform numerous assignments simultaneously. at long last time showed up to become familiar with the ideas of Multithreading in Java.
- Java gives Automatic Garbage Collection: There is programmed memory for the executives in Java that is overseen by the Java Virtual Machine(JVM). At whatever point the articles are not utilized by programs any longer and they don't allude to anything.

2.3.1.2. Disadvantages of Java

- Execution Java language: It is a more slow language when contrasted with different dialects as it is a memory burning-through language.
- Look and Feel: The default look of GUI applications written in Java utilizing the Swing toolbox is very not quite the same as local applications.
- Memory Management: In Java, Memory is overseen through trash pickup, at whatever point the refuse gatherer runs, it influences the exhibition of the apparatus. This is frequently in light of the fact that all different strings inside the require to be halted to allow the junk authority string to figure.
- Java requires huge memory space: Java requires a critical or significant measure of memory space when contrasted with different dialects like C and C++. During the execution of trash assortment, the memory productivity and the exhibition of the framework might be unfavorably influenced.
- Verbose and Complex codes: Java codes are verbose, implying that there are numerous words in them and there are numerous long and complex sentences that are hard to peruse and comprehend. This can decrease the meaningfulness of the code.

2.3.2. KOTLIN:

In 2017 Google announced that it will support Kotlin as an alternative first-class language for Android programming. Kotlin is interoperable with Java, and all Java libraries can be called from Kotlin. Loosely speaking, Kotlin is a tidier form of Java. The learning curve from Java to Kotlin is smooth. At an execution level, Kotlin compiles to Java Bytecode. It was developed by JetBrains and officially supported by Google as a first-class language for Android development. Kotlin offers concise syntax and many modern features that make it more expressive and safer than Java.

2.3.2.1. Advantages of Kotlin

- Smaller learning curve: Compared to its predecessor Java, Kotlin's learning curve is much smaller. Anyone who has a basic understanding of programming can start developing in Kotlin without any previous Android development experience. Instead of just cloning the functionalities of Java, Kotlin focuses on enhancing its features to be much more reliable and easy, based on the most useful parts of the high-level programming languages. This allows Kotlin to deliver the exact same functionality with a much cleaner and easy-to-learn structure.

- Productivity improvement: Because it is easy to learn and maintain, there is a huge productivity boost compared to Java and other Android development programming languages, it may not be as fast as coding in Flutter. However, given the functionality and the well-thought-out features, if you really want to make Android development much more fun and easy, Kotlin is the best choice out there. Kotlin can be extremely productive at the same time being powerful for developing complex applications that require a lot of business logic.
- Fewer bugs: If you code in Kotlin, the chances of making bugs are extremely low compared to other complex programming languages. The reason is very simple and straightforward. Because it is easy to learn and only involves less code to accomplish the same thing, the number of bugs will be also less. So, if you are a Kotlin programmer, the time you spend on fixing bugs will be extremely low compared to other languages. Definitely a catch for experienced developers who frequently spend hours figuring out the source of a bug.
- Better reliability: More and more developers are moving towards programming languages that are easy to write and manage. This means complex programming languages such as Java might go stale in the near future. While it is not a certainty, the latest statistics based on the programming language surveys suggest that it is better to go with the high-level programming languages unless you want to really dig deep into the features provided by the platforms. Of course, if you want to make use of the extremely advanced functions that are only accessible with low-level languages such as Java, you have no other option except to go for it. If that is not the case, going with a high-level programming language such as Kotlin will definitely provide you with better reliability moving forward.
- Incorporate with Java: If you want to start using Kotlin, you do not have to completely rewrite your code base into the language. You can simply integrate it with the existing Java code and start writing the new sections in Kotlin. The reason is very simple, Kotlin is built on top of Java. This means, any code that you write in Kotlin is basically Java code. This can be a huge advantage for companies that already use a lot of Java code.is built on top of Java. This means, any code that you write in Kotlin is basically Java code. This can be a huge advantage for companies that already use a lot of Java code.

2.3.2.2. Disadvantages of Kotlin

- Not as mature as Java: One of the biggest drawbacks of starting with Kotlin is that it is not a mature programming language like Java. This means that there could be a lot of bugs and huge changes coming up with every update. Even though there is a huge community around the language, if you run into some rare bugs, you might get stuck on it without getting an answer for a while. While programming languages such as Java have been around for decades, it is very easy to get an answer for almost any bug that you might come across.
- A bit slower: Kotlin is fast in a lot of areas, such as performing incremental builds. However, when it comes to raw power in developing clean Android applications, Java is still the winner. The reason is obvious, because Kotlin is built on top of Java it is not going to get the exact same performance because of the underlying codebase that has to run on top of the basic Java language. It is not extremely slow, and is completely ignorable considering the ease it provides in

developing Android applications, eradicating the complexity for the beginners. This is a common disadvantage of high-level programming languages.

- Hiring can be difficult: Because Kotlin is developed recently and is not as mature as Java as mentioned earlier, the number of available developers for Kotlin is also not as big. If you are a big company and planning to hire Kotlin developers as soon as possible, especially if the number is high, you might find it difficult to find experienced Kotlin developers. While many job portals such as Indeed have a pretty big list of Kotlin developers, finding good ones might be a bit difficult compared to hiring a Java developer. If you are a small company and just want to hire one or two, it will not be a big problem.
- Learning Kotlin can be hard: we say learning Kotlin can be hard, we are not referring to the programming language itself, but finding proper learning materials. When you have carefully crafted learning materials made by experienced professionals, learning anything can be extremely easy. Because Kotlin is a fairly new programming language, the number of available resources that you can refer to is minimal. That means you will have to spend a lot of time figuring out how to do a certain thing in Kotlin, even when there are Java alternatives available everywhere on the internet.
- You still need Java: Even though you are planning to completely program you are application in Kotlin, you might still want to use Java For certain things. Because the language is not mature enough to provide every single functionality that you might need, especially accessing the advanced functionalities of the hardware, you might still want to make use of Java to make those things work.So technically, you will be incorporating a lot of Java code inside your Kotlin project. While it cannot be considered as a complete disadvantage, because Kotlin might force you to learn Java anyway, there is still a setback. We should not forget that Java also has a lot of advantages and disadvantages.

2.3.3. Comparisons between JAVA and KOTLIN

criteria	**JAVA**	**KOTLIN**
syntax	C-style syntax with semicolons	Modern, expressive syntax
Null Safety	Null values allowed	Null safety with nullable types
Functional Support	Limited functional programming features	Strong functional programming support
Extension Functions	Not supported	Supported
Properties	Getters and Setters	Property syntax with implicit getters/setters
Collections	JAVA collection with API	Kotlin Collections API with additional features
Operator Overloading	Limited operator overloading	Operator overloading supported

2.4. Programming Language for both Android and iOS

2.4.1. HTML5 and Apache Cordova:

Apache Cordova started out as a project called PhoneGap from Adobe. With Apache Cordova you can run HTML and JavaScript (JS) code in a captive browser instance called the Webview. By using a webview, you can write code once and then execute it anywhere. One of the criticisms of the cross-platform approach is that the response can be sluggish owing to the JavaScript code being executed in the webview. The HTML code is packaged along with the app itself and installed on the mobile device. This opens up the possibility for using a wide variety of web UI frameworks like Jquery, React JS, Bootstrap, Angular JS, or Vue. There are also other layered frameworks around it like Ionic that ultimately run on Cordova. Apart from the UI libraries, there is a concept of plugins that allow the JS code to access native device capabilities like the camera, contact list, or location. There is a large ecosystem of third party plugins available. The most popular package manager is npm. The IDEs that are popular among the developer community are Visual Studio Code, and Eclipse. Ionic, which comes as a wrapper on top of Apache Cordova, and currently uses Angular JS, also has its own IDE called Ionic Creator that's available for a purchase. However, you can still develop an Ionic Angular JS app using any other IDE as mentioned earlier.

2.4.2. C# and Xamarin:

C# is an object-oriented programming language that was developed by Microsoft. The Xamarin framework (acquired by Microsoft) allows you to program in C# against the .NET framework. The .NET framework is implemented on the iOS platform using an open source implementation called mono. The popular IDEs for writing C# and Xamarin code is Visual Studio Code by Microsoft. The C# code is cross-compiled and runs natively on the iOS or Android device. This allows for a no-lag execution that comes very close to native development. There are special extensions called Xamarin.iOS and Xamarin.Android that you can use to access iOS and Android native capabilities that can be called from C#. For iOS, you need XCode on a Mac machine to build the installable iOS app. "*Source* (IBM)"

2.4.3. JavaScript and React Native:

React Native was released in 2015 by Facebook. React Native uses JavaScript as a programming language for writing mobile apps. There is no HTML used in writing React Native apps. This code is then interpreted at runtime and executed using a "bridge" for accessing the device's native SDK capabilities. React Native apps use the platform native UI library to render the UI components, which makes the UI truly native. React Native has become very popular because the learning curve for JavaScript is very low.

2.4.4 Comparisons between HTML5/Apache Cordova, C#/Xamarin and JavaScript/React Native

Criteria	HTML5/Apache Cordova	C#/Xamarin	JavaScript/React Native
Language	HTML, CSS, JavaScript	C#	JavaScript
Native Performance	Limited	Excellent	Good
Cross-Platform	Yes	Yes	Yes
Development	Web-based	IDE-based	IDE-based
Performance	Moderate	Excellent	Good
Popular Applications	Apache Cordova, Ionic, PhoneGap	Microsoft Teams, Slack, UPS Mobile	Facebook, Instagram, Airbnb
Access to Device APIs	Through Plugins	Native access	Native access
Development Speed	Fast	Fast	Fast
UI Components	Limited	Extensive	Extensive

Code Reusability	High	High	High

While these are the major programming languages for android , other programming languages can be used such as; dart, python, Lua, ruby, etc.

SECTION 3: REVIEW AND COMPARE MOBILE APP DEVELOPMENT FRAMEWORKS BY COMPARING THEIR KEY FEATURES (LANGUAGE, PERFORMANCE, COST & TIME TO MARKET, UX & UI, COMPLEXITY, COMMUNITY SUPPORT) AND WHERE THEY CAN BE USED.

3.1. Definition of mobile app framework:

A mobile app development framework is a software toolkit that provides a foundational structure for developing mobile applications. The common types of mobile development frameworks include:

3.2. Different App Frameworks:

3.2.1. Native App Frameworks

Apps are designed for specific operating systems (e.g., Android, iOS, Windows) and utilize device features such as RAM, camera, and GPS.

Example: iOS (Swift / Objective-C) and Android (Java / Kotlin)

3.2.2. Cross-Platform App Frameworks

These frameworks allow developers to write code once and deploy it across multiple platforms.

Examples: react native, flutter, Xamarin, ionic, phonegap/cordova

3.2.3. Progressive Web App (PWA) Frameworks

PWAs are web apps that use modern web technologies to offer a native app-like experience across devices and platforms.

Examples: angular, react, vue.js, polymer

3.2.4. Game Development Frameworks

Game dev frameworks simplify game creation by handling low-level details like rendering, physics, and input. This lets developers focus on game logic and design.

Examples: ARKit (iOS), Unity with AR Foundation, and ARCore (Android)

3.2.5. Augmented Reality (AR) and Virtual Reality (VR) Frameworks

AR and VR frameworks simplify development by handling tracking, rendering, and interaction, allowing for immersive user experiences.

Examples:ARKit (iOS), Unity with AR Foundation, and ARCore (Android)

3.3. Comparing features of the different App Frameworks and Stating where they can be used.

3.3.1. Native App Frameworks

Framework	**Language**	**Performance**	**Cost & Time to Market**	**UX & UI**	**Complexity**	**Community Support**	**Usage**
iOS	Swift/Objective-C	High	Swift: Shorter time,	Native and seamless	Moderate	Strong and active	Native iOS app development
			Objective-C: Longer	iOS user experience		community	
Android	Java/Kotlin	High	Kotlin: Shorter time,	Native user experience	Moderate	Strong and active	Native Android app development

			Java: Longer time	adhering to Material		community	

3.3.2. Cross-Platform App Frameworks

Framework	**Language**	**Performance**	**Cost & Time to Market**	**UX & UI**	**Complexity**	**Community Support**	**Usage**
React Native	JavaScript/TypeScript	Moderate to High	Moderate	Native-like experience	Moderate	Strong and active	Cross-platform (iOS, Android)
Flutter	Dart	High	Moderate	Native-like experience	Moderate	Strong and active	Cross-platform (iOS, Android, Web)
Xamarin	C#	High	Moderate to High	Native-like experience	High	Strong and active	Cross-platform (iOS,

Framework	Language	Performance	Cost & Time to Market	UX & UI	Complexity	Community Support	Usage
							Android, Windows)
Ionic	HTML/CSS/JavaScript	Moderate	Low	Web-based UI	Low	Active community	Cross-platform (iOS, Android, Web)
PhoneGap/Cordova	HTML/CSS/JavaScript	Low to Moderate	Low	Web-based UI	Low	Active community	Cross-platform (iOS, Android, Web)

3.3.3. Progressive Web App (PWA) Frameworks

Framework	**Language**	**Performance**	**Cost & Time to Market**	**UX & UI**	**Complexity**	**Community Support**	**Usage**
Angular	TypeScript	High	Moderate	Native-like experience	Moderate to High	Strong and active	Web, Mobile, Desktop
React	JavaScript/TypeScript	High	Moderate	Customizable	Moderate	Strong and active	Web, Mobile, Desktop
Vue.js	JavaScript	Moderate to High	Low to Moderate	Customizable	Low to Moderate	Strong and active	Web, Mobile
Polymer	JavaScript	Moderate	Low to Moderate	Material Design	Low	Moderate community	Web

3.3.4. Game Development Frameworks

Framework	**Language**	**Performance**	**Cost & Time to Market**	**UX & UI**	**Complexity**	**Community Support**	**Usage**
Unity	C#	High	Moderate to High	Customizable	Moderate to High	Strong and active	2D/3D Game Development
Unreal Engine	C++	High	Moderate to High	Customizable	High	Strong and active	3D Game Development
Godot	GDScript/C#	Moderate to High	Low to Moderate	Customizable	Moderate	Strong and active	2D/3D Game Development
Cocos 2d	C++/Lua/JavaScript	Moderate	Low to Moderate	Customizable	Moderate	Strong and active	2D Game Development
Phaser	JavaScript	Moderate	Low to Moderate	Customizable	Low to Moderate	Strong and active	2D Game Development

3.3.5. Augmented Reality (AR) and Virtual Reality (VR) Frameworks

Framework	Language	Performance	Cost & Time to Market	UX & UI	Complexity	Community Support	Usage
ARKit	Swift/Objective-C	High	Low to Moderate	Customizable	Moderate	Strong and active	iOS (iPhone, iPad)
ARCore	Java/Kotlin	High	Low to Moderate	Customizable	Moderate	Strong and active	Android (Smartphones, Tablets)

SECTION 4: STUDY MOBILE APPLICATION ARCHITECTURES AND DESIGN PATTERNS

Mobile application architecture and design patterns play a crucial role in ensuring that mobile apps are scalable, maintainable, and performant.

4.1. Mobile Application Architectures

4.1.1. Client-Server Architecture:

- In mobile apps, the client (mobile device) interacts with the server (backend) to fetch data, perform operations, and receive responses.
- Common protocols used in mobile app communication include HTTP(S), RESTful APIs, and WebSocket for real-time communication.

4.1.2. Components of Mobile App Architecture:

- Presentation Layer/UI:Handles user interaction, interface design, and presentation logic.
- Business Logic Layer:Implements application-specific logic, rules, and algorithms.
- Data Access Layer:Manages data retrieval, storage, and manipulation from various sources (local storage, databases, APIs).

4.1.3. Key Considerations in Mobile App Architecture:

- Scalability: Design the architecture to handle increasing user loads and data volumes efficiently.
- Security: Implement measures like data encryption, secure authentication, and secure communication protocols to protect user data.
- Performance: Optimize app performance through efficient data caching, network optimizations, and UI responsiveness.
- Offline Capabilities: Incorporate offline data storage and synchronization mechanisms for uninterrupted app usage in offline mode.
- Cross-Platform Compatibility: Choose architecture patterns and frameworks that support cross-platform development if targeting multiple platforms.

4.2. Design Patterns in Mobile App Development:

4.2.1. MVC (Model-View-Controller):

- Separates the app into three components:
- Model: Represents data and business logic.
- View: Handles user interface and presentation logic.
- Controller: Mediates between the model and view, handling user input and updating the model/view accordingly.
- Promotes code organization, reusability, and separation of concerns.

4.2.2. MVVM (Model-View-ViewModel):

- Similar to MVC but introduces a ViewModel layer:
- Model: Represents data and business logic.
- View: Handles user interface and presentation logic.
- ViewModel: Acts as an intermediary between the model and view, exposing data and commands to the view.
- Facilitates data binding, testability, and UI updates based on data changes.

4.2.3. MVP (Model-View-Presenter):

- Separates the app into three components:
- Model: Represents data and business logic.
- View: Handles user interface and presentation logic.
- Presenter: Mediates between the model and view, handling user input, updating the model, and updating the view.
- Focuses on improving testability, code maintainability, and UI logic separation

4.2.4. Singleton Pattern:

- Ensures that a class has only one instance and provides a global point of access to that instance.
- Useful for managing shared resources, such as database connections, network managers, or application settings.

4.2.5. Observer Pattern:

- Defines a one-to-many dependency between objects, where changes in one object (the subject) trigger updates to its dependents (observers).
- Commonly used for handling event-driven UI updates, data synchronization, and notifications.

4.2.6. Factory Pattern:

- Encapsulates object creation logic, allowing the calling code to create objects without specifying their concrete classes.
- Improves code maintainability, flexibility, and testability by promoting loose coupling.

By understanding and implementing appropriate mobile application architecture and design patterns, developers can create robust, scalable, and user-friendly mobile apps that meet business requirements and user expectations.

SECTION 5: STUDY HOW TO COLLECT AND ANALYZE USER REQUIREMENTS FOR A MOBILE APPLICATION (REQUIREMENT ENGINEERING)

5.1. How to collect user requirements

User requirements are statements in natural language along with corresponding diagrams (tables, forms, intuitive diagrams) detailing the services provided by the system and operational constraints it must comply with. Additionally, it’s worth noting that user requirements primarily focus on the user’s needs. Thus, these user requirements cater to the customer.

5.1.1. Identify your target audience:

- Determine the demographics, behavior patterns, and characteristics of your target users. Consider factors such as age, gender, occupation, location, and interests.
- Conduct market research and user surveys to gain insights into user preferences, habits, and expectations.
- Create user personas that represent different user archetypes within your target audience. Personas can help you understand user motivations, goals, and pain points.

5.1.2. Conduct user interviews:

- Schedule one-on-one interviews with representative users.
- Prepare a set of open-ended questions to encourage users to express their needs, challenges, and desired features.
- Conduct interviews in a comfortable and neutral environment to foster open and honest conversations.
- Listen actively and take detailed notes during the interviews to capture important insights and quotes.

- Ask follow-up questions to gain a deeper understanding of user requirements and motivations.
- Consider recording the interviews (with user consent) for later reference and to ensure accurate transcription of the conversation.

5.1.3. Organize focus groups:

- Bring together a group of users who represent your target audience.
- Facilitate discussions by presenting specific scenarios and encouraging participants to share their thoughts, ideas, and concerns.
- Assign a moderator to guide the discussion and ensure that all participants have an opportunity to express their opinions.
- Observe the interactions and dynamics within the group, noting any areas of agreement or disagreement.
- Capture the insights, suggestions, and disagreements that arise during the discussion.
- Use techniques such as card sorting or dot voting to prioritize features or ideas collectively.

5.1.4. Use surveys and questionnaires:

- Create online surveys or questionnaires to gather data from a larger user base.
- Keep the questions concise, focused, and easy to understand.
- Include a mix of closed-ended questions with predefined options and open-ended questions for users to provide additional comments or suggestions.
- Consider using Likert scale questions to measure user preferences or satisfaction levels.
- Distribute the surveys through various channels, such as social media, email newsletters, or website pop-ups.
- Consider incentivizing users to increase participation rates, such as offering discounts or entering them into a prize draw.

5.1.5. Analyze existing user feedback:

- Review user feedback from various sources such as app store reviews, customer support inquiries, and social media comments.
- Look for recurring patterns, pain points, feature requests, and areas where users express frustration or satisfaction.
- Categorize the feedback based on themes or topics to identify common user requirements.
- Use sentiment analysis tools to gauge overall user sentiment and identify areas that require improvement.
- Pay attention to both positive and negative feedback to understand user preferences and pain points.

5.1.6. Observe user behavior:

- Conduct usability testing sessions or observe users interacting with similar mobile applications.
- Define specific tasks for users to complete while using the application.
- Document their actions, frustrations, and suggestions for improvement.

- Note how users navigate through the application, perform tasks, and interact with different features.
- Pay attention to areas where users encounter difficulties, confusion, or errors.
- Use screen recording tools to capture the user's interactions and facial expressions during the testing session.

5.1.7. Collaborate with stakeholders:

- Engage with stakeholders such as project managers, designers, developers, and business analysts.
- Gather their insights, perspectives, and requirements.
- Discuss business goals, technical constraints, and any legal or regulatory considerations.
- Identify any specific requirements or constraints imposed by stakeholders.
- Align stakeholder expectations with user needs to ensure a balanced approach.

5.1.8. Create user personas and scenarios:

- Based on the collected information, create detailed user personas that represent different user archetypes within your target audience.
- Include information such as demographic details, goals, motivations, behavior patterns, and pain points.
- Develop user scenarios that illustrate how users would interact with the mobile application in different situations.
- Use personas and scenarios to help stakeholders and development teams empathize with the end users and make informed decisions.

5.1.9. Prioritize and document requirements:

- Review all the collected information, including user interviews, feedback, observations, and stakeholder input.
- Identify common themes, patterns, and priorities in the requirements.
- Prioritize requirements based on user needs, business goals, technical feasibility, and any constraints identified.
- Document the requirements in a clear, concise, and structured manner.
- Include functional requirements (features and capabilities), usability requirements, performance requirements, and any other relevant criteria.
- Use tools such as user story mapping or requirement management systems to organize and track the requirements.

5.1.10. Validate and iterate:

- Validate the requirements with users and stakeholders to ensure accuracy and alignment with their expectations.
- Share the documented requirements with users and stakeholders for review and feedback.
- Conduct usability testing sessions with a prototype orapplication to gather feedback on the proposed features and user flows.
- Seek feedback on the documented requirements to identify any gaps, ambiguities, or conflicting requirements.

- Iterate on the requirements based on the feedback received, making necessary adjustments and refinements.
- Maintain open communication channels with users and stakeholders throughout the development lifecycle to accommodate changes, new insights, and evolving needs.
- Regularly review and update the requirements as the project progresses to ensure they remain relevant and aligned with user and business goals.

5.2. How to analyze user requirements

Requirement Analysis, also known as Requirement Engineering, is the process of defining user expectations for a new software being built or modified. In software engineering, it is sometimes referred to loosely by names such as requirements gathering or requirements capturing. Requirements analysis encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product or project, taking account of the possibly conflicting requirements of the various stakeholders, analyzing, documenting, validating and managing software or system requirements.

Here are the objectives for performing requirement analysis in the early stage of a software project:

- From What to How: Software engineering task bridging the gap between system requirements engineering and software design.
- 3 Orthogonal Views: Provides software designer with a model of:
- system information (static view)
- function (functional view)
- behavior (dynamic view)
- Software Architecture: Model can be translated to data, architectural, and component-level designs.
- Iterative and Incremental Process: Expect to do a little bit of design during analysis and a little bit of analysis during design.

Activities for Requirement Analysis

Requirements analysis is critical to the success or failure of a systems or software project. The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design. Conceptually, requirements analysis includes four types of activity:

1.Eliciting requirements: the task of communicating with customers and users to determine what their requirements are. This is sometimes also called requirements gathering.

2.Analyzing requirements: determining whether the stated requirements are unclear, incomplete, ambiguous, or contradictory, and then resolving these issues.

3.Requirements modeling: Requirements might be documented in various forms, such as natural-language documents, use cases, user stories, or process specifications.

4.Review and retrospective: Team members reflect on what happened in the iteration and identifies actions for improvement going forward.

5.2.1. Gather all requirements documentation:

- Collect all the documentation related to user requirements, including interview transcripts, survey results, focus group notes, user feedback, and stakeholder input.
- Ensure that you have a comprehensive and organized collection of all the information gathered during the requirements gathering phase.

5.2.2. Review and categorize the requirements:

- Read through all the collected information to gain a holistic understanding of the user requirements.
- Identify common themes, patterns, and recurring requirements across different sources.
- Categorize the requirements based on their nature, such as functional, non-functional, usability, performance, security, or regulatory requirements.

5.2.3. Prioritize the requirements:

- Assess the importance and urgency of each requirement to prioritize them effectively.
- Consider the impact of each requirement on the user experience, business goals, and technical feasibility.
- Use techniques such as MoSCoW prioritization (Must have, Should have, Could have, Won't have) or numeric ranking to prioritize the requirements.
- Involve stakeholders in the prioritization process to ensure alignment with business objectives.

5.2.4. Validate and clarify requirements:

- Validate the requirements with users and stakeholders to ensure accuracy and alignment with their expectations.
- Share the documented requirements with users and stakeholders for review and feedback.
- Seek clarification from users and stakeholders on any ambiguous or unclear requirements.
- Address any inconsistencies, contradictions, or conflicts among the requirements.
- Iterate on the requirements based on the feedback received, making necessary adjustments and refinements.

5.2.5. Analyze dependencies and constraints:

- Identify any dependencies or relationships between different requirements.
- Determine if certain requirements are dependent on the implementation of other requirements.
- Consider technical constraints, such as platform limitations or compatibility requirements.
- Take into account any legal, regulatory, or industry-specific constraints that may impact the implementation of certain features or functionalities.

5.2.6. Create a requirements traceability matrix:

- Develop a traceability matrix to link each requirement to its source (e.g., user interview, survey, focus group) and associated stakeholders.

- Use the traceability matrix to track and maintain the relationship between requirements and their origins.
- Ensure that the matrix allows for easy navigation and reference during the development process.

5.2.7. Document the requirements:

- Create a comprehensive requirements document that captures all the analyzed and validated user requirements.
- Structure the document in a clear and organized manner, using headings, subheadings, and bullet points for easy readability.
- Include a description of each requirement, along with any supporting information or rationale.
- Specify any acceptance criteria or measurable metrics for each requirement to facilitate evaluation and testing.

5.2.8. Communicate and seek feedback:

- Share the requirements document with stakeholders, including project managers, designers, developers, and business analysts.
- Conduct meetings or workshops to present and discuss the requirements.
- Encourage stakeholders to provide feedback, raise concerns, or suggest any modifications or additions to the requirements.
- Maintain open and transparent communication channels throughout the development process to accommodate changes, new insights, and evolving needs.

