http://www.moderntechno.de/index.php/meit/article/view/meit29-01-056
DOI: 10.30890/2567-5273.2023-29-01-056

QUALITY ATTRIBUTES AND ARCHITECTURAL PATTERNS OF MODERN MOBILE APPS

Bilohub Dmytro
2nd-year master's degree student, Software Engineering
Skrypchenko Mykyta
2nd-year master's degree student, Software Engineering
Tytenko Sergiy
Ph.D., Associate Professor
ORCID: 0000-0002-7548-9053
American University Kyiv, Ukraine, Kyiv, Poshtova Pl. 3, 02000

***Abstract**. When embarking on the development of a mobile application, the initial step is the selection of the appropriate architectural framework, which should be based on the specific domain, intended functionality, and the identified quality attributes. This article delves into the examination of established architectural paradigms utilized in mobile application development, alongside an exploration of the quality attributes that should be taken into account during the architectural selection process. The aim here is to unearth prevalent mobile architecture patterns and provide insights into methods for measuring and comparing these architectures against one another.*

***Keywords:** software architecture, mobile development, quality attributes, architectural patterns.*

Introduction

Mobile applications have witnessed an unprecedented surge in popularity and ubiquity over the past decade. This phenomenon can be attributed to several factors, including increasing smartphone adoption, changing consumer behavior, and the normalization of mobile-first approaches in businesses. As pointed out in [1], the penetration rate of smartphones reached 78.05 percent globally in 2020, and this figure is projected to rise to almost 87 percent in the United States by 2025. With this surge, mobile applications have become the primary gateway to the online world for a significant portion of the global population.

In 2010, only 27 percent of mobile users in the United States owned smartphones, a stark contrast to the anticipated 87 percent in 2025 [1]. This transformative shift has allowed not only enterprises but also individual developers and teams to create and distribute their applications to a worldwide audience. However, this diverse landscape comes with various challenges, ranging from differences in team sizes, budget constraints, and time-to-market pressures to distinct marketing strategies and business models.

Not only enterprises but individual developers and teams are allowed to create their applications and distribute them to a worldwide audience. The organizations vary in team sizes, budget constraints, time-to-market and marketing strategies, and business models.

Mobile applications have become integral to our daily lives, facilitating a wide range of activities, from online shopping to managing finances and accessing government services. Consequently, mobile application developers bear a considerable responsibility in ensuring that their products meet high standards of usability, availability, and reliability. This responsibility extends to both individual developers and large enterprises.

Amidst this dynamic landscape, the software architecture of a mobile application emerges as a pivotal factor. Software architecture encompasses the fundamental design decisions that dictate how an application will function and evolve. In the context of mobile applications, where agility, scalability, and user experience are paramount, the choice of software architecture becomes even more crucial.

High competition levels and therefore high levels of expectations from users make mobile application vendors careful in technical decisions and practices. The first thing every vendor should be aware of is the software architecture of its product.) – In the realm of mobile applications, a prescient perspective has been validated, primarily attributed to the rapid pace of evolution within this domain. Each vendor must possess a profound understanding of the software architecture underpinning their product as a foundational consideration.

Software Architecture and Quality Attributes for Mobile Applications Development

How can you ensure that the software architecture you've selected is the most suitable for your needs? It is not an easy question, and there is no one right answer or a silver bullet. As stated in the work [2], software architecture is like a bet, a wager, wouldn't it be nice to know the outcome in advance? To know it, we need to step out and as we already did for the piece of code and then for software — we need to establish the rules, and measurements for a clear understanding of which architecture could fit better in certain conditions. However, the topic has already been well researched, we found plenty of space to fill towards the architecture of mobile applications.

Project Managers seek to achieve business requirements, Software Architects seