REPUBLIC OF CAMEROON REPUBLIC DU CAMEROUN

Peace-Work-Fatherland Paix-Travail-Patrie

University of Buea Universite de Buea

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**DEPARTMENT OF COMPUTER ENGINEERING**

**CEF440: Internet Programming and Mobile Programming**

Mobile Application Development Processes

**PRESENTED BY MEMBERS OF GROUP 9**

|  |  |
| --- | --- |
| **MATRICLE** | **NAME** |
| FE21A128 | AHOUMO TEMATEU ROXANE PHILIPPINE |
| FE21A152 | BOUCHUKE BABILA DANIEL |
| FE21A166 | DERRICK MBUNBO FORCHA |
| FE21A325 | TIANI PEKINS EBIKA |
| FE21A303 | SAMUEL OSH |

SUPERVISED BY **Dr NKEMENI VALERY**

ACKNOWLEDGMENTS

Acknowledging the assistance of DR NKEMENI VALERY the course instructor and an elderly, for his marvelous work and knowledge he has mounted upon me through this years of studies, my team and some close friends for their guidance and care .may the good LORD continue to guide and protect you all. Thanks

Table of content

1 INTRODUCTION

1. REVIEW AND A COMPARATIVE ANALYSIS OF MAJOR MOBILE APPLICATIONs (native, progressive web apps, hybrid apps)

**2**.1 What is a mobile app?  
  **2**.2 Classification of mobile applications

1. PROGRAMMING LANGUAGES USED IN MOBILE APPLICAATION

**3**.1Native Development (Platform-Specific) Languages:

* 1. Hybrid Development (Cross-Platform) Languages:
  2. Web Development languages:

1. REVIEW AND A COMPARATIVE ANALYSIS OF MOBILE APPLICATION DEVELOPMENT FRAMWORKS (features: language, performance, cost & time to market, UX & UI, complexity, community support) and where they can be used.

4.1mobile application frameworks

4.2 comparative analysis of framework used in mobile app development

5 MOBILE APPLICATION ARCHITECTURES AND PATTERNS

5.1 Mobile, app architecture

5.1.1 Why Is App Architecture Essential?

5.2 The Fundamental Layers of Mobile App Architecture

* 1. Top Architecture Design Pattern: MVC, MVP, and MVVM
  2. The Key Principles of Good App Architecture
  3. How to Choose the Right Architecture for Your Application
  4. **Mobile application Design pattern**

5.6.1What are Design Patterns?

6 WAYS AND METHOD TO COLLECT AND ANALYSE USER REQUIREMENTFOR MOBILE APPLICATION (Requirement Engineering)

* 1. Why is requirement engineering so hard?

6.2 state of the art for research

6.3 Research strategies

7 HOW TO ESTIMATE THE COST OF A MOBILE APPLICATION

* 1. Factors Influencing Mobile App Development Costs

7.2 Methods for Estimating Mobile App Development Costs Fixed Price Model.

7.3Tips for Accurate Cost Estimation

1. CONCLUSION

9 RECOMMENDATIONS

10 REFERENCES

Table of figures

[Figure 1: Summarized table showing the comparative analysis of the major frameworks 5](#_Toc155685950)

[Figure 2: Layers of three-tier architecture and dependencies 5](#_Toc155685951)

[Figure 3: Model-view-control architecture 5](#_Toc155685952)

[Figure 4: Model view presenter architecture 5](#_Toc155685953)

[Figure 5:Model-view-viewmodel architecture 5](#_Toc155685954)

[Figure 6:Singleton design partern 5](#_Toc155685955)

[Figure 7:Observer pattern 5](#_Toc155685956)

[Figure 8: Deligation pattern 5](#_Toc155685957)

**1 INTRODUCTION**

In the last 20 years, mobile phones have become increasingly popular and are now everyday objects. Additionally, the steady advance of technology means that devices are getting better all the time and can, on account of improved features, run more complex applications. The first applications in mobile phones were native and dependent on the mobile device and its operating system which are different to the latter’s of todays. Additionally, Internet access in the early devices was very limited and expensive. Therefore, applications were confined to the use of in-phone resources like SMS, camera, music files and personal information. The release of the iPhone in 2007revolutionized the mobile phone world, as it included the possibility of executing more complex applications, known as mobile applications, and provided access to the Internet. The term smartphone refers to this kind of advanced mobile phones. Connectedness is a key characteristic of mobile apps, leading to an explosion in data traffic (4932% increase from 2006 to 2009 in the USA, according to AT&T) .The use of smartphones has now grown so much that some users regard them as a basic necessity. Star mobile applications like instant messaging have users using their phones at any time of the day or night. The rate of mobile applications and development in the overall software development arena is getting bigger and bigger over recent years. However, the information available about the software development processes applied is scarce.

2 **REVIEW AND COMPARATIVE ANALYSIS OF MAJOR MOBILE APPLICATIONs**

**2.1 What is a mobile app?**  
A mobile app is a software application designed to run on mobile devices, such as smartphones and tablets, offering various functionalities, from entertainment to productivity. Mobile apps are developed for the specific mobile device’s operating system, primarily Android or iOS, but some mobile apps can be hybrid Mobile which can be downloaded from app stores or accessed through web browsers like web-based apps.

**2.2 Classification of mobile applications**  
Mobile apps can be categorized in multiple ways, such as by platform, target audience, content type etc. Arguably, the most important categorization criteria is that of technology - this influences the app's performance, user experience, and compatibility with other devices.  
There are 3 types of mobile apps based on technology: **native apps, web apps, and hybrid apps**.  
  
a.) **Native Applications**  
a native app is developed specifically for one platform or operating system, using the development tools and language that the platform supports. ). These apps are directly installed on a mobile device and can take full advantage of all device hardware, such as the camera, GPS, accelerometer, and more. Native apps are designed to provide a seamless user experience, adhering to the guidelines and design principles of the platform for which they are developed. **Instagram and Google** map are popular example of a native app.

**Advantages of native based mobile applications**

* **Optimized performance**: Native apps are known for their speed and responsiveness. A native app is compiled into machine code, which makes it much faster and more efficient than its web and hybrid counterparts.
* **Better user experience**: By adhering to specific platform guidelines, native apps offer a user experience that matches the operating system's look and feel, providing a more intuitive and integrated interface thereby enhancing user engagement.
* **Full access to device hardware**: Native app development allows apps to utilize the full spectrum of device-specific features such as GPS, camera, microphone, and gestures, enabling richer functionalities and interactions.
* **Offline functionality**: Unlike web apps that require an internet connection, native apps can offer full or partial offline access to content and features, enhancing usability in scenarios with limited or no connectivity.

**Disadvantages of a native app**

* **Higher development costs**: Developing native apps for multiple platforms can be costly since it requires creating separate versions for each OS, increasing the effort, time, and resources needed.
* **Longer development time**: Building an app from scratch for different operating systems can significantly extend the development timeline, especially if you aim to maintain feature parity across platforms.
* **More complex maintenance and updates**: Each native app version needs to be updated and maintained separately, which can complicate the process of rolling out new features or fixes across all platforms simultaneously.
* **Platform dependency**: Native apps are tied to the platform they are developed for, which can be a limitation if users switch to a different OS, potentially reducing the app's reach.

B) **Web based applications**  
 web app is a website that is designed to look and feel like a native app but is accessed through a web browser, such as Chrome or Safari, on a mobile device. A web app is built using web technologies like HTML5, CSS, and JavaScript**. Google Docs** is a popular example of a web app

**Advantages of a web app**

* **Cross-platform compatibility**: A web app can be used on any device with a web browser, eliminating the need to develop separate app versions for different operating systems.
* **Lower development costs**: Since web apps are developed with standard web technologies and do not require platform-specific development, the cost of building and maintaining web apps is generally lower than native apps.
* **Reduced storage space on user devices**: Since a web app is accessed through a browser and does not need to be installed on the user's device, it consumes less storage space. This can be particularly advantageous for users with devices that have limited storage capacity.

**Disadvantages of a web app**

* **Limited access to device features**: A web app cannot directly access device hardware and features (such as the camera, accelerometer, or GPS) as extensively or efficiently as a native app. This can limit functionality.
* **Dependence on internet connectivity**: Most web apps require an active internet connection to function, which can limit their use in areas with poor or no connectivity.
* **Browser compatibility issues:** A web app must be designed to work across different browsers, which can lead to compatibility issues and inconsistencies in appearance and functionality across devices and browsers.

C) **Hybrid app**  
 A hybrid app is a blend of native and web apps. A hybrid app is built using web technologies (like HTML, CSS, and JavaScript that you may use for a web app) and then wrapped in a native container using platforms like Cordova or React Native. This approach allows a hybrid app to be installed like a native app but still use web app technologies. Therefore, hybrid mobile app approach offers a mechanism to turn browser-based apps into mobile apps that could be compiled into binary executable files, could be downloaded from App stores onto users’ devices, and could be run both online and offline. **Evernote and Uber** are examples of hybrid apps.

**Some Advantages of a hybrid mobile applications include;**

* **Cross platform app development**: Hybrid apps are built using a single codebase, which can significantly reduce development time and costs by allowing developers to deploy the app across multiple platforms (iOS, Android, etc.) without writing platform-specific code. Cross-platform app development significantly reduces development time and costs.
* **Access to device features**: Despite being developed with web technologies, hybrid apps can access device hardware and features (camera, GPS, accelerometer, etc.) through the native container, enabling functionality that is closer to that of native apps.
* **Ease of updates**: Similar to a web app, updating hybrid apps can be simpler than native apps since many updates can be pushed directly to the web portion of the app without requiring users to download an update from the app store.

Whereas it drawback are as follows

* **Performance limitations:** While a hybrid app can offer good performance, it may still lag behind a native app in terms of speed and responsiveness, especially for high-performance applications or those requiring advanced graphics.
* **User experience compromises**: Achieving a native-like user experience can be more challenging with hybrid apps. Differences in UI elements and navigation patterns can make the app feel less integrated with the device's operating system.
* **Complexity in accessing device features**: Although hybrid apps can access device features, doing so often requires additional plugins or frameworks. This can introduce complexity and potential issues with compatibility or performance.
* **Dependence on third-party platforms**: The development and maintenance of hybrid apps depend on third-party platforms and tools (e.g., Cordova, Ionic). Changes or discontinuation of these tools can require significant adjustments or even impact the app’s viability.

1. **PROGRAMMING LANGUAGES USED IN MOBILE APPLICAATION AND THEIR COMPARISON**

Choosing the right language for your mobile app depends on your target platform, project needs, and desired functionalities which is beneficial throughout the development process. For a language to be compared, it must be categorised. Mobile app programming languages can be categorized into their;

* Development Approach:
* Functionality:

Below is a comparison of languages used in mobile app development with their advantages and disadvantages

* 1. **Native Development (Platform-Specific) Languages:**

**Swift (iOS):**

-Modern, concise, and allows for high-performance apps.

-Steeper learning curve compared to some options.

**Objective-C (iOS):**

-Legacy language for iOS, but Swift is increasingly preferred.

**Java (Android):**

-Mature language with a vast developer community.

-Can be verbose.

**Kotlin (Android):**

- Google's recommended language for modern Android development.

- More concise and interoperable with Java.

* 1. **Hybrid Development (Cross-Platform) Languages:**

**JavaScript (with frameworks like React Native):**

-Leverage your JavaScript skills to build apps for iOS, Android, and potentially other platforms.

-Large developer pool and fast development cycles.

-Performance might be a concern for complex apps.

**HTML5/CSS3/JavaScript (with frameworks like Ionic):**

-Primarily for web apps, but can be used for hybrid apps.

- Faster development but limited by reliance on web technologies and potential drawbacks in native -features and performance.

* 1. **Web Development languages:**

**HTML5/CSS3/JavaScript**: The foundation of web development. Offers a familiar development environment but might lack the performance and functionality of native apps.

**4 REVIEW AND A COMPARATIVE ANALYSIS OF MOBILE APPLICATION DEVELOPMENT FRAMWORKS**

4.1 -**Mobile application frameworks**

They are pre-built toolkits that help developers construct mobile apps efficiently. They are software libraries or collections designed to support the development of mobile apps for specific environments. We can see them as skeletons that hold up the numerous connecting part of mobile development. They are essential while building projects by helping developers avoiding starting from scratch to add, Framework also reduces dependency on programming languages.

Example of mobile development framework include React Native, java, Ionic and many more which is classified under three parts: Native, Web, and Hybrid. For specificity, each framework for mobile app development has it function and where they are best served.

* **Native Frameworks:** These frameworks allow you to build apps specifically for a particular platform (iOS or Android) thus providing the best performance and access to all the device's features .it requires a mastery on platform-specific programming languages (Swift for iOS and Java/Kotlin for Android).
* **Hybrid Frameworks:** These frameworks combine native and web technologies. You can use tools like HTML, CSS, and JavaScript to build the core app functionality, and then integrate native code modules for platform-specific features. Coding is done once and deployed on both iOS and Android .performance might be slightly slower than native apps.
* **Web App Frameworks**: These frameworks allow you to build web apps that can be accessed through a mobile device's browser. They offer the easiest and fastest development process but may lack some functionalities and responsiveness compared to native or hybrid apps.

**4.2** A summarized table showing the comparative analysis of the major framework used in mobile app development

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Native Frameworks** | **Web App Frameworks** | **Hybrid Frameworks** |
| Example | (e.g. React Native, Flutter)  React native for skype, Instagram etc. | (e.g. React, Angular for PWAs)  React for Facebook lite... etc. | (e.g., Xamarin, Ionic)  Xamarin for Uber driver app…etc. |
| Language | Platform-specific (Swift for iOS, Java/Kotlin for Android) or language specific (Dart for Flutter) | JavaScript (React), TypeScript (Angular) | Mostly web languages (HTML, CSS, JavaScript) with some native code integration |
| Performance | Best performance, full access to native device features such as camera | Lowest performance, relies on device browser capabilities | Good performance, might be slightly slower than native |
| CPU Usage | Generally lower CPU usage compared to hybrid frameworks due to direct access to native code. | Generally highest CPU usage as they rely on the device browser to render and execute the app. | Can have higher CPU usage than native frameworks because they rely on a layer of abstraction to interact with native components. |
| Complexity | Since it requires platform-specific knowledge or expertise in the language used (Dart for Flutter). It’s more complex to develop. | Simple to set up, uses standard web development skills. | Moderate complexity, requires understanding of web technologies and potentially some native development for specific features. |
| UX & UI | Native look and feel, best user experience | Limited to web UI elements and functionalities | Good UX/UI, can achieve a native look and feel but might require more personal efforts |
| Cost & Time to Market | Generally higher cost due to potentially needing separate developers for iOS and Android. Development time can be longer for complex apps. | Lowest cost and fastest time to market, but limited functionality might require more development effort later. | Lower development cost due to code reusability. Time to market can be faster. |
| Community Support | Large and active communities for both iOS and Android development | Very large and active web development communities. | Has a Growing community support, |

|  |
| --- |
| Figure(Table1) |

As shown from the table above with key features, native framework for developing mobile applications is Ideal for complex apps requiring high performance and full access to device features such as cameras etc, as to Hybrid mobile application framework which would be Suitable for medium-complexity apps where a native look and feel is important but development speed is a priority (e.g. business apps).In addition, for simpler apps, content-based apps, or internal enterprise tools where web-based functionality is sufficient , Progressive Web Apps (PWAs) Can be good thus offering app-like experiences through the browser.

**5 MOBILE APP ARCHITECTURES AND PATTERNS**

5.1 **Mobile app architecture**

Is a set of structural components that frame the system and its inner interactions, specifics of which are defined by the app’s features and requirements. We need to note that how smoothly and reliably an app runs depends significantly on the quality of its architecture.

**4.1.1 Why Is App Architecture Essential?**

Quality architecture helps with risk management and enables cost reductions. An application with robust, well-planned architecture is more likely to succeed in its target market. Any mobile app project starts with the planning and designing phase, and choosing the right architecture is a core priority. An insufficient approach to this step can slow down the development process and make it more extensive. It can also lead to various performance issues and system failures.

* Poor mobile app architecture may also lead to:
* Difficulties with development and maintenance
* Lower code readability
* Source code testing complications
* Greater exposure to errors, for a more detailed understanding, let’s review three core layers of mobile app architecture.

**5.2 The Fundamental Layers of Mobile App Architecture**

The multi-layer approach is widely used in mobile app development as it segregates the different, application-specific operational layers. Developing and executing each component separately allows developers to solve complicated matters quickly without changing the entire application. Here, the number of layers (or tiers) varies according to an app’s business and functional requirements, but the three-tier structure is the most common mobile app architecture pattern.

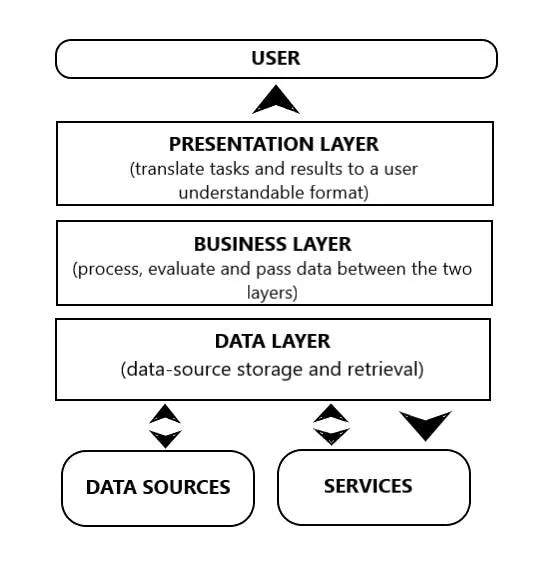


Figure 2: Layers of three-tier architecture and dependencies between them

* **Data layer**

The data layer is the heart of an application in the making and is comprised of data access components, server communications, and logic. This layer executes connections to the network and enables the storage of accrued information either locally or on the cloud. The patterns chosen for core operations will determine the application’s overall performance. Security and safety of data maintenance is a key priority at this level.

* **Business layer**

At this tier, all domain processes and operations take place as the business layer explains the logic, which, in turn, drives an app’s core functionalities. Business logic can take many forms, but it’s mainly responsible for business components and the corresponding data flow. Additionally, the business layer includes navigation methods, logging, caching, and other technical processes.

* **Presentation layer**

This layer is about how an app presents itself to the user and includes user interface (UI) elements such as themes, fonts, and colours and UI process components. The prioritization of features and functionalities also takes place on this level. Because end-users should be able to easily navigate in the app, the best practice is to keep the presentation level simple and avoid multi-level menus. With the multi-layer approach, the separately built dimensions function together smoothly as a complex system.

**5.3 Top Architecture Design Pattern: MVC, MVP, and MVVM**

There are myriad architecture patterns to choose from, but among the most popular are Model-View-Controller (MVC), Model-View-Presenter (MVP), and Model-View-ViewModel (MVVM).

These patterns are widely used to moderate complex codes and simplify UI code by making it more neat and manageable. MV(X) architectures divide the visualizing, processing, and data management functions for UI applications, which increases an app’s modularity, flexibility, and testability.

Let us take a closer look at each model to understand their differences.

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

MVC is commonly used when designing simple applications as it’s more readily modified than the other two and makes implementing changes to the app simple.MVC consists of three components: Model, View, and Controller. “Model” is subject to the app’s business logic and manages the state of an application and handles data changes and manipulations. “View” manages UI elements by presenting data to users and managing user interactions. “Controller” mediates between view and model by processing incoming requests. Depending on an app’s requirements, there may be one or more controllers.

* This pattern enables a faster development process and offers multiple views for the model.

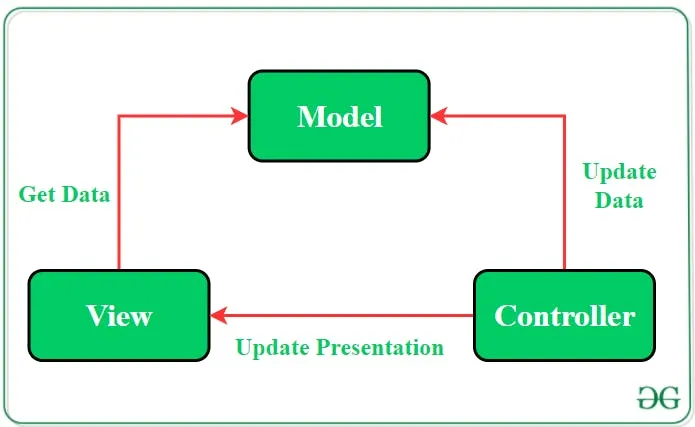


Figure 3 Source: Geeksforgeeks

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

MVP is derived from the MVC pattern, and here, the “controller” is replaced by “presenter”. Performance-wise this pattern offers high reliability as there is less hindrance than with the other two models in terms of rendering frames.

Similar to MVC, the model covers the app’s business logic and how data is handled while the view is separated from the logic implemented in the process. The presenter’s major function is to manipulate the model and update the view. When receiving input from a user via view, the presenter processes the data and sends the results back to view.

* MVP offers easier debugging and allows code reusability.

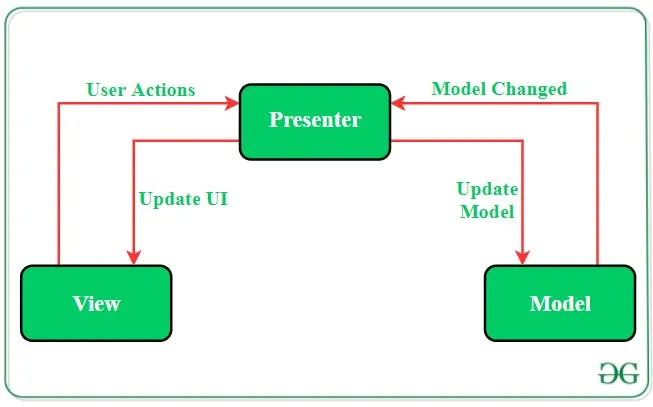


Figure 4 :Source: Geeksforgeeks

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

Designed for more explicit separation of UI development from business logic, MVVM is similar to MVC. The “model” here handles basic data and “view” displays the processed data. The View-Model element discloses methods and commands that help maintain the state of view and control the model. Two-way data binding synchronises models and properties with the view. Due to data binding, this pattern has higher compatibility than others.

* MVVM’s significant advantages include easier testing and maintenance as it allows developers to readily implement changes because the different kinds of code are separated.

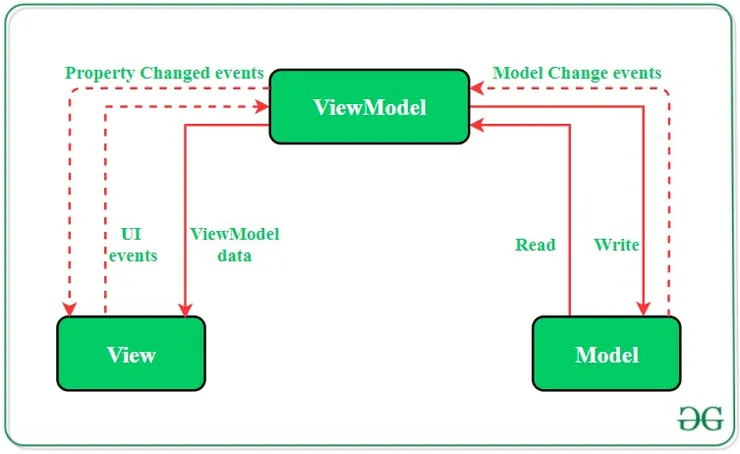


Figure 5 Source: Geeksforgeeks

To sum up, MVP and MVVM allow developers to break an app down into modular, single-purpose components. At the same time, these two patterns add more complexity to an application. If you intend to build a simple application with one or two screens, MVC may be a better solution. MVVM, meanwhile, works well in more complex applications that handle data from elsewhere, be it a database, file, web service, and so on.

**5.4 The Key Principles of Good App Architecture**

Let's suppose you’ve designed your app architecture per business and user requirements, but how do you verify that it’s composed correctly and your app will function as intended? A well-designed architecture should meet a set of specific conditions including:

* **Efficiency**: The application performs the tasks and functions under any condition and handles heavy loads effectively.
* **Portability**: Since frequent environmental (market or technical-specific) changes are natural in mobile applications, good architecture must allow adjustments without crashing the whole system.
* **Scalability**: Post-release updates and further development are a key part of the software development life cycle. Solid architecture results in faster changes and updates because it's developed in separate threads.
* **Testability**: If an app’s architecture is easily tested it reduces the number of errors and increases reliability.

**5.5 How to Choose the Right Architecture for Your Application**

Many app architecture patterns are similar, so it’s not always easy to select one pattern in particular. To choose an effective architecture for your app, the following steps are to be considered:

* Analyse the current state of your codebase to define any issues that need to be resolved or code that needs improvement.
* Evaluate various architecture patterns and try a few on for size before settling for one in particular.
* After the decision is made, examine how effectively your architecture solves the key problems.

**5.6Mobile application Design pattern**

**5.6.1What are Design Patterns?**

Design patterns are reusable solutions to common problems in software design. In a nutshell, design patterns are great for:

* Creating a common language between developers
* Fast-tracking developer on boarding
* Making the development process faster and simpler

Design pattern for mobile development are of three types which are:

1. **Creational –** describe how to create or instantiate objects, and the most used ones are Builder and Singleton
2. **Structural** – describe how objects are composed and combined to form larger structures (Model-View-Controller, Model-View-ViewModel, Decorator, Adapter, and Facade)
3. **Behavioural –** describe how objects communicate with each other (Delegation, Strategy, and Observer)
4. **Singleton Pattern Design Patterns**

The Singleton Pattern is one of the most used creational design patterns. It ensures that only one instance exists for a given class and that there’s global access to that point of instance. Having one global instance which has access to all others and uses all others is not really something that should be used.

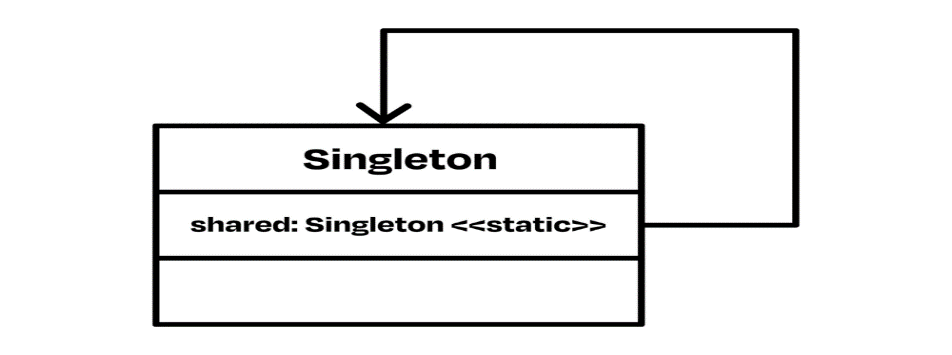


Figure 6

But, for example, Apple uses this approach in its ecosystem. Communication with the hardware part and of the smartphone goes through shared instances like:

**UserDefaults**.standard gives access to temporary memory,

**UIApplication**.shared provides access to other smartphone entries,

**UIScreen**.main for screen access,

**FileManager**.default gives file access.

Overall, the Singleton pattern is okay, but the downside is testing phase and is a considerable problem.

1. **The Decorator Pattern**

Then we have the Decorator Design pattern, which dynamically adds behaviours and responsibilities to an object without modifying its code, like extensions and delegations.

1. **[I]The Observer Pattern**

The Observer pattern you can use whenever you want to listen to changes made on another object. It consists of three objects:

**Observer**, which receives updates

**Subscriber,** which sends updates

**And the value,** which is the underlying object that’s changed

It’s mainly used when we want to receive changes on the object or a part of the system, and it’s helpful when the whole functionality paradigm is based on the observer pattern.



Figure 7

**[II] The Delegation Pattern**

When we talk about delegation patterns, we talk about mechanisms in which one object acts on behalf of or in coordination with another object. That object permits the second one to act as it or do some other action.



Figure 8

In the delegation design pattern we have two different objects:

-The delegate

-The object that needs the delegate e.gApple uses this approach in most of the UIKit classes.

**6**

**WAYS AND METHOD TO COLLECT AND ANALYSE USER REQUIREMENT FOR MOBILE APPLICATION (Requirement Engineering)**

The success of a software system depends on how well it fits the needs of its users and its environment. Software requirements comprise these needs, and requirements. Engineering (RE) is the process by which the requirements are determined. Successful RE involves understanding the needs of users, customers, and other stakeholders; understanding the contexts in which the to-be-developed software will be used; modeling, analyzing, negotiating, and documenting the stakeholders’ requirements; validating that the documented requirements match the negotiated requirements.

Here in Section 2, we are going to bring an overview of the essential difficulties in requirements engineering. In Section 3, we provide a summary of the state of the art of RE knowledge and research. In Section 4, we enumerate general research strategies for advancing the RE and we end with at section 5 with a conclusion and possible recommendations for this state of the art.

**6.1 Why is requirement engineering so hard? :**

In general, the research challenges faced by the requirements engineering community are distinct from those faced by the general software engineering community, because requirements reside primarily in the problem space whereas other software artifacts reside primarily in the solution space. That is, requirements descriptions, ideally, are written entirely in terms of the environment, describing how the environment is to be affected by the proposed system. In contrast, other software artifacts focus on the behavior of the proposed system, and are written in terms of internal software entities and properties. Stated another way, requirements engineering is about defining precisely the problem that the software is to solve (i.e., defining what the software system is to do), whereas other SE activities are about defining and refining a proposed software solution.

Several consequences follow from this distinction that cause RE to be inherently difficult such as;

• Requirements analysts start with ill-defined, and often conflicting, ideas of what the proposed system is to do, and must progress towards a single, detailed, technical specification of the system.

• Requirements artifacts have to be understood and usable by domain experts and other stakeholders, who may not be knowledgeable about computing. Thus, requirements notations and processes must maintain a delicate balance between producing descriptions that are intuitive and suitable for a non-computing audience and producing technical documents that are precise enough for downstream developers.

**6.2 State Of The Art for Research**

In this section, we summarize the state of the art of RE knowledge and research, as a baseline from which to explore future research directions. The research space is roughly decomposed into four categories of requirements problems (elicitation, modeling, analysis and validation, and requirements management). This decomposition is comparable to the top-level decomposition in Zave’s proposed scheme for classifying RE research [2]. The resulting matrix is shown in Table 1 below.

**A. Elicitation:**

Requirements elicitation comprises activities that enable the understanding of the goals, objectives, and motives for building a proposed software system. Elicitation also involves identifying the requirements that the resulting system must satisfy in order to achieve these goals.

• Techniques for identifying stakeholders help to ensure that everyone who may be affected by the software is consulted during elicitation.

• Analogical techniques, like norms, metaphors [3], and personas, help stakeholders to consider more deeply and be more precise about their requirements.

• Feedback techniques use models, model animations and prototypes [4], mockups, and storyboards to elicit positive and negative feedback on early representations of the proposed system.

**B. Modeling:**

In requirements modeling, a project’s requirements or specification is expressed in terms of one or more modeling notations. Modeling notations help to raise the level of abstraction in requirements descriptions by providing a vocabulary and structural rules that more closely match better than natural language does the entities, relationships, behavior, and constraints of the problem being modeled

* Modeling strategies provide guidelines for structuring models. For example, RE reference models [5] decompose requirements-related descriptions into the stakeholders’ requirements, the specification of the proposed system, and assumptions made about the system’s environment. In addition, they establish correctness criteria for verifying that the specified system will meet the requirements. In contrast, the viewpoints approach retains each stakeholder’s requirements in separate models, and the synthesis of a consistent global model that captures all of the stake holders’ concerns is delayed until conflicts can be re solved knowledgeably.
* Model patterns [6] encode common solutions to complex modeling problems. The RE community is also working on tools to help specifiers apply these patterns.
* Model transformations combine or manipulate existing models to derive new models. For example, model synthesis [7] and model composition techniques [8] integrate complementary sub-models into a composite model. In contrast, model merging techniques unify different views of the same problem.

**C. Analysis, validation, and verification:**

Requirements analysis assesses the quality of requirements models and documentation. Most of the research in this area focuses on new or improved techniques for detecting errors in models, where an “error” can be ambiguity [9], inconsistency, an unknown interaction among requirements [10], a possible obstacle to requirements satisfaction, or missing assumptions.

* Requirements validation ensures that models and documentation accurately express the stakeholders’ needs. Unlike the above analyses, which check that a software specification adheres to precise well-formedness criteria, validation is normally a subjective evaluation of the specification described or undocumented requirements. As such, validation usually requires stakeholders to be directly involved in reviewing the requirements artifacts [11].
* Research in this area focuses on improving the information provided to the stakeholder for feedback, including animations, simulation, and derived invariants.
* In cases where a formal description of the stakeholders’ requirements exists, obtained perhaps by validation, verification techniques can be used to prove that the software specification meets these requirements. Such proofs often take the form of checking that a model satisfies some constraint. For example, model checking [12] checks behavioral models against temporal-logic properties about execution traces; and model satisfiability [13] checks that there exist valid instantiations of constrained object models, and that operations on object models preserve invariants.

**D. Requirements management:**

* Requirements management is an umbrella activity that comprises a number of activities related to the management of the project or of the requirements engineering process. Such activities include traceability, impact analysis, cost estimation, risk management, as well as management of requirements variations.
* Research in this area focuses on easing management tasks and improving analysis techniques. For example, researchers have proposed a number of prioritization, visualization, and analysis techniques to help managers select an optimal combination of requirements to be implemented [14]; identify acceptable off-the-shelf solutions; and determine the maturity and stability of elicited requirements, so that the requirements most likely to change can be isolated

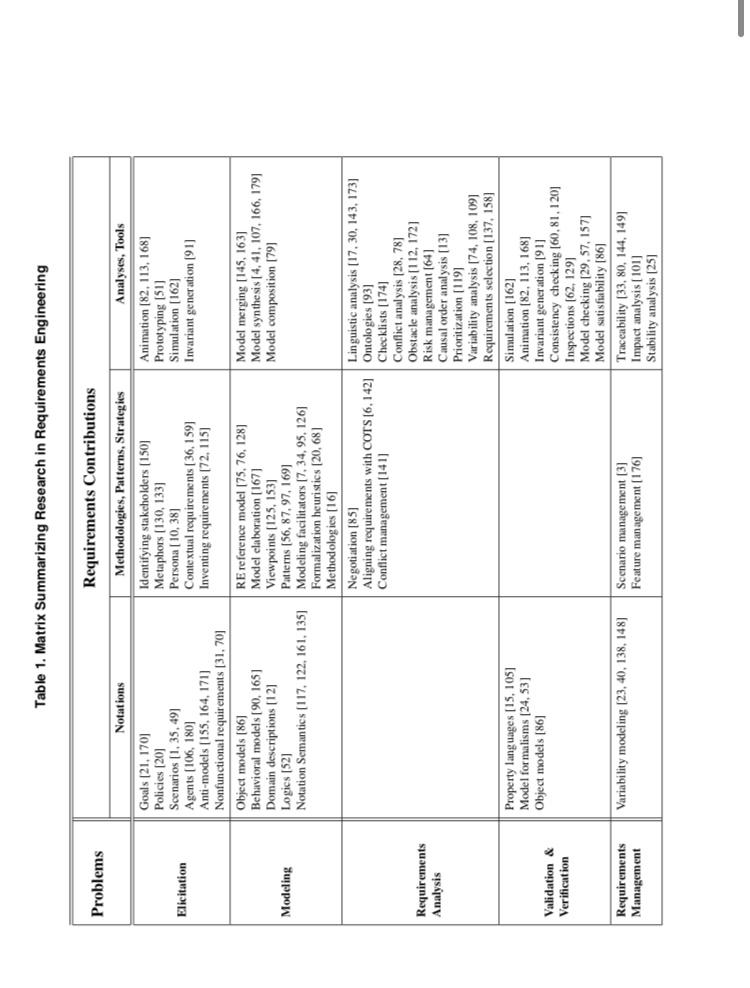


Figure 8

**6.3 Research Strategies:**

In this section, we will discuss ways of advancing the state of the art of RE research. We review several major strategies for conducting research, and look at how each have or might be applied to requirements-related research. Each strategy attempts to achieve a slightly different research objective, but all contribute in some way to advancing the state of the art, either by adding new knowledge or by improving the maturity of previous work which we will all see in Table 2 below.

**A. Paradigm Shift.**

A paradigm shift is a revolutionary solution that introduces radically new ideas or technologies to tackle a new or existing problem. A paradigm shift may be called for when researchers can no longer make progress on an important problem by extending or adapting existing solutions. Typically, there are two means by which a paradigm shift happens: push and pull.

* A paradigm shift is pushed onto a community when a new technology serendipitously makes major advances towards solving a problem for which it was not originally intended. A classic example of such a shift is the World Wide Web, which has significantly changed the way that society communicates and the way that services are delivered to consumers.
* Alternatively, a paradigm shift can be pulled when there is a real or a perceived crisis that cannot be solved by improving current ideas and techniques . For example, object-based design conventions (Encapsulation, Polymorphism, and Abstraction) were invented in response to serious concerns about how to structure programs and data in a way that promoted modularity. As they gained popularity, they evolved into object-oriented programming methodologies, were codified in new design methods, and were eventually supported by new programming language constructs.

**B. Leverage other disciplines:**

An RE researcher can leverage another discipline by identifying the analogous relationships between the two disciplines and then recasting promising knowledge, philosophies, principles, or practices from the other discipline into solutions that are appropriate for requirements problems. For example, software engineering, as a discipline, emerged when researchers and practitioners attempted to manage the “software crisis” by borrowing and adapting from the engineering profession several ideas about design principles, development processes, and discipline. Also the concept of genetic algorithms leverages ideas from biology, in that the algorithms “evolve” by using feedback from previous computations to improve future computations. Sutcliffe et al. [19] use genetic algorithms to select an optimal set of components that satisfy a given set of fitness criteria for reliability requirements.

**C. Leverage technology:**

Technological advances in computing and related fields can be combined and adapted to apply to problems in requirements engineering. In general, artificial intelligence, library science, information science, cognitive psychology, linguistics, statistics, and mathematics are all fertile areas for ideas and techniques that are suitable for such adaptation.

* For example, Overmeyer , use natural-language processing techniques to parse textual requirements descriptions and to generate corresponding semi-formal models, such as data-flow diagrams and communication diagrams.
* They and other researchersuse linguistic-analysis techniques to detect possible ambiguities and unintended inconsistencies in textual or use case requirements

**D. Domain-specific:**

A researcher can sometimes make better progress by narrowing the scope of a requirements problem and studying it in the context of a particular application domain. For example, there is a paradigm shift towards more domain-specific specification languages that provide native facilities for describing important entities and behaviors in that domain and provide macros for eliding recurrent requirements details. Along these lines, the International Telecommunication Union (ITU) has standardized a number of specification, design, and testing languages; design methodologies; and interface specifications – all of which support software aspects of telecommunication systems.

**E. Evaluation:**

Proposed RE technologies become theories, solutions, or practices through evaluation-based research that demonstrate effectiveness. Evaluation techniques include experience, collection and analysis of data, field studies, case studies, controlled experiments, and analytical reasoning.

* Evaluation criteria range from qualitative or statistical metrics, to effectiveness in solving real or realistic problems, to comparisons with competing technologies. A mature RE technology should be evaluated on real-world applications or in an industrial setting, to assess its scalability, practicality, and ease of use. In contrast, comparative studies are effective in evaluating the relative strengths and weaknesses of competing solutions to a problem. Notable comparative studies have investigated the criteria for choosing a specification language and the effectiveness of methods for inspecting requirements documents [24].
* because the contributions that requirements knowledge and artifacts make towards achieving downstream milestones would be more concrete.

**7 HOW TO ESTIMATE THE COST OF A MOBILE APPLICATION**

Mobile development comes along with cost which its Estimation can be tricky as it depends on several factors. Here's a breakdown of the key elements that influence mobile app development costs:

* **Factors Influencing Mobile App Development Costs**

– Complexity and Size of the App

– Platform Choice (iOS, Android, Both)

– User Interface and User Experience Design

– Backend Development and APIs

– Location and Structure of the Development Team

– Maintenance and Updates

* **Also, we discuss on tips and methods for and accurate cost Estimation**

– Fixed Price Model

– Time and Material Model

– Dedicated Team Model

* **Tips for Accurate Cost Estimation**

– Ensuring Clear Communication with the Development Team

– Taking into Account Unexpected Costs

– Regularly Revisiting and Revising the Budget

**7.1 Factors Influencing Mobile App Development Costs**

Determining the cost of mobile app development isn’t a one-size-fits-all process. It depends on a multitude of factors that vary from one project to another. Let’s delve into these elements to get a clearer picture.

* **Complexity and Size of the App**: A simple app with basic functionality will cost significantly less than a complex app with advanced features like data synchronization, user authentication, complex backend, etc. The size refers to the number of screens/pages in the app. More screens mean more design, development, and testing effort, thus higher cost.
* **Platform Choice (iOS, Android, Both)**: The chosen platform also affects the cost. Developing for one platform (iOS or Android) is less costly than creating a cross-platform app. However, depending on your target audience, you may need to consider developing for both platforms.
* **User Interface and User Experience Design**: A well-designed UI/UX is crucial for app success. Complex designs with custom graphics, animations, and transitions will require more effort, hence increased costs. Simpler designs, on the other hand, will be less expensive but might not stand out in the crowded app market.
* **Backend Development and APIs**: If your app needs to fetch data from a server or requires real-time synchronization (like in a chat app), a backend server will need to be developed, increasing costs. Similarly, if your app needs to interact with other apps or services using APIs, it will require additional development effort.
* **Location and Structure of the Development Team**: The location of your development team significantly influences the cost due to varying labor rates. Developers in North America tend to be more expensive than those in Asia. Additionally, a larger team with specialized roles (UI/UX designers, front-end and back-end developers, QA testers) will ensure a high-quality app but will also increase costs.
* **Maintenance and Updates:** Post-launch costs like server costs, emergency maintenance, app updates, and customer support are often overlooked but should be factored into the overall budget. Regular updates to keep up with platform changes (like new OS releases) and new feature additions are also a part of the total cost.

Understanding these factors is key to obtaining a realistic estimate of mobile app development costs and ensuring the financial success of your project.

**7.2 Methods for Estimating Mobile App Development CostsFixed Price Model**:

Under this model, itCraft estimates the total cost of the project based on the defined requirements and specifications before the development starts. This method is most suitable for small projects with clear, unchanging requirements. The main advantage is that the client knows the exact cost upfront, helping with budget planning. However, it lacks flexibility for changes or additions once the project is underway.

* **Time and Material Model:** This model is based on the actual time and resources utilized on the project. Clients are billed according to the hours worked by the development team. This approach is suitable for projects where requirements are expected to evolve or are not fully defined at the beginning. It offers high flexibility as changes can be made during the development process, but the final cost might exceed the initial estimates if the project scope expands or unexpected challenges occur.
* **Dedicated Team Model**: In this model, itCraft provides a dedicated team of professionals who work exclusively on your project. The cost is based on the team size and composition, and the time they spend on the project. This model is ideal for long-term projects where requirements might change over time. It gives you greater control over the development process and ensures that the team is fully invested in your project.

**7.3 Tips for Accurate Cost Estimation**

* **Ensuring Clear Communication with the Development Team**

Open and continuous communication with your development team is critical for accurate cost estimation. As a client, you should be clear and precise about your requirements, expectations, and budget

* **Taking into Account Unexpected Costs**

No matter how well you plan, there will always be some unforeseen expenses during the app development process. These could be due to sudden changes in requirements, extra time spent on bug fixing, or additional features requested by users after launch. Therefore, it’s wise to set aside a contingency fund as part of your budget. This prepares you for unexpected costs and ensures that the development process isn’t hampered due to budget constraints.

* **Regularly Revisiting and Revising the Budget**

Mobile app development is a dynamic process, and costs can fluctuate based on a variety of factors. For example, additional features may be needed, or development could take longer than expected. It’s important to revisit and revise your budget periodically, keeping track of the actual spend against the estimated cost. This will help identify if the project is on track financially, and if not, corrective actions can be taken early.

1. **CONCLUSION**

This report has provided a comprehensive overview of mobile application development paradigm which is not a one time but a continuous process that requires clear requirement gathering, communication with an accurate cost estimation. Following these tips ensure a smooth development process. And to remember, an investment in a mobile app is an investment in your business’s future. By prioritizing and planning correctly, you can ensure this investment gives you a high return in the long run.

**9 RECOMMENDATIONS**

. These are some recommendations that the RESERCH AND MOBILE APP DEVELOPMENT community could take immediate action on, to start improving the maturity of both current and applications than would be developed in the future;

* Researchers should work with practitioners. Such partnerships can help to ensure that researchers have a thorough understanding of mobile application strategies framework and policies
* They need to answer questions of

1) What do you want your app to do?

2) What problem is it going to solve?

3) How will you market your app?

By so doing chosen a design pattern e.g. **singleton** pattern and with a rich architecture like that of **Model-view-viewmodel architecture** would reduce time and cost for the overall development process.

* Developers need to Sketch” Brainstorming Ideas” in several domains in the field of the mobile app development process. researching on programming languages with rich potential, with and easy to use scheme could be beneficial
* App developers and researchers must lay focus on targeting their audiences — teens, children, teachers, wanderers, gamers etc. Weather the application would be for online or off line consumption? Determine them right away! Before engaging in mobile app development and processes.

1. **REFERENCES**

[1] K. Tracy, Mobile Application Development Experiences on Apple’s iOS and Android OS,

IEEE Potentials, vol. 31, pp. 30–34, 2012.

[2] Kayun Chantarasathaporn and Chonawat Srisa-an. Energy conscious factory method design pattern for mobile devices with c# and intermediate language. In Proceedingsof the 3rd International Conference on Mobile Technology, Applications &#38;

Systems, Mobility ’06, New York, NY, USA, 2006. ACM.

[3] D. Berry and E. Kamsties. Ambiguity in Require- ments Specification. Perspectives on Software Require- ments, chapter 2. Kluwer Academic Publishers, 2004.

[4] D. Jackson. Software Abstractions: Logic, Language, and Analysis. MIT Press, 2006.

[5] J. Hall and L. Rapanotti. A reference model for require- ments engineering. In Proc. of the IEEE Int. Req. Eng. Conf. (RE), pages 181–187, 2003.