5.2.9. Continuously refine and update the requirements:

- Recognize that requirements analysis is an iterative process.
- Regularly review and update the requirements as the project progresses.
- Incorporate new insights or changes in user needs or business priorities into the requirements document.
- Keep track of any changes made to the requirements for future reference and version control.

SECTION 6: STUDY HOW TO ESTIMATE MOBILE APP DEVELOPMENT COST

It’s not a secret that for most customers, the key question associated with mobile app development lies in its cost since we’re all interested in getting a high-quality product at a reasonable price.

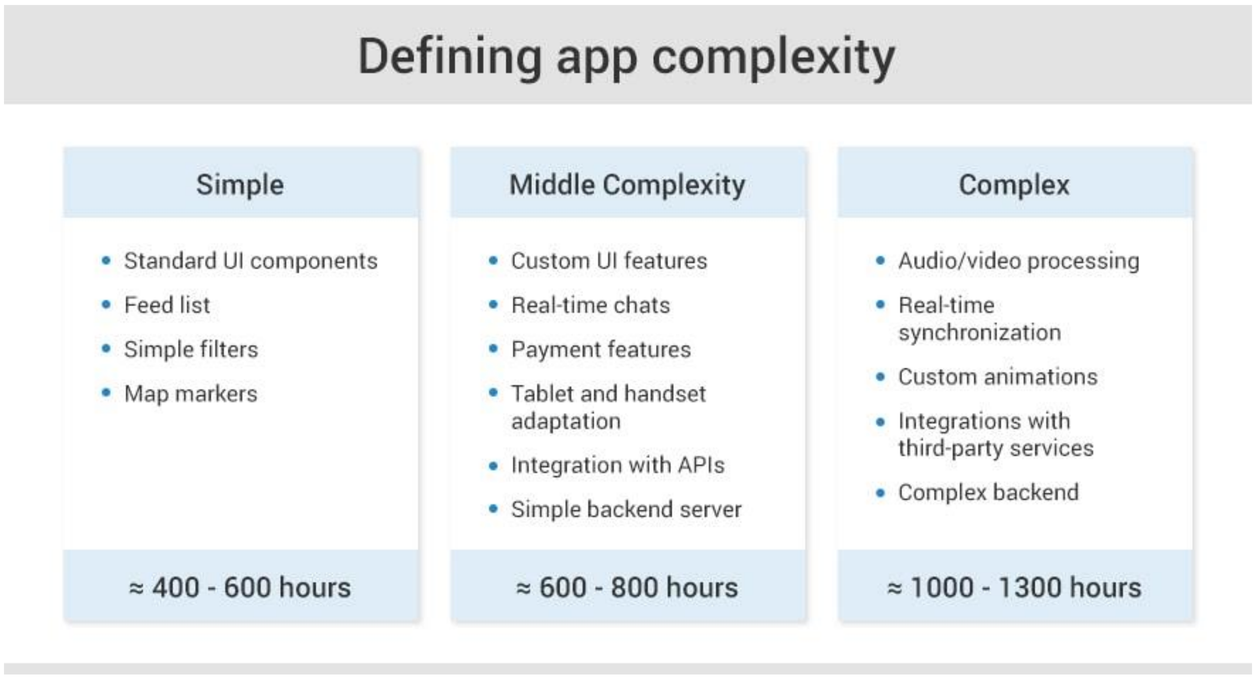
This way, knowing the costs of making a mobile app is vital for both businesses and individuals, as expenses can differ based on various factors affecting the development process.

6.1. Factors Influencing the Cost

The main factors that need to be taken into account when creating a mobile app include:

6.1.1. Product Type
The cost of app development depends heavily on its type, with native and hybrid options available. Native apps are platform-specific (iOS, Android, Windows), while hybrid apps work across multiple platforms. Development costs are influenced by the number of platforms and devices supported, typically cheaper for iOS due to fewer device variations compared to Android. Client preferences matter, but market analysis of device/OS usage helps determine cost-effective strategies. Supporting more Android devices increases costs due to adaptation and testing for different specifications.

6.1.2. Product Complexity
The technical complexity of an app impacts development costs and time. Apps can be categorized as simple, moderately complex, or complex. Costs for a simple app range from $20,000 to $30,000, middle complexity from $30,000 to $40,000, and complex apps start at $50,000, based on an average hourly rate of $50. Development time starts at 400 hours but varies based on complexity, talent involved, platform, and custom features. Android app development typically takes 10-20% more time than iOS, leading to higher costs.



Defining app complexity

Simple	Middle Complexity	Complex
• Standard UI components • Feed list • Simple filters • Map markers	• Custom UI features • Real-time chats • Payment features • Tablet and handset adaptation • Integration with APIs • Simple backend server	• Audio/video processing • Real-time synchronization • Custom animations • Integrations with third-party services • Complex backend
≈ 400 - 600 hours	≈ 600 - 800 hours	≈ 1000 - 1300 hours

6.1.3. After-Release Support
One of the most important things to remember while planning a budget for an app is the after-release support. Usually, it is not included in the initial estimate but makes a huge impact on the product's success.

