**UNIVERSITY OF BUEA**

**Republic Of Cameroon Republic Du Cameroun**

**Ministry Of Higher Education Ministere De L’enseignement Superieur**

**Peace-Work-Fatherland Paix-Travail-Patrie**



**FACULTY OF ENGINEERING AND TECHNOLOGY**

**DEPARTMENT OF COMPUTER ENGINEERING**

**CEF 440 : INTERNET PROGRAMMING (J2EE) AND MOBILE PROGRAMMING**

[GitHub - Asumu22/group-15](https://github.com/Asumu22/group-15)

**TASK 1**

**GROUP 15**

GROUP MEMBERS

|  |  |  |  |
| --- | --- | --- | --- |
| NO | NAMES | MATRICULE | SPECIALTY |
| 01 | KONGNYUY RAYMOND AFONI | FE21A219 | NE |
| 02 | BESSONG ELIAS ASUMU | FE21A149 | SE |
| 03 | KANKO KEMEDJEU DUPLEX | FE21A210 | NE |
| 04 | KENEDY MALLEY ITUKA | FE21A212 | NE |
| 05 | AZEFACK JUNIOR | FE21A146 | NE |

**Instructors :**

**DR. VALERY**

TABLE OF CONTENT

[1. Review and compare the major types of mobile apps and their differences 4](#_Toc162816258)

[Comparison 4](#_Toc162816259)

[Choosing the Right Option: 4](#_Toc162816260)

[2. Reviewing and Comparing Mobile App Programming Languages 6](#_Toc162816261)

[1. Swift 6](#_Toc162816262)

[2. Kotlin 6](#_Toc162816263)

[3. Java 6](#_Toc162816264)

[4. Dart 6](#_Toc162816265)

[5. JavaScript 6](#_Toc162816266)

[6. C# 6](#_Toc162816267)

[7. Objective-C 6](#_Toc162816268)

[8. Python 6](#_Toc162816269)

[Choosing the Right Language: 7](#_Toc162816270)

[Conclusion: 7](#_Toc162816271)

[3. Review and compare mobile app development frameworks by comparing their key features and where they can be used. 8](#_Toc162816272)

[A Comparison of Mobile App Development Frameworks 8](#_Toc162816273)

[1. Swift (Native) 8](#_Toc162816274)

[2. Kotlin Native 8](#_Toc162816275)

[3. Xamarin 8](#_Toc162816276)

[4. Ionic 9](#_Toc162816277)

[5. React Native 9](#_Toc162816278)

[6. Flutter 9](#_Toc162816279)

[Choosing the Ideal Framework: 10](#_Toc162816280)

[4. Study mobile application architectures and design patterns 11](#_Toc162816281)

[Architectural Patterns: 11](#_Toc162816282)

[Other Important Patterns: 14](#_Toc162816283)

[Choosing the right design pattern 15](#_Toc162816284)

[5. Study how to collect and analyse user requirements for a mobile application (Requirement Engineering) 16](#_Toc162816285)

[Step 1: Define Your App Idea and Purpose 16](#_Toc162816286)

[Step 2: Gather and Align the App and Business Objectives or Goals 16](#_Toc162816287)

[Step 3: Run a Market Analysis and Competitor Analysis 16](#_Toc162816288)

[Step 4: Determine a User Persona Scenarios 17](#_Toc162816289)

[Step 5: Gather and Prioritise Functional and Non-Functional Requirements 17](#_Toc162816290)

[Step 6: Design Use Cases and other UML diagram 17](#_Toc162816291)

[Step 7: Write an App Requirements Document 18](#_Toc162816292)

[Step 8: Deploy Prototyping and Wireframing 18](#_Toc162816293)

[Step 10: Apply Agile Methodology 18](#_Toc162816294)

[6. Study how to estimate mobile app development cost 19](#_Toc162816295)

[1. Initial Development Phases - Cost Impact 19](#_Toc162816296)

[2. Ongoing Maintenance Costs 19](#_Toc162816297)

[3. Additional Costs to Consider in long term 19](#_Toc162816298)

[4. Factors Influencing Cost 19](#_Toc162816299)

[6. Conclusion 20](#_Toc162816300)

REFERENCES.......................................................................................................................................................21

# Review and compare the major types of mobile apps and their differences

**What's the Difference Between Progressive Web Apps, Native Apps, and Hybrid Apps?**

An application is a software that lets you exchange information with customers and help them complete specific tasks. Different types of applications, are based on their development method and internal functionality.

**Progressive 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.

## Comparison

|  |  |  |  |
| --- | --- | --- | --- |
| **Characteristics** | **PWAs app** | **Hybrid app** | **Native app** |
| **Usage** | Users can access directly from a browser | Users have to install the app on their device of choice | Users have to install the app on their device of choice |
| **Internal working** | Client code in the browser communicates with remote server-side code and databases | Client code and browser code wrapped in a native shell or container | Client code written in technology and language specific to the device or platform it will be installed on |
| **Native device features** | Not accessible | Accessible | Accessible |
| **User experience** | Inconsistent and dependent on the browser being used | Consistent and engaging | Consistent and engaging |
| **Access** | Limited by browser and network connectivity | One-step access with offline features | One-step access with offline features |
| **Performance** | Slower and less responsive | Faster, but may consume more battery power | Performance can be optimized to device |
| **Development** | Cost-efficient, faster time to market | Cost-efficient, faster time to market | Expensive, slower time to market |

## Choosing the Right Option:

The best type of app for your needs depends on

* **your budget,**
* **development timeline,**
* **desired features, and**
* **target audience**

Native Apps: Ideal for complex apps requiring **full device functionality and top-notch UX** ( games, photo editing).

PWAs: Great for simpler apps, news outlets, or when you need a fast and **cost-effective solution** with **offline capabilities.**

Hybrid Apps: A good option if you need **an app for both iOS and Android** with a balance between development speed and features.

# Reviewing and Comparing Mobile App Programming Languages

Choosing **the right** programming language for your mobile app development project is crucial. Here's a breakdown of popular languages across native, cross-platform, and progressive web app development:

## 1. Swift

Developed by Apple, swift is the go-to language for iOS app development. Its clean syntax and powerful features make it easy to write and maintain code.

Swift offers safety features like **optional and automatic memory management, reducing the chances of runtime errors**. Popular apps like Airbnb and LinkedIn use Swift for their iOS versions due to its performance and reliability.

## 2. Kotlin

Kotlin has gained popularity for Android app development due to **its interoperability with Java and concise syntax**. It offers features like **null safety and extension functions, enhancing productivity and code safety**. Apps like Pinterest and Trello have embraced koltin for its modern language features and seamless integration with existing Java codebases.

## 3. Java

As a long-standing player in the mobile app development scene, java remains **a robust choice for Android app** development. Its platform independence, strong community support, and extensive libraries make it a versatile language. Apps like WhatsApp and Instagram rely on Java for their Android versions due to its stability and scalability.

## 4. Dart

Dart, paired with Google’s Flutter framework, enables **cross-platform app developmen**t with native-like performance. Its **Just-in-Time compilation** allows for fast development cycles, while Ahead-of-Time compilation ensures efficient production builds. Flutter powers apps like Alibaba and Google Ads due to its fast development pace and expressive UI capabilities.

## 5. JavaScript

React Native leverages JavaScript to build **cross-platform mobile apps** with a **single codebase**. Its **component-based architecture** and **hot reloading feature accelerate development**. Popular apps **like Facebook and Instagram use React Native** for their ability to deliver native-like performance and seamless user experiences a cross platforms.

## 6. C#

Allows developers to build native Android, iOS, and Windows apps with **a shared codebase**. Its **strong integration with Visual Studio and access to native APIs ensure high performance** and platform-specific functionalities. **Apps like UPS and Alaska Airlines rely on Xamarin** for its code-sharing capabilities and native user experiences.

## 7. Objective-C

While Swift has largely replaced **Objective-C for iOS development**, it remains relevant for **maintaining legacy codebases**. Objective-C **offers dynamic messaging and runtime reflection, allowing for flexible app development**. **Apps like Airbnb and Uber** initially relied on Objective-C **before transitioning to Swift.**

## 8. Python

Python’s **simplicity and versatility** extend to mobile app development, particularly with frameworks **like Kivy and BeeWare.** Its **readable syntax and extensive libraries facilitate** rapid prototyping and development. Apps like Instagram and Dropbox use Python for backend services and automation tasks, showcasing its flexibility beyond mobile development.

### Others languages like Ruby , LISP, Malboge, Go exist.

### Native App Development:

### Languages:

* iOS: Swift, Objective-C
* Android: Java, Kotlin

### Cross-Platform App Development:

### Languages:

* JavaScript: Widely used for its large developer community and ease of learning.
* **C#**: Popular for enterprise-level apps with its mature framework and integration with Microsoft tools.

### Progressive Web App (PWA) Development:

* Languages: HTML, CSS, JavaScript

## Choosing the Right Language:

It is essential to prioritize versatility, platform compatibility, performance, and security.

* Python stands out for its adaptability.
* gains ground for Android app development over Java.
* JavaScript excels in creating both websites and mobile apps.
* Swift remains the go-to choice for iOS app development
* Java remains strong for cross-platform and corporate applications.
* Rust emerges as a formidable option for its combination of performance and security.

## Conclusion:

There's no one-size-fits-all solution. By understanding the strengths and weaknesses of each language and its suitability for your project requirements, you can make an informed decision that optimizes development time, cost, and user experience.

# 3. Review and compare mobile app development frameworks by comparing their key features and where they can be used.

Building a mobile app requires selecting the most suitable framework. This guide compares popular frameworks based on key features to assist you in making an informed decision

## A Comparison of Mobile App Development Frameworks

## Swift (Native)

Language: Native (Swift)

Performance: Excellent (Native)

Cost & Time to Market: Lower (Native development)

UX & UI: Flexible (Native UI components)

Complexity: High (Native development complexity)

Pros: Ideal for complex, high-performance iOS apps due to its native development approach and excellent performance. Large and active developer community provides ample resources and support.

Cons: Steeper learning curve due to the use of Swift language and complexity of native development. May require a more experienced development team, potentially impacting costs.

## Kotlin Native

Language: Native (Kotlin)

Performance: Excellent (Native)

Cost & Time to Market: Lower (Native development)

UX & UI: Flexible (Native UI components)

Complexity: High (Native development complexity)

Pros: Excellent choice for performance-critical Android apps leveraging the power of Kotlin language and native development. Growing developer community offers increasing support.

Cons: Similar to Swift, Kotlin Native requires a higher level of development expertise and might not be suitable for simpler apps due to its complexity.

## Xamarin

Language: C#

Performance: Excellent (Native-like)

Cost & Time to Market: Moderate (Higher development cost)

UX & UI: Flexible (Native UI components)

Complexity: Moderate (C# knowledge needed)

Pros: Established framework with a large and active community supporting C# development. Well-suited for complex enterprise-level cross-platform apps due to its robust features.

Cons: Development costs might be higher compared to some cross-platform options due to the potential need for C# expertise.

## Ionic

Language: Web technologies (HTML, CSS, JS)

Performance: Good (Web-based)

Cost & Time to Market: Lower (Faster development)

UX & UI: Limited (Web-like UI)

Complexity: Lower (Web development skills)

Pros: Faster development due to the use of web technologies, potentially reducing costs. Large and active developer community provides ample support for web development skills.

Cons: Limited ability to create a fully native user experience and user interface compared to native frameworks. Performance might not be ideal for complex apps.

## React Native

Language: JavaScript

Performance: Excellent (Native-like)

Cost & Time to Market: Moderate (Faster development)

UX & UI: Flexible (Customizable UI)

Complexity: Moderate (Learning curve)

Pros: Popular framework with a large and active community. Enables faster development of cross-platform apps with good performance due to JavaScript usage.

Cons: Learning curve associated with React Native concepts might add complexity for developers unfamiliar with the framework.

## Flutter

Language: Dart

Performance: Excellent (Native-like)

Cost & Time to Market: Moderate (Faster development)

UX & UI: Flexible (Customizable UI)

Complexity: Moderate (Learning Dart)

Pros: Uses Dart for code reusability across platforms, potentially leading to faster development. Offers excellent performance comparable to native apps. Growing developer community provides increasing support.

Cons: Learning Dart, a new language for some developers, adds complexity. Might not be as established as React Native yet.

## Choosing the Ideal Framework:

The optimal framework selection hinges on your project's specific requirements. Here's a quick guide:

Complex, high-performance native apps: Prioritize native development frameworks like Swift or Kotlin Native.

Enterprise-level cross-platform apps: Consider Xamarin's established nature and familiarity for C# developers.

Simple to moderately complex cross-platform apps: Explore React Native, Flutter, or Ionic for their development speed and cross-platform capabilities.

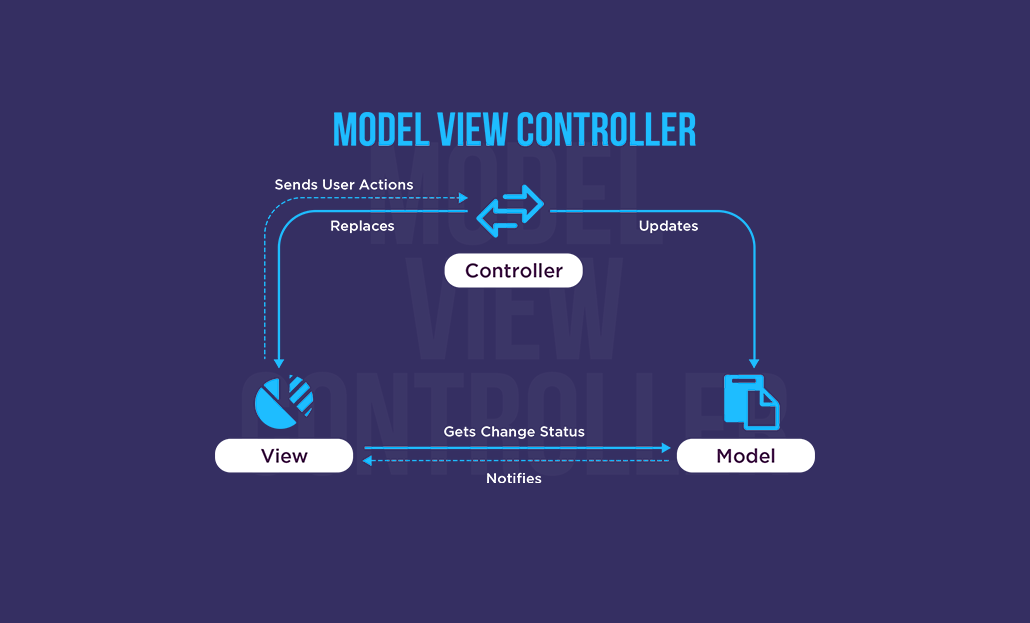
# 4. Study mobile application architectures and design patterns

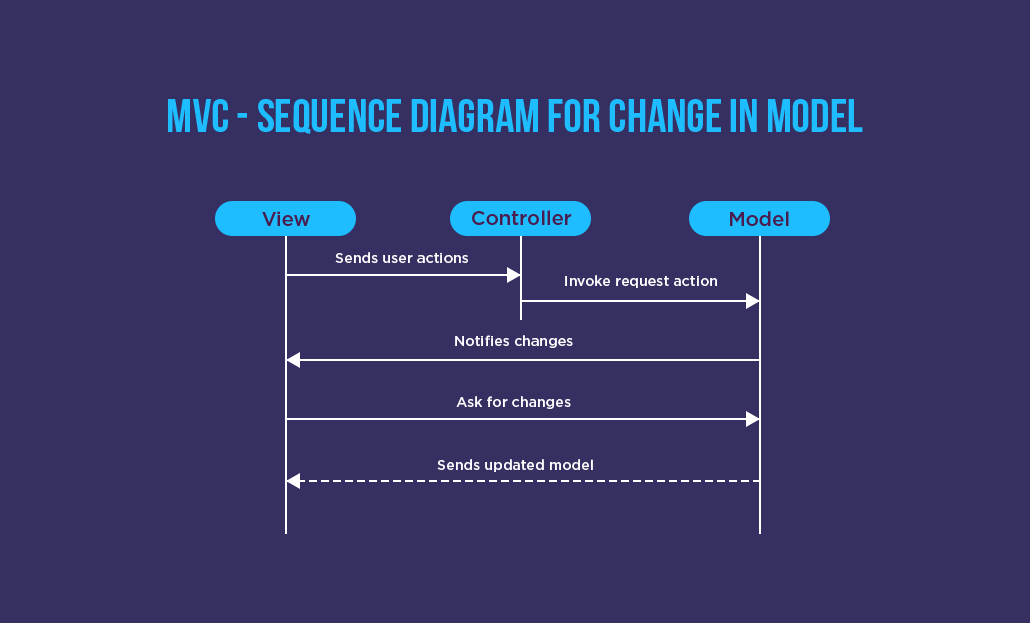
Architectures and Design patterns are well-established solutions to common problems encountered in software development, and mobile app development is no exception. They offer a structured approach to building mobile apps, promoting reusability, maintainability, and efficiency. Here's a breakdown of some key design patterns used in mobile app development:

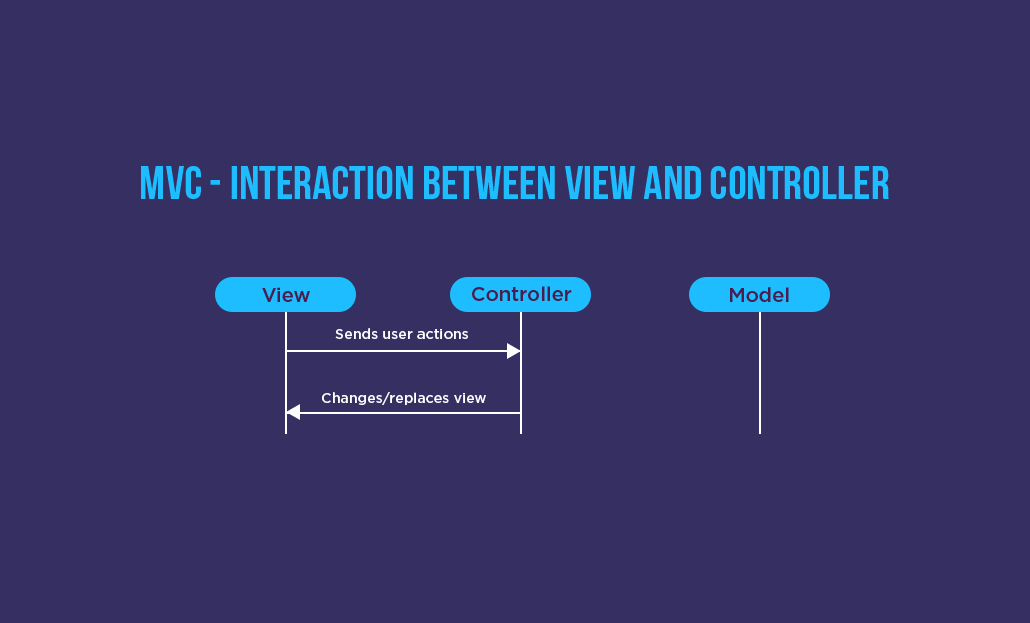
## Architectural Patterns:

1. Model-View-Controller (MVC): This classic pattern separates the application logic (Model) from the user interface (View) and the controller that handles user interactions (Controller). This promotes cleaner code and easier maintenance.

The MVC is the first approach introduces in the 1970s

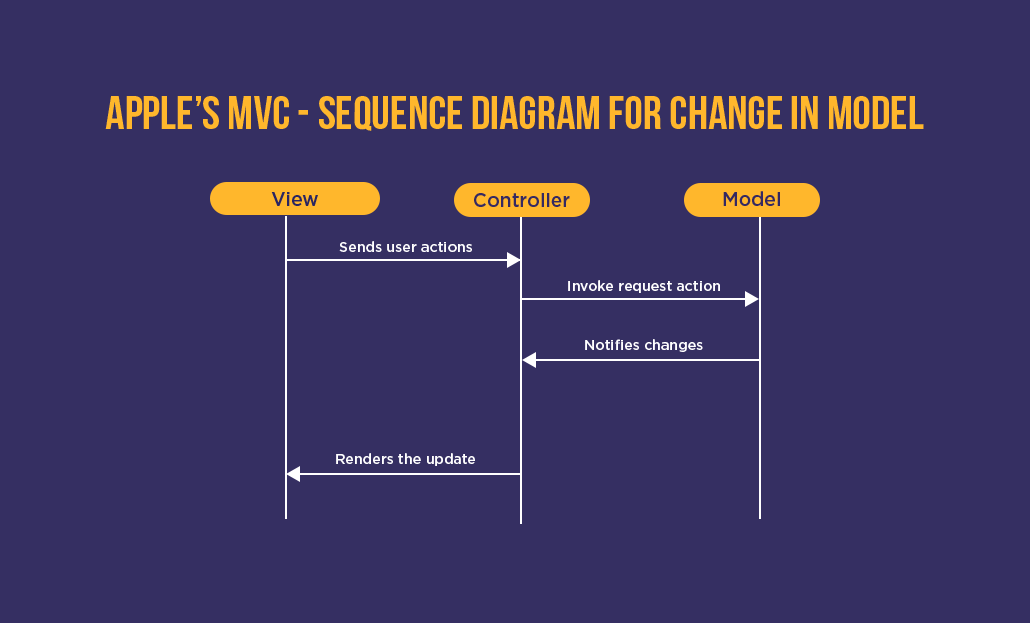




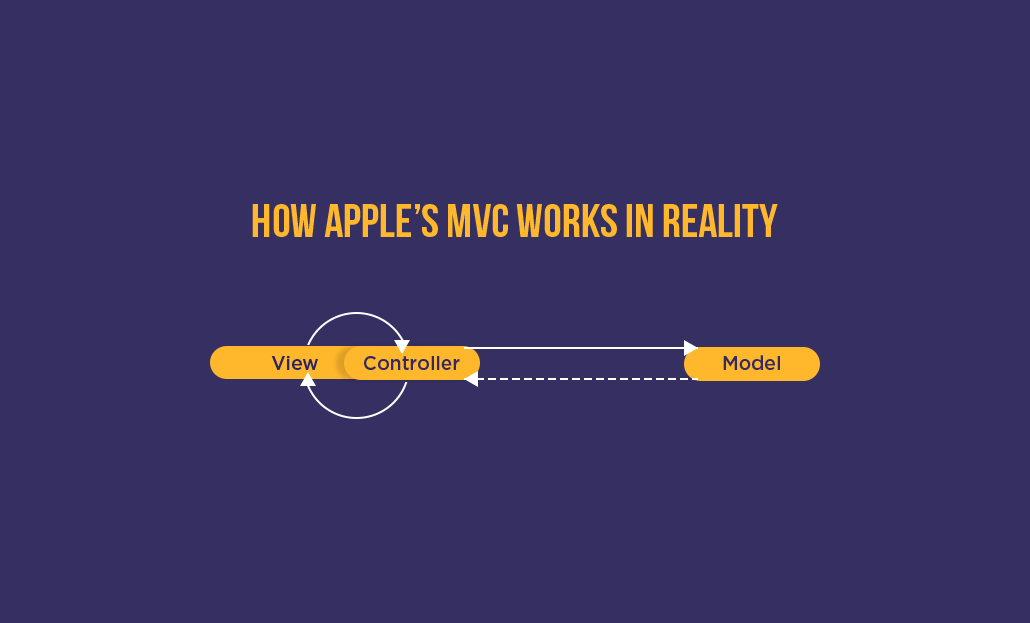


Disadvantages of MVC Design Pattern: All three components are tightly coupled, and this dramatically affects the reusability of each component since each component knows about the other two

### Apple’s MVC Design Pattern(variant)

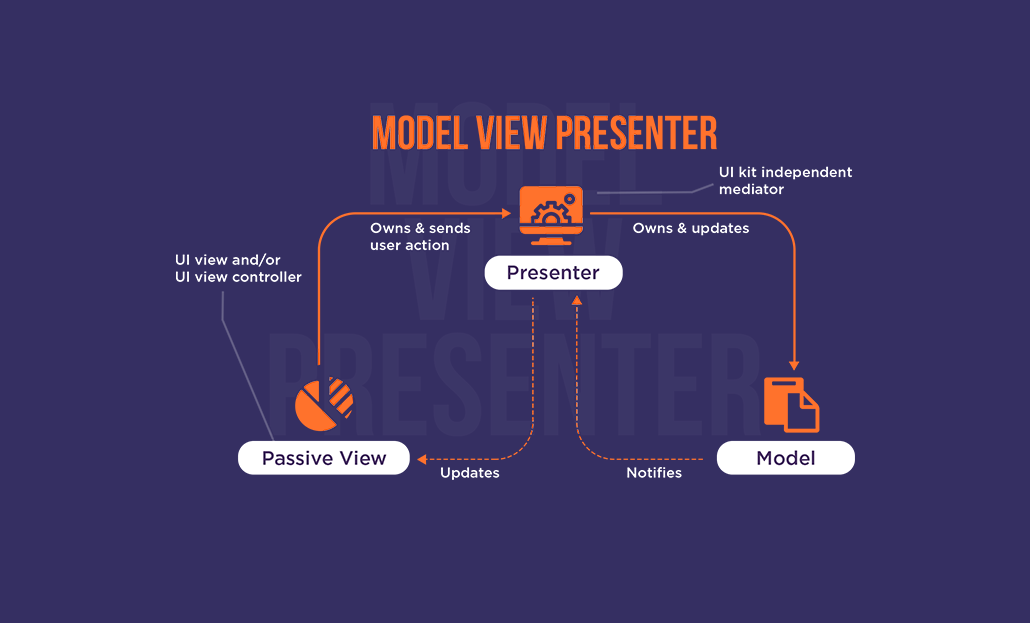


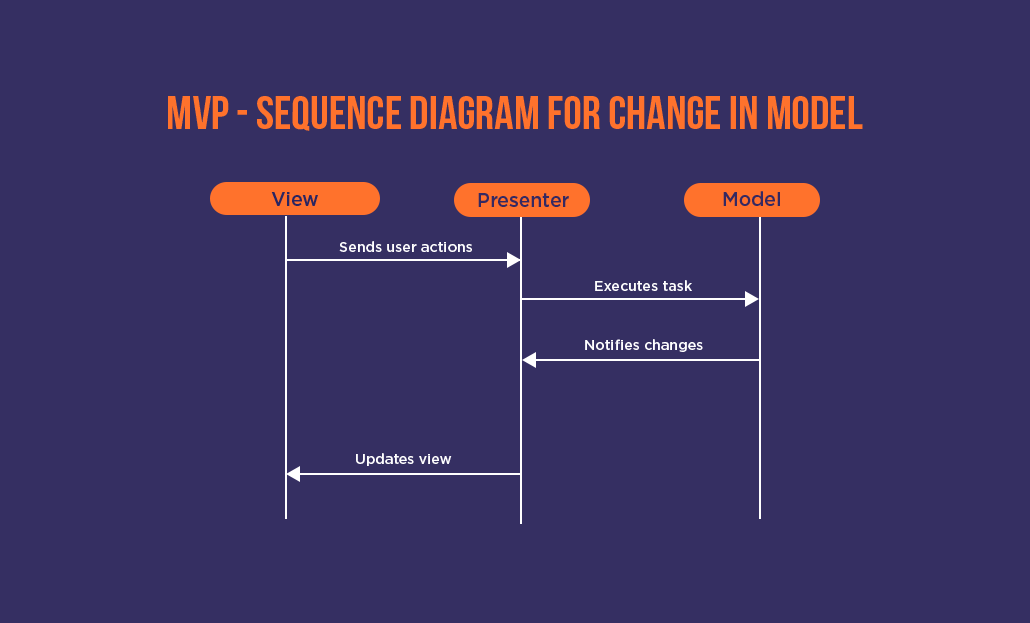
Advantage over MVC: In this design pattern, there is no direct connection between the Model and the View. In addition, the controller includes the process logic of representation



Disadvantage: when the View is tightly coupled with the controller, it becomes harder for the unit test

2. MVP (Model-View-Presenter): Similar to MVP separates concerns but uses a Presenter component to handle business logic and data presentation for the View. This pattern can be useful for complex user interactions. Introduced in the 1990s.





### Advantage over MVC:

There is no direct communication between the Model and the Passive View

The Presenter doesn’t involve in the lifecycle of the View

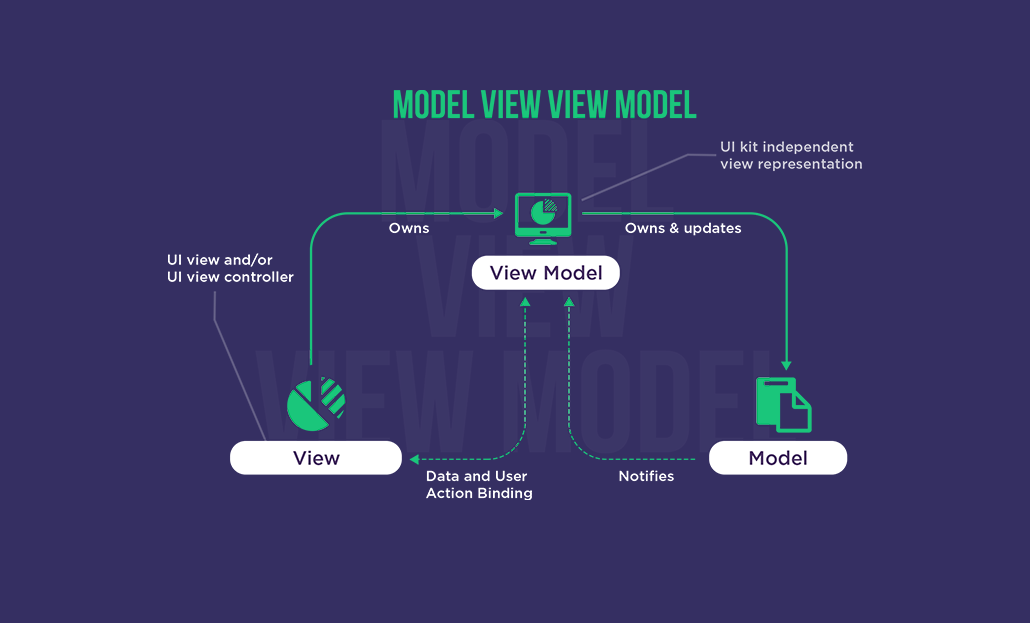
The Passive View doesn’t aware of the existence of the Model and Presenter and the passivity enhances the testability.

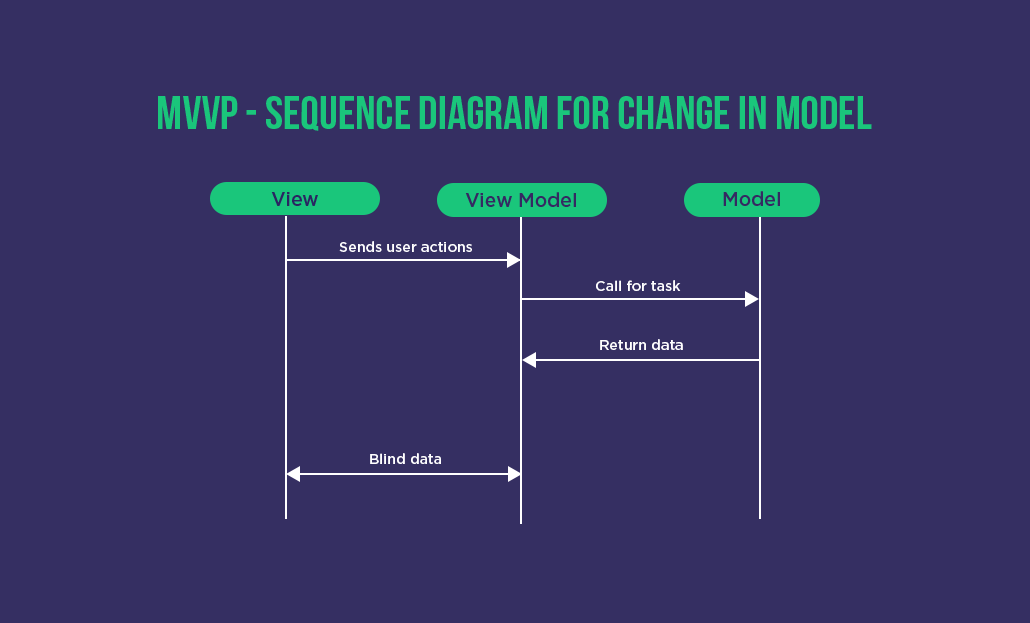
### Disadvantage of MVP:

Results assembly problem because of having three separate layers

Note – Supervising Controller pattern solves the binding issue by enabling binding between the Model and View, but again it becomes similar to classic MVC architecture and includes all those concerns.

3. Model-View-ViewModel (MVVM): John Goosman announced this pattern in 2005, An extension of MVC, MVVM introduces a ViewModel that acts as an intermediary between the Model and the View. The ViewModel prepares data for the View and handles user interactions without complex logic in the View itself. This is a popular pattern for mobile development frameworks like Android's Jetpack.





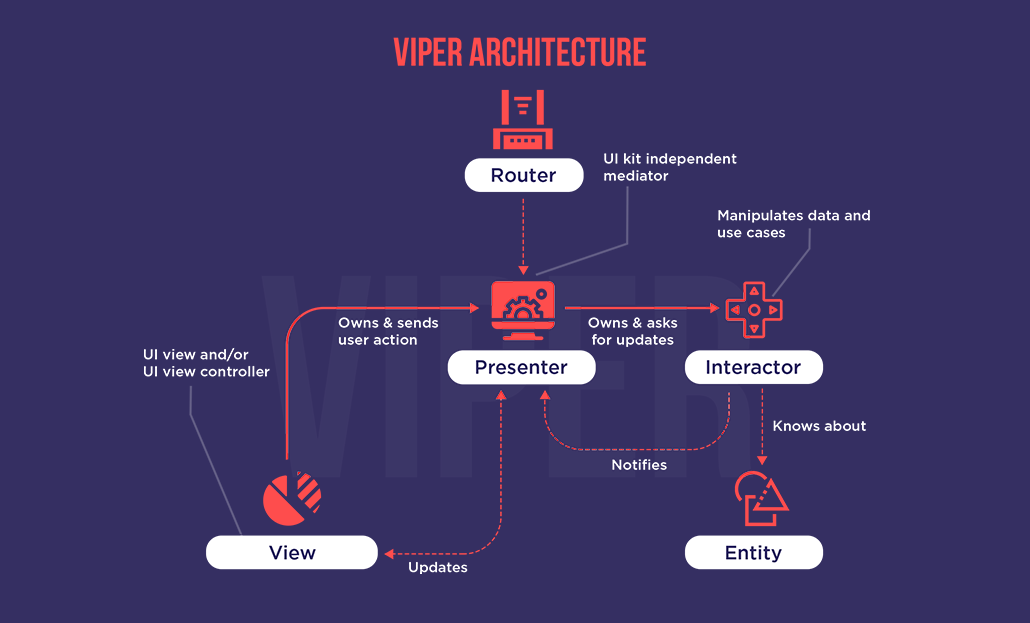
Advantage: Since one-way communication from View to the ViewModel reduces the lines of code required for synchronizing View and ViewModel.

## Other Important Patterns:

Singleton: This pattern ensures only one instance of a particular class exists throughout the application. It's useful for managing global application state or resources like a network connection. (Use with caution as overuse can lead to tight coupling)

Adapter: This pattern allows incompatible interfaces to work together. For example, an adapter can convert data from an external API to a format compatible with your app's internal model.

VIPER: The new Pattern VIPER (View Interactor Presenter Entity Router) entered the development world in 2012. It fuels the idea of separating responsibilities with five layers listed below



## Choosing the right design pattern

Here are some general considerations:

App Complexity: For simpler apps, MVC might suffice. More complex apps might benefit from MVVM or MVP for better separation of concerns.

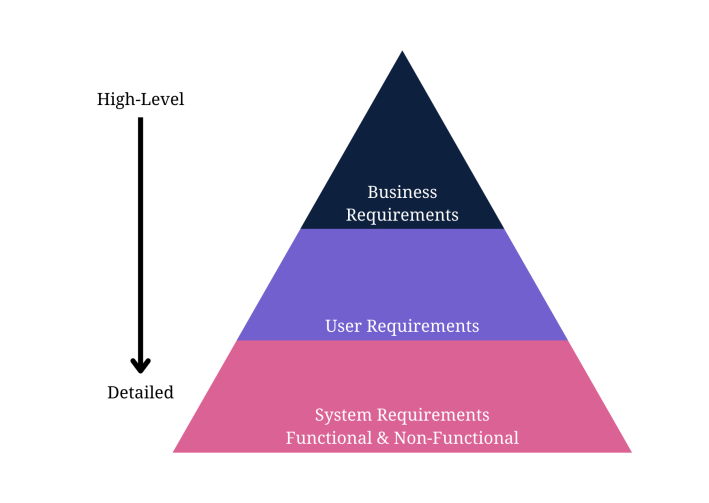
1. Maintainability: Design patterns promote cleaner code structure, making your app easier to maintain and update in the future.

2. Reusability: If component reuse is necesary

3. Developer Familiarity: Consider the development team's experience with different design patterns.

# 5. Study how to collect and analyse user requirements for a mobile application (Requirement Engineering)

Mobile apps have different requirement types to collect, including:



## Step 1: Define Your App Idea and Purpose

Mobile development requirements-gathering starts with a business idea. The first information you need is the idea or purpose of the mobile app.What purpose will it serve? Does it offer a solution to a potential problem?You need to identify a problem the app will solve to recognise the idea or purpose behind it.

## Step 2: Gather and Align the App and Business Objectives or Goals

This step encourages you to gather business requirements to understand how the enterprise aligns with the idea from the first step.Gathering business requirements to document involves these steps:

Identify the stakeholders for the right mobile application software development based on the business idea.

Define clear and concise business goals and objectives to understand the project’s scope.

Elicit stakeholder requirements and user requirements with elicitation techniques.

Document the requirements in a business requirements document.

Validate your requirements with stakeholders for a further transparent and opportunistic process.

## Step 3: Run a Market Analysis and Competitor Analysis

Conduct a market or competitor analysis to truly understand the user’s perspective and design the appropriate user personas.It also helps your team gather more user requirements for the development company.

The following steps explain the process of gathering user and competitor requirements:

Identify the direct, indirect, secondary, and substitute competitors for the mobile app. Remember to recognise any businesses offering similar mobile app services or products and those offering different products in a broader niche umbrella.

Gather competitor information, including products, descriptions, pricing structures, geographic reach, engaging promotions, target market positioning, business reputation, user profiles, and key partnerships to understand what your product needs to compete against.

Use an analytic technique fro example, SWOT analysis to determine your competitor’s strengths, weaknesses, opportunities, and threats. You could learn from another app’s mistakes to improve your requirements and identify possibly unique features other apps don’t provide.

Next, determine what competitive advantage you hold over other apps. The pricing is also better than competitors. Choose elements in which you wish to compete with other apps.

## Step 4: Determine a User Persona Scenarios

A user persona: the target users for the mobile app.It should describe the ideal person who uses the app, with some flexible aspects for alternate users.The ultimate user person could include the following details about target users; Age (also, typical generational qualities), Behavioural considerations, Gender (including non-binary if relevant to the product), Geographic location, Goal or problem the app addresses or solves, Goal quotes or principles, Goal-related frustrations, Motivation to use the app, Range of hobbies and daily activities, Typical occupation range

## Step 5: Gather and Prioritise Functional and Non-Functional Requirements

Your user and business requirements are shaped through the initial steps of app requirements gathering.

examples of **functional** mobile requirements:

* How the app allows users to sign up, verify accounts, or subscribe to a newsletter.
* Buttons and dashboards users interact with to complete a specified task.
* External and internal interfaces users interact with on the app.
* The necessary administrative functions for different user classes.

examples of **non**-**functional** requirements in mobile development:

* How fast the app responds to user input.
* How the app protects user and business data.
* Whether the app can work on multiple Platforms.
* How much data does the app store and is it scalable?
* How reliable and maintainable the app remains.
* Does the app comply with local laws and regulations?

Include requirements for **user experience (UX) and user interface (UI)** with your functional and non-functional requirements.

## Step 6: Design Use Cases and other UML diagram

A mobile system requirements document won’t be complete unless you add use case. Design them before documenting the specifications for the development company or team.Use cases to add visual representation to your documents for mobile app development documentation

## Step 7: Write an App Requirements Document

Delivering a proper app requirements document means you need to know how to write a mobile app requirements document. Mobile application development relies on the requirements document to design proper flow or the best app features and hit the right target audience.

Content;

Business requirements

User requirements

Software requirements specification

Technical specifications

Functional specifications

Non-functional requirements

Hardware interfaces

A list of must-have features

Unique app features

Internal and external interfaces

Non-functional key metrics

Acceptance criteria

## Step 8: Deploy Prototyping and Wireframing

Prototyping and wireframing let you design the user flow of user interfaces and basic app functions.It also lets you test and validate layouts and transitions between app pages.

Step 9: Validate the App Requirements

Validation is a quality control process you use before launching the final product based on your requirements.The prototype app collects feedback from stakeholders, and you can invite stakeholders to verify that the app meets the documented requirements. Use the feedback for the final step.

## Step 10: Apply Agile Methodology

Agile methodology in requirements-gathering means you’ll always adapt the requirements document as per the feedback from stakeholders, testing, and initial product releases.

Agile methodology focuses on user experience and constant testing and validation to further improve your application.

# 6. Study how to estimate mobile app development cost

Let’s break down the different stages involved in a mobile app development project. This way we’ll get a good understanding of how app development costs are formed.

## 1. Initial Development Phases - Cost Impact

Discovery: Low cost - Lays the foundation for the entire project, minimizing rework and development time later.

Design (UI/UX): Moderate cost - Defines the app's look and feel, impacting development complexity. Poor design choices can lead to costly revisions.

Development & Infrastructure: High cost - Most time-intensive stage, involving coding, integrations, and infrastructure setup. Complexity directly affects development effort and cost.

Testing & Quality Assurance: Moderate cost - Crucial for a polished app but can be expensive if bugs are found late in development.

Publishing: Low cost - Relatively inexpensive to submit the app to stores, but marketing costs for user acquisition are separate.

## 2. Ongoing Maintenance Costs

Maintenance typically costs 15-20% of the total development cost annually. However, the first year can be as high as 50%.Costs include;

* Hosting fees to keep the app server operational.
* Customer support to address user inquiries and resolve issues.
* Regular updates to add new features, improve functionality, and fix bugs.
* Support for the latest versions of third-party services integrated into the app.
* Performance updates to optimize speed, stability, and user experience.

## 3. Additional Costs to Consider in long term

User acquisition and engagement (marketing expenses)

App content localization and internationalization (translation and cultural adaptation)

Legal and intellectual property protection (trademark registration, copyright protection, etc.)

Scalability and future development (upgrading infrastructure in case of growth and adding features)

## 4. Factors Influencing Cost

I. Development method (traditional vs. no-code vs. low-code)

- Traditional Coding (Highest Cost): involves writing code from scratch using programming languages,Offers most flexibility but requires skilled developers.

- No-Code (Lowest Cost): Enable the creation of functional apps without the need for extensive coding knowledge, pre-built components and visual interfaces that allow users to create apps by simply dragging and dropping elements. Quick and affordable, but limited customization.

- Low-Code (Mid-Range Cost): lies between traditional coding and no-code development Balances customization with faster development.

II. App requirements (complexity, features, screens): More complex features (login systems, payments, social media feeds) and advanced UI elements increase development time and cost.

III. Industry-specific requirements: requirements like security for healthcare apps can add complexity and cost.

IV. Licensing and third-party integrations: Integrating external services (payment gateways, maps) adds development and potentially licensing fees.

V. Backend infrastructure: Building a robust backend to handle user load and future growth impacts cost.

VI. Security and data protection measures: Implementing strong security measures to protect user data is crucial and can be expensive.

VII. Target audience platform (iOS vs. Android)

VIII. Team and Contract Types: Team size, skills, and seniority, Contract type (fixed term vs. ongoing)

IX. Location and outsourcing (affect pricing models like hourly rate vs. fixed cost)

## 6. Conclusion

Consider the entire app lifecycle cost, not just initial development. This helps with budgeting and finding the right development partner for your business needs. Remember, an app is an investment, and cost is a major factor in its success.

# References

<https://aws.amazon.com/compare/the-difference-between-web-apps-native-apps-and-hybrid-apps/>

<https://ellow.io/best-programming-language-for-mobile-app-development/>

<https://www.choicely.com/blog/how-to-estimate-your-total-app-development-costs-free-app-cost-calculator>