

**FACULTY OF ENGINEERING**

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**DEPARTMENT OF COMPUTER ENGINEERING, SE**

**CEF440: Internet and Mobile Programming**



**Presented by:**

| BOBGALA HARRISON DIMA | FE21A151 |
| --- | --- |
| KAMCHE YANN ARNAUD | **FE21A208** |
| MESSI II INNOCENT ROSTAND | **FE21A236** |
| NYANDO ONONGWENE | **FE21A290** |
| NKWI CYRIL AKINIMBOM | **FE21A281** |





**Table of Content**

[**1. Abstract 2**](#_1fob9te)

[**2. Major Types of Mobile Applications: Review and Comparison 2**](#_3znysh7)

[2.1. Native Mobile Apps 2](#_i9dvnrjxx650)

[2.2. PWAs 3](#_h3b4997u7j41)

[2.3. Hybrid Apps 3](#_x5f33q1p5as)

[2.4. Comparison 4](#_r68hlgodcghe)

[2.5. Real World Implementations 4](#_chpmklawn8bl)

[**3. Mobile Application Programming Languages: Review and Comparison 4**](#_2et92p0)

[3.1. Java 4](#_u80btfmsmixm)

[3.2. Kotlin 6](#_incw5nzch5d6)

[3.3. Swift 8](#_eqwnsjhio5a5)

[**4. Mobile Application Development Frameworks: Review and Comparison 10**](#_tyjcwt)

[4.1. Comparison 10](#_6wqvqzttde1j)

[4.2. Performance Benchmarks and Usage Scenarios 11](#_2kzlnp3g981i)

[4.3. Choosing the Right Framework: 12](#_xfd4gbjwm7ox)

[**5. Mobile Application Architectures and Design Patterns 12**](#_3dy6vkm)

[5.1. Types of mobile app architecture patterns 12](#_8h7oqjdvnlt1)

[5.2. Types of mobile app design patterns 15](#_y7xjk2aqb79)

[**6. Collection and Analysis of User Requirements for a Mobile Application 16**](#_1t3h5sf)

[6.1. Introduction 16](#_mt15w65tcwqp)

[6.2. Methods Used To Collect User Requirements for an application 16](#_lpe6qdgymucf)

[6.3. Analysing collected user data for an application 17](#_276oouvzm0dt)

[6.4. Tools and Platforms for Data Collection and Analysis 17](#_b2x9lccxcbkl)

[**7. Estimation of Mobile App Development Cost 18**](#_4d34og8)

# 

# **Abstract**

The mobile app development process is a dynamic journey comprising planning, design, development, deployment, and maintenance stages. This abstract provides a concise overview of this process. It emphasises the initial planning phase, where ideas are refined, followed by the design phase focusing on user experience (UX) and interface (UI) design. The development phase encompasses coding, testing, and optimization, ensuring functionality and performance. Additionally, considerations such as platform compatibility and security are crucial. The deployment and maintenance phases involve app store submission, user feedback integration, and continuous updates to ensure relevance and longevity. Adherence to this process enables developers to navigate complexities effectively and deliver impactful mobile solutions.

# **Major Types of Mobile Applications: Review and Comparison**

An application is a software that lets you exchange information with customers and help them complete specific tasks. Different types of applications, or apps, are based on their development method and internal functionality. Web apps are delivered over an internet browser. Users don't need to install them on their devices. Native apps, on the other hand, are built for a specific platform or device type. The user must install the appropriate software version on their device of choice. Hybrid apps are native applications with a web browser embedded inside them.

## **Native Mobile Apps**

A native app is one that has been developed in the specific programming language of an operating system. In other words, if we are talking about Android, the native app will have been developed with the Kotlin or Java programming language.

And if we are talking about iOS, we will find native apps that have been developed in Swift or Objective C.

The most widely used programming languages in recent years are, in the first case, Kotlin, and in the second, Swift.

This means that if you want to have an app published on Google Play and App Store, you will have to program it twice, once for each of the languages mentioned above.

To give you an idea, **Netflix or Spotify** are examples of native apps.

**Advantages of native applications:**

* **Speed:**  As they are developed in the language of the operating system they work with, native apps are faster
* **Better user experience:** This favours a more positive user experience in the app. This will translate into ratings and comments that will benefit the reputation of the app and attract more downloads.
* **Higher performance:** One of the great advantages that other types of apps do not have is to take full advantage of the functionalities of the smartphones on which they are installed, as they have access to all their sensors (provided that the user gives permission, of course). For example, they can have access to geolocation, camera, microphone, push notifications, fingerprint, internal storage of the device, accelerometer and gyroscope.
* **Widgets:** A native app can benefit from widgets. That is, it can incorporate a thumbnail view of the app on the smartphone's homepage.

**Disadvantages of native applications:**

* **High cost**: As the app has to be programmed twice, once for each operating system, the cost is higher than other types of apps. But, depending on your business model and objectives, it could still be your best option.
* **Greater complexity:** Native apps are more complex to develop and you need an expert team to develop them.
* **Development time:** Being more complex, it will require more development time than other types of apps.

## **PWAs**

These apps aim for app-like UX but with web strengths. They are cheaper and faster to develop than native apps. PWAs are recommended for eCommerce projects.

**PWAs are websites rather than Apps. They work through a browser.** Therefore, it is not necessary to install any files to use them. They are quite versatile and are guaranteed to work on mobile devices of any type, size and operating system.

To give you an idea, **Pinterest or Trivago** are examples of PWAs.

They are called "**progressive**" because they gradually absorb functionalities from apps such as:

* Camera permit.
* Location permit.
* Allow push notifications.
* Auto-completion of forms.
* Offline storage, i.e. without an internet connection. But it is minimal compared to the native ones, as the progressive ones use the browser's storage space.

**Advantages of Progressive Web Apps**

* **They weigh little :** They weigh practically nothing (only what we store offline), as there is no need to download or install a file. They are accessed from a web browser and used through it.
* **There is no need to download them :** As we said, they are opened through the browser. Unlike native or hybrid apps, you will not be able to access them through Google Play or the App Store, but through Google Chrome, Safari, or the browser you use.
* **They are updated automatically :** Normally, the apps we download have to be updated manually or we have to activate the update in the background. With PWAs, however, we will always have access to the most up-to-date version, which will be the one available on the web.
* **They are cheaper :** The reason is that you will only have to develop it in web code. In the case of native and hybrid (these only partially), you will have to develop them twice: once for Android and once for iOS.

**Disadvantages of PWAs**

* **Push notifications do not arrive on iOS** : This is because Safari does not receive notifications on mobile. And this is a disadvantage mainly for companies that want to notify their customers or potential customers of news, promotions, offers, etc. via the app.
* **They cannot be hosted on Google Play or Apple's App Store:** Which is where most people go to look for an app. These markets generate a lot of traffic and opting for a PWA could result in missing out on many important opportunities.
* **The UX is more limited than in native apps:** That is to say, you will not be able to create designs or develop special functionalities. For example, animations, augmented reality... All this would only be possible from a native application.
* **A Progressive Web App cannot access all the functionalities of the operating system :** We are talking about functionalities such as fingerprint access, accelerometer, gyroscope or push notifications in the case of iOS.
* **They do not work offline:** Only minimally, with many restrictions. This can be a complication that affects the user experience for users who are used to using most apps both with and without internet access.

## **Hybrid Apps**

These apps are a compromise between native and web apps, and are often the most cost-effective option. Hybrid apps are native applications with a web browser embedded inside them. They can work cross-platform but may have limitations in leveraging native features. When we talk about hybrid applications, we are referring to a type of mobile applications that consist of a **web code base technology** to which is added **native code.**

To give you an idea, **Twitter or Instagram** are examples of hybrid apps.

### **Advantages of Hybrid Apps**

* Lower development costs
* Easier maintenance
* Integration with Web-based services
* Have an embedded browser

### **Disadvantages of Hybrid Apps**

* Complex apps with many features will run slower
* Rely on system browser security
* Having just one codebase means the app runs equally everywhere(could not perform specific iOS or Android features)

# **Mobile Application Programming Languages: Review and Comparison**

Various programming languages dominate the dynamic world of mobile application development, each tailored to specific platforms or offering cross-platform capabilities. From **Java** and **Kotlin** for Android development to **Swift** and **Objective-C** for iOS, developers have an array of options to choose from based on project requirements and platform preferences.

This section discusses and compares the different mobile app programming languages mentioned above in terms of syntax, performance characteristics, optimisations available and benchmarks showing strengths and weaknesses.

## **Java**

Java is known for its versatility, reliability, and extensive ecosystem, Java enables developers to create robust and feature-rich native Android applications. With its object-oriented approach, Java facilitates modular and reusable code, simplifying the development process and enhancing code maintainability. Java's widespread adoption, coupled with its extensive documentation and community support, makes it a preferred choice for crafting high-quality mobile experiences for millions of Android users worldwide.

* + 1. **Performance Characteristics**

Java performance refers to the speed and efficiency of a Java application or system, and is typically measured by the amount of time it takes to complete a particular task or the amount of resources it uses to do so.

In Java, performance can be affected by a variety of factors, including the code itself, the JVM implementation, and the hardware it runs on.

* **Memory Leaks and Out of Memory Errors:** A memory leak occurs when a program continues to allocate memory without releasing it, causing memory usage to increase over time. This can lead to an *OutOfMemoryError.*
* **Thread Deadlocks:** A thread deadlock is a situation when two or more threads are blocked indefinitely, waiting for each other to release the resources they need to proceed.
* **Code-Level Issues:** Code-level issues in Java are problems that are caused by inefficient or poorly optimised code. These issues can result in poor application performance, higher resource consumption, and poor user experience.

Some basic parameters that can be tuned to improve Java performance include:

* **Footprint:** The amount of memory and CPU resources consumed by a Java application. Tuning the footprint involves minimising the amount of memory and CPU resources used by the application.
* **Throughput:** The amount of work that can be performed by a Java application within a given period of time. Tuning throughput involves optimising the application’s algorithms, data structures, and I/O operations.
* **Latency:** The time it takes for a Java application to respond to user requests. Tuning latency involves minimising the response time of the application.

However, these parameters are interrelated, and tuning one parameter may affect the others. For example, combining low memory usage with high throughput increases latency.

* + 1. **Performance Optimisations**

With the right techniques and a thorough understanding of the language, we can improve the efficiency and responsiveness of Java applications.

* **Choose the Right Data Structures.** Selecting the appropriate data structure for your use case is vital for achieving optimal performance.
* **Optimise Memory Usage.** Efficient memory management can significantly improve the performance of Java applications.
* **Use StringBuilder for String Concatenation.** In Java, strings are immutable. When we concatenate strings using the ‘+’ operator, a new String object is created, which can impact performance. Instead, use StringBuilder to concatenate strings more efficiently.
* **Use Caching and Memoization.** Caching and memoization can help optimise the performance of a Java application by storing and reusing the results of expensive calculations.
  + 1. **Advantages of Java**
* Straightforward. It is easier to program, compose, gather, investigate, and learn than elective programming dialects.
* Item oriented. It grants you to make standard projects and reusable code.
* Memory distribution
* Multithreaded. It has the potential to perform numerous assignments simultaneously.
* Automatic garbage collection. There is programmed memory for the executives in Java that is overseen by the Java Virtual Machine (JVM).
  + 1. **Disadvantages of Java**
* It is a slower language when contrasted with different dialects as it is a memory burning-through language.
* The default look of GUI applications written in Java utilising the Swing toolbox is very not quite the same as local applications.
* Java requires a critical or significant measure of memory space when contrasted with different dialects like C and C++.
* 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.

## **Kotlin**

Kotlin is a modern, trending programming language that was released in 2016 by JetBrains. It has become very popular since it is compatible with Java (one of the most popular programming languages out there), which means that Java code (and libraries) can be used in Kotlin programs. Kotlin is used for:

* Mobile applications (especially Android apps)
* Web development
* Server-side applications
* Data science
  + 1. **Performance Characteristics**

Performance in the context of Kotlin applications involves various aspects ranging from how quickly the code executes to how effectively it uses resources such as memory and processor power.

* **Null-safety.** It prevents null pointer exceptions, which is advantageous for reliability, but if overused without care, it can add unnecessary checks that slow down performance.
* **Use of inline functions and lambdas.** Inline functions are a powerful tool that can reduce overhead by inlining the bytecode of a function at the call site. On the other hand, Lambda expressions can sometimes lead to the creation of anonymous classes and associated overhead if not handled properly.
* Kotlin compiles down to bytecode. Kotlin benefits from the JVM's performance optimizations such as just-in-time (JIT) compilation, garbage collection, and an extensive array of debugging and monitoring tools.
  + 1. **Performance Optimisations**
* **Use “val” Instead of “var” Wherever Possible.** Immutable variables are a good practice in many programming paradigms due to their thread-safety and clear contract. In Kotlin, prefer to use val over var unless you have a compelling reason to allow a variable to change its value.
* **Leverage the Power of Inline Functions.** Kotlin's inline functions expand at call sites, leading to reduced overhead from function calls, particularly when using higher-order functions or lambdas.
* **Smart Casting over Manual Casting.** Smart casting can improve readability and reduce errors resulting from incorrect casts, contributing to application reliability.
* **Avoid Unnecessary Object Creation.** Creating new objects can be expensive, particularly if done repeatedly within loops or in frequently called functions.
  + 1. **Advantages of Kotlin**
* **Smaller learning curve.** Compared to its predecessor Java, Kotlin’s learning curve is much smaller.
* **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.
* **Fewer bugs.** If you code in Kotlin, the chances of making bugs are extremely low compared to other complex programming languages.
* **Incorporate with Java.** If you want to start using Kotlin, you can simply integrate it with the existing Java code and start writing the new sections in Kotlin.
  + 1. **Disadvantages of Kotlin**
* **Not as mature as Java.** This means that there could be a lot of bugs and huge changes coming up with every update.
* **A bit slower.** When it comes to raw power in developing clean Android applications, Java is still the winner.
* **Hiring can be difficult.** The number of available developers for Kotlin is also not as big.

## **Swift**

Swift is a Programming Language used for developing iOS-based platforms like MacOS, Apple mobiles, Apple iOS Watches, and other iOS Software-based Peripherals. This language is almost 80% similar to C and Python languages. Swift programming language is easy to use and also a powerful programming language.

Swift programming language is highly known for being the best security programming language in the world. The syntax of Swift is also very easy to learn.

* + 1. **Performance Characteristics**
* **Quick start-up time:** Swift-based applications quickly start since there are almost no warm-up operations, making Swift an ideal fit for cloud services.
* **Expressive and safe:** Swift enforces type-safety, optionals, and memory safety features that help prevent common programming errors and improve code reliability.
* **Supported ecosystem:** The Swift ecosystem contains many useful libraries and tools specifically designed for server-side development.
  + 1. **Performance Optimisation**
* **Set Optimization Level in Build Settings.** The first thing you need to do to optimise your code is to let Xcode optimise it by itself.
* **Use “Final” and “Private” for Methods and Classes.** Swift is an Object-Oriented language, meaning you can subclass, and override methods to extend functionality.
* **“Inline” Your Code**. It’s always best practice to create small functions, with each function dedicated to a single task.
  + 1. **Advantages of Swift**
* Easy to learn and understand. Not a very complex program to code, it is easy to learn on the go.
* Powerful programming language for building better applications.
* Help in designing application UI with library functions.
* Long term development support and great community.
* Trust and Security
  + 1. **Disadvantages of Swift**
* Not really popular
* Poor interoperability with third-party tools and IDEs
* Lack of support for previous iOS versions

# **Mobile Application Development Frameworks: Review and Comparison**

This reviews and compares popular mobile app development frameworks based on key features:

* **Language**
* **Performance**
* **Cost & Time to Market**
* **UX & UI**
* **Complexity**
* **Community Support**

## **Comparison**

We will also explore where each framework is best suited for app development.

| **Framework** | **Performance** | **Cost & Time**  **To Market** | **UX & UI** | **Complexity/**  **Language** | **Community Support** |
| --- | --- | --- | --- | --- | --- |
| Swift(Native - iOS) | Excellent(Optimised for Apple hardware) | Higher (Potentially) | Full access to native UI components & design patterns | Moderate(Learn Swift) | Large & active community, extensive Apple documentation |
| Kotlin(Native - android) | Excellent(Optimised for Android) | Higher (Potentially) | Full access to native UI components & design patterns | Moderate(Learn Kotlin) | Large & active community, extensive Apple documentation |
| React Native (Hybrid) | Good(may have limitations) | Moderate(faster with single codebase) | Near-native look & feel with good libraries | Moderate(learn Javascript and React) | Large & active community, extensive |
| Flutter (Hybrid) | Very Good(custom rendering engine) | Moderate(faster with single codebase) | Excellent UI capabilities with rich widgets | Moderate(Learn Dart) | Growing community, good resources |
| Xamarin( Hybrid) | Good(may have limitations) | Moderate( faster with single codebase) | Good UI with integration of native controls | Moderate (learn C#) | Large .NET developer community |
| Apache Cordava( web-based) | Lower(limited native features) | Lower(faster with web skills) | More limited UI capabilities | Lower(familiar web technologies like HTML,CSS, Javascript ) | Moderate community, Adobe documentation |
| Ionic(Web-based) | Lower(limited native features) | Lower(faster with web skills) | More limited UI capabilities | Lower(familiar web technologies, might involve additional frameworks like HTML,CSS, Javascript with frameworks) | Moderate community, Angular community support |

## 

## **Performance Benchmarks and Usage Scenarios**

* **React Native:** Offers good performance for many apps, but might struggle with highly complex or graphics-intensive tasks. Ideal for social media, e-commerce, and basic utility apps.
* **Flutter:** Known for excellent performance due to its own rendering engine. Suited for games, animation-heavy apps, and those requiring high responsiveness.
* **Xamarin:** Generally good performance, especially for apps that leverage the .NET ecosystem. Well-suited for enterprise applications and those requiring native-level integration.
* **Ionic:** Performance can vary depending on the complexity of the web app integrated. Best for simpler web apps or those with existing web backends.

## Choosing the Right Framework:

* **Native development (Swift, Kotlin):** Best for performance-critical apps, full access to native features, and a truly native look and feel. Consider the target platform (iOS or Android).
* **Hybrid (React Native, Flutter, Xamarin):** Good for faster development, sharing code across platforms, and reaching a wider audience. Consider factors like developer skills, performance needs, and supported platforms.
* **Web (Cordova, Ionic):** Suitable for simpler apps, rapid development using web skills, and targeting multiple platforms without a significant performance penalty. Be aware of potential limitations.

Remember, the best framework depends on your project's specific requirements, team expertise, and target platforms.

# **Mobile Application Architectures and Design Patterns**

## **Types of mobile app architecture patterns**

When defining app architecture, another approach is through the pattern model, which encapsulates how components interact and organise within the application.

1. **Model-View-Controller (MVC):**

MVC stands out as the simplest architecture pattern, widely adopted in iOS app development. It delineates the application into three main components: Model, View, and Controller.

The *Model* component undertakes data handling responsibilities akin to the data layer in other architectural models. It retrieves data from various sources such as databases or APIs.

The *Controller* component acts as a liaison between the Model and View components, processing data from the former and delivering it to the latter. Core app logic and algorithms reside within the Controller.

The *View* component is responsible for user interface (UI) rendering, presenting data to users.

## 

1. **Model-View-Presenter (MVP):**

MVP architecture, prevalent in Android development, shares similarities with MVC but exhibits distinct characteristics. It comprises Model, View, and Presenter components.

In MVP, the Model component retains data handling responsibilities, while the View component represents the UI. However, the key distinction lies in the Presenter component.

Unlike MVC, where the Controller manages interactions, in MVP, the Presenter orchestrates data processing for the View. Additionally, Views initiate requests to the Presenter and Model, fostering reusability and modularity.

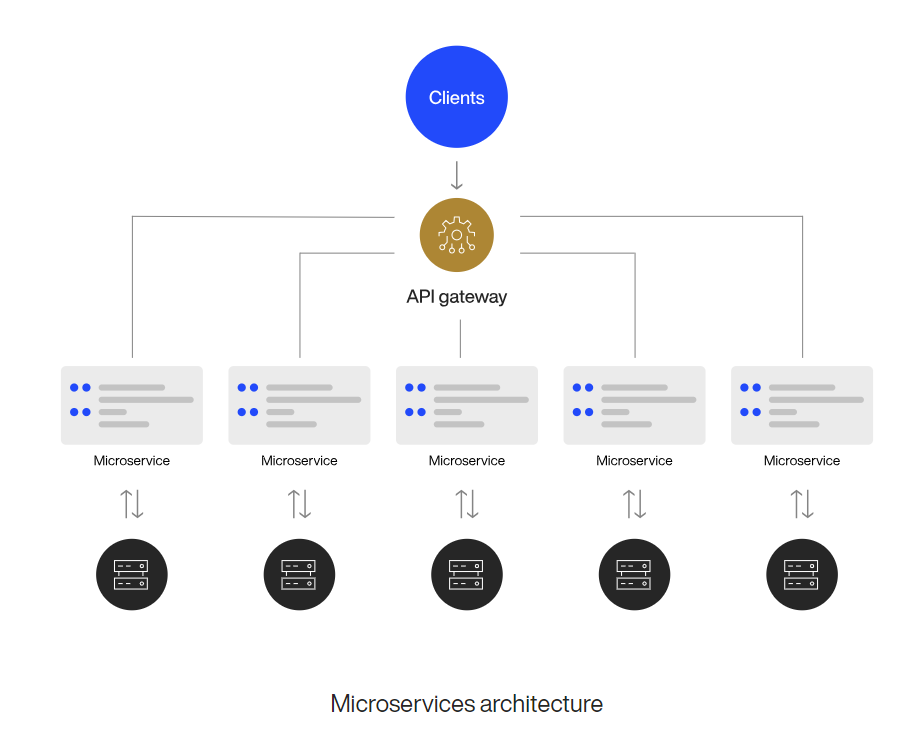
## 

1. **Model-View-ViewModel (MVVM):**

MVVM architecture offers a refined separation of concerns, enhancing maintainability compared to MVC and MVP. This pattern segregates code logic into View, ViewModel, and Model components.

The View component encompasses visual elements and UI, but it relinquishes direct control over UI elements. Instead, data binding serves as the conduit between View and ViewModel.

The ViewModel contains application logic, decoupled from the View, promoting greater separation of concerns. Meanwhile, the Model handles data operations.



1. **VIPER ARCHITECTURE**

VIPER stands for View, Interactor, Presenter, Entity, and Router. VIPER is primarily based on clean architecture ideas, which aim to separate the concerns of different layers of the utility. Each layer has a single duty and communicates with different layers through properly-defined interfaces

* **View**: This layer handles the user interface, displaying data from the presenter and forwarding user actions.
* **Presenter**: Responsible for formatting data and updating the view, the presenter fetches data from the interactor and handles user actions, often navigating to other screens via the router.
* **Interactor:** This layer contains business logic, accessing data from external services, manipulating it, and communicating with the entity layer for data storage.
* **Entity**: Represents data models and systems, ensuring consistent data representation throughout the application. It may also include data access objects (DAOs) or repositories.
* **Router:** Manages navigation between screens, creating and presenting view controllers, handling dependencies, and executing navigation requests received from the presenter.

## 

## **Types of mobile app design patterns**

In mobile development, various design patterns are employed to enhance code organisation, maintainability, and flexibility. Here are some commonly used design patterns:

1. **Singleton Method Design Pattern**: The singleton policy ensures that there is only one instance of a class and provides global access. This is especially useful when you want to manage a single instance of an object or control access to a delayed object.
2. **Factory Method Design Pattern:** Defines an interface for creating objects, allowing subclasses to modify the type of the created object, beneficial for creating objects with different functionalities, like payment gateways in a mobile app.
3. **Observer Method Design Pattern**: Establishes one-to-many dependencies between objects, notifying dependents when the state of an object changes, ideal for distributed event scheduling, such as updating various features in a reports app.
4. **Dependency Injection (DI) Method Design Pattern:** Provides class dependencies externally rather than creating them within the class, enhancing code modularity and testability, as seen in placing a database object externally in an Android app.
5. **Adapter Method Design Pattern:** Allows using the interface of an existing class as a link to a new one, useful for working with third-party libraries without modifying existing code, such as adapting data format from a third-party library to match the app's expectations.
6. **Strategy Method Design Pattern:** Defines a family of algorithms, providing flexibility to select the appropriate algorithm at runtime, beneficial for offering different options for a task, like retrieving weather information through various methods in a weather application.
7. **Composite Method Design Pattern:** Organises objects in a tree structure to represent a part-of-the-whole relationship, helpful for managing individual objects and sets of objects accurately, such as creating complex shapes from simple shapes in a mobile drawing app.

# **Collection and Analysis of User Requirements for a Mobile Application**

## **Introduction**

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, analysing, documenting, validating and managing mobile applications or system requirements.

## **Methods Used To Collect User Requirements for an application**

Here, we focus on exploring the diverse methods used to collect user requirements for an application, highlighting their importance and applicability in the development lifecycle. Some of these methods include:

1. **Surveys and Questionnaires**

User expectations and preferences can be gotten through the distribution of surveys or questionnaires to a wide audience to gather quantitative data on user preferences, needs, and behaviours.

1. **Interviews**

Conduct one-on-one or group interviews with users to gather qualitative insights into their needs, preferences, and challenges.

1. **Observational Analysis**

Here, we observe users in their natural environment as they interact with similar products or perform relevant tasks.

1. **User Analytics**

User requirements can be also collected by utilising analytics tools within existing apps or platforms to gather data on user interactions, behaviours, and usage patterns.

1. **Feedback via prototypes**

A prototype of the mobile application can be developed in order to gather feedback from its users and iterate the various changes in the next version to be launched.

## **Analysing collected user data for an application**

Data regarding user requirements collected using the above methods can be analysed as follows (respectively):

* Analyse survey responses to identify common trends, preferences, and pain points. Use statistical analysis to quantify results and prioritise requirements.
* Transcribe and analyse interview recordings or notes to identify recurring themes, motivations, and specific requirements. Look for deeper insights that may not emerge from quantitative methods.
* Document user behaviours, pain points, and workflow inefficiencies. Analyse observational data to identify opportunities for improvement and inform design decisions.
* Analyse quantitative data to identify patterns, trends, and areas for improvement. Use metrics such as engagement, retention, and conversion rates to prioritise requirements and measure success.
* Analyse user feedback from testing sessions to identify usability issues, feature preferences, and areas for improvement. Iterate on prototypes based on user input to refine requirements.

## **Tools and Platforms for Data Collection and Analysis**

For detailed data and efficient data collection and analysis, the following tools and technologies can be used:

* **Google Forms:** A free tool provided by Google for creating online surveys and forms to collect data.
* **SurveyMonkey:** An online survey tool that allows users to create and distribute surveys to collect data from respondents.
* **Microsoft Forms:** A part of the Microsoft Office 365 suite, it allows users to create surveys, quizzes, and polls to collect data.
* **R programming language:** R is a programming language and software environment for statistical computing and graphics. It offers a wide range of packages and libraries for data analysis, making it suitable for conducting advanced statistical analyses and modelling.

# **Estimation of Mobile App Development Cost**

The final cost of an application depends on several factors:

1. **Features and Functionality:** The complexity and number of features in the app significantly impact development costs. Basic apps with standard features such as user authentication, push notifications, and basic UI elements will cost less than apps with complex features like real-time messaging, geolocation, AR/VR integration, and social media integration.
2. **Platform:** The choice of platform (iOS, Android, or both) affects development costs. Developing for multiple platforms increases costs compared to focusing on a single platform. Additionally, each platform may have its own set of design and development requirements, which can affect costs.
3. **Complexity:** The complexity of the app, including its user interface, backend integration, and business logic, influences development costs. Highly complex apps with intricate workflows, data processing, and integrations with third-party services will require more development time and resources, thus increasing costs.
4. **Design:** The design of the app, including UI/UX design and customization requirements, affects development costs. Custom-designed apps with unique user interfaces and interactive elements will cost more than apps using pre-built templates or standard design patterns.
5. **Team Size and Location:** The size and location of the development team impact costs. Larger teams with more developers, designers, and project managers will incur higher expenses. Additionally, labour costs vary depending on the location of the development team, with teams in regions with higher living standards typically charging more for their services.