After-release support may include the following:

- Updates to the app
- New features creation
- Internalization

- Extend to the next platform
- Back-end server maintenance
- Cloud hosting
- Legal support
- Marketing and promotion

6.1.4. Hourly Rates

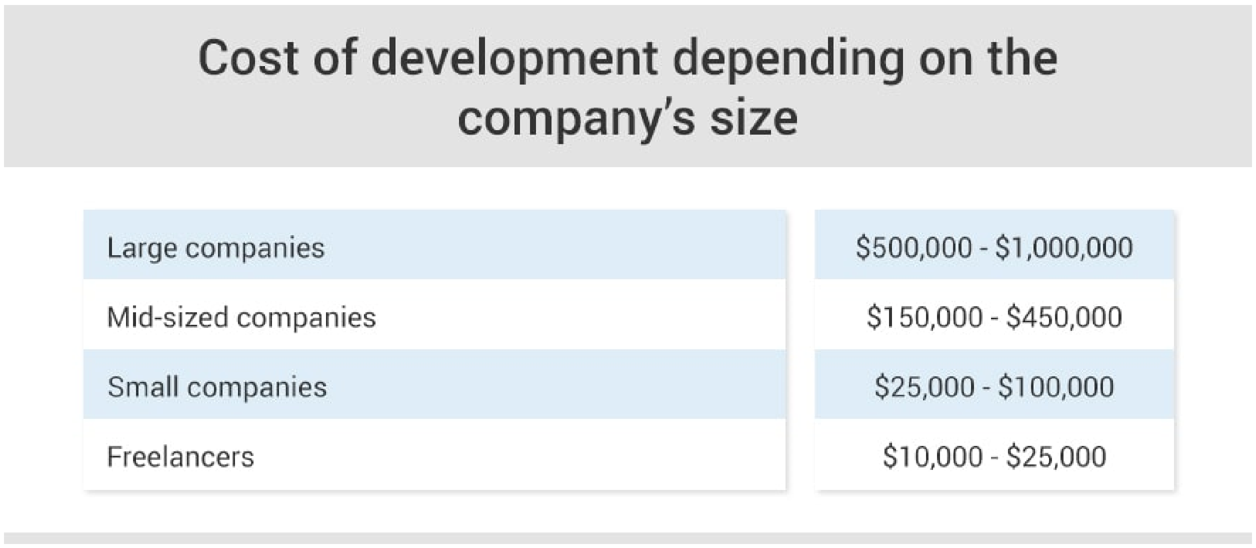
Considering such differences in hourly rates, it is no wonder many US and Canadian customers are opting for outsourcing mobile app development.

Nevertheless, there are also those who wrongly believe that the higher the price — the better the quality. In fact, middle and lower-priced mobile app developers are often known for providing excellent results. So, let's move on and look at the mobile app development companies.

6.1.5. Choosing a Vendor

Meanwhile, the size of the vendor company might depend on the price of the development.

As a rule, the larger the company is — the higher the price.



Cost of development depending on the company's size

Large companies	$500,000 - $1,000,000
Mid-sized companies	$150,000 - $450,000
Small companies	$25,000 - $100,000
Freelancers	$10,000 - $25,000

6.1.6. App Development Team

The size of the development team also affects the price of the app. The larger the project and the higher the requirements — the more programmers should be involved in the development process and, as a result, the higher the overall price. Conditionally, we can subdivide teams into basic and extended.

A basic team is generally made up of the following professionals:

- Project Manager
- Android/iOS app developers (two or more)
- QA Engineer
- An extended team generally includes all the players of a basic team plus:
- Business Analyst

- UI/UX Designer
- Admin panel designer
- Front-end developer
- Back-end developer (if needed)
- DevOps
- Both iOS and Android developers

6.1.7. Types of Apps

Mobile apps are as diverse as the people who use them. From simple tools that provide information to complex platforms for social connections or gaming, mobile apps come in various types, each serving specific needs.

6.1.7.1. Informational Apps

Informational apps are designed to provide users with valuable content or resources. They can include news apps, educational tools, or reference guides.

Typically, these apps have lower development costs due to their austere functionality — their primary goal is simply to present information in an appealing manner.

Examples: Wikipedia, news apps, language learning apps.

6.1.7.2. E-commerce Apps

E-commerce apps allow users to make online transactions, browse products, make purchases, and manage their accounts. These apps often have features like secure payment gateways and order tracking.

Development costs for e-commerce apps can vary widely based on the complexity of features, integration with payment systems, and security measures.

Examples: Amazon, eBay, Etsy.

6.1.7.3. Social Networking Apps

Social networking apps connect people, help them share content, and build online communities. These apps often include features like profiles, messaging, and multimedia sharing.

Developing social networking apps can be complex, with costs impacted by user interface features and scalability plans.

Examples: Facebook, Instagram, LinkedIn.

6.1.7.4. Gaming Apps

Gaming apps provide interactive and entertaining experiences, ranging from casual games to sophisticated, graphics-intensive projects.

The development cost of gaming apps is often higher due to the need for advanced graphics, complex gameplay mechanics, and potential multiplayer features.

Examples: Candy Crush, PUBG, Among Us.

6.1.8. App Maintenance and Ongoing Costs
Once an app is live in app stores, users expect it to function flawlessly without bugs and glitches.

To keep a mobile app up and running after launch, you need to ensure its ongoing maintenance and support.

The ongoing maintenance costs involve continuous monitoring, updates, and enhancements needed to meet user expectations, address security concerns, and stay compatible with the latest operating systems.

To allocate your budget accurately, estimate yearly maintenance expenses considering factors like app complexity, update frequency, user base size, and technological changes.

The ongoing involvement of the app development team, whether in-house or external, is also a crucial factor in efficient maintenance since you utilize their knowledge for quick bug fixing, updates, and improvements.

6.2. App Cost Breakdown
Creating a mobile app involves various stages, each contributing to the overall cost. Understanding the list of these costs is essential for correct budgeting and project management.

User Interface/User Experience (UI/UX) Design

The design phase is crucial for creating an engaging and convenient app. Usually, it includes:

- Wireframing and Prototyping: Planning the app's structure and user flow.
- Graphic Design: Creating visually appealing elements.
- UI/UX Testing: Providing a good user experience.
- Investing in a well-designed app is necessary to attract and retain users, but it can significantly add to the overall cost.
- Development
- Development is where the app comes to life. Key components of development costs include:
- Front-end Development: Creating the user interface and user interactions.
- Back-end Development: Building server-side functionality and databases.
- Integration of Features: Executing various functionalities and services.

The app's complexity, the quantity of features, and the technology used all impact the development costs.

6.3. Quality Assurance (QA) and Testing
Making the app function correctly and free it from bugs is crucial for its success. Testing normally includes:

- Functional Testing: Confirming each feature works as intended.
- Compatibility Testing: Demonstrating the app works on various devices and operating systems.
- Security Testing: Identifying and addressing potential issues.

Investing in thorough testing reduces the risk of post-launch issues but adds to the overall development cost.

6.4. Ongoing Maintenance

Post-launch activities are essential for the app's longevity. Ongoing maintenance includes:

- Regular Updates: Keeping the app compatible with the corresponding devices and the latest OS versions.
- Bug Fixes: Addressing issues reported by users.
- Enhancements: Implementing new features or adjusting existing ones based on user preferences.

While many focus on the initial development cost, it's important to allocate resources for post-launch activities. Resources for ongoing maintenance, updates, and improvements ensure the app remains viable and competitive.

6.5. Current App Development Team

The cost of the development team, including salaries, benefits, and overheads, contributes to the overall project cost. Factors affecting this cost include:

- Geographical Location: Development in regions with higher living expenses generally leads to higher hourly rates.
- Experience and Skill Level: Highly skilled and experienced engineers may request higher compensation.

6.6. Pricing Models Used by App Development Agencies

Choosing the right pricing model can greatly impact your mobile app development project. App development agencies offer various pricing structures, each with advantages and considerations.

6.6.1 Time and Material

Per time & material approach, clients pay based on the number of hours worked by the development team on their project.

Advantages: The hourly structure is well-suited for projects with growing demands or situations when clients want to monitor the development's progress.

Considerations: Hourly rates may result in higher costs if the project goes beyond the initial discussions. Additionally, budgeting for such a payment model can be challenging if the project scope is not well-defined.

6.6.2. Project-Based

A project-based model implies paying a fixed price rate agreed upon for the entire project, regardless of the hours spent.

Advantages: Clients know the total cost upfront.

Considerations: In case of unexpected changes in project scope, a client may get extra charges.

6.6.3. Outstaffing

Per the outstaffing model, clients hire mobile app developers and designers to work exclusively on their projects.

Advantages: Upon the outstaffing structure, a client gets a team that focuses solely on the project while paying less than hiring an in-house team.

Considerations: Although outstaffing is one of the best ways to develop an app, it requires effective communication and project management from the team, especially when team members are in different time zones.

6.6.4. Dedicated Team

A dedicated team resembles the outstaffing model, but the client has more control over team composition and project management.

Advantages: Clients have more influence over the development process.

Considerations: Clients need to handle project management and maintain team coordination and communication.

6.7. Saving on App Development Costs

Developing an app can be a nightmare for your budget. Luckily, there are some strategies that proved to be helpful in reducing an average app development cost:

- **MVP (Minimum Viable Product):** MVP development implies creating only the essential features that make your app functional and valuable to users. After the app release, you can collect feedback and make adjustments based on user needs.
- **Prioritizing Features:** By focusing on essential features, you can manage development costs more wisely. Prioritize features for the initial release and plan subsequent updates for extra functionalities.
- **Thorough Planning:** Provide clear and comprehensive documentation of your app requirements since well-documented plans minimize miscommunication and prevent costly changes.
- **Good Communication:** Maintain open communication with the development team and provide timely feedback to avoid rework and keep the project on track.

Saving on app development costs is about making strategic decisions at every stage of the process. By adopting a consistent and holistic approach, focusing on essential features, and considering all possible nuances, you can create a high-quality app.

6.8. Conclusion

Building a successful app requires a balanced budget; low costs don't guarantee quality, nor does excessive spending ensure success. Understanding cost factors aids in selecting app type, functionality, and development company. While our cost approach offers estimates, actual prices vary between companies. Consider factors influencing costs to make informed decisions and choose a suitable development company.

SECTION 7: REFERENCES

- https://en.wikipedia.org/wiki/Mobile_app
- https://www.deliberatedirections.com/swift-programming-language-advantages
- https://www.geeksforgeeks.org/what-is-objective-c/
- https://developer.ibm.com/articles/choosing-the-best-programming-language-for-mobile-app-development/
- https://developer.ibm.com/articles/choosing-the-best-programming-language-for-mobile-app-development/
- https://aws.amazon.com/compare/the-difference-between-web-apps-native-apps-and-hybrid-apps/
- https://guidingcode.com/user-and-system-requirements-in-software-engineering/
- https://www.linkedin.com/advice/3/how-can-you-overcome-challenges-gathering-user-requirements-vymaf
- https://enlabsoftware.com/development/software-engineering-analyze-requirements.html
- https://twproject.com/blog/project-requirements-collect-analyze/
- https://www.visual-paradigm.com/guide/requirements-gathering/requirement-analysis-techniques/
- https://themindstudios.com/blog/mobile-app-requirements-document/#waystodevelopandmanagerequirements
- https://guidingcode.com/user-and-system-requirements-in-software-engineering/
- https://www.geeksforgeeks.org/software-engineering-requirements-engineering-process/
- Scand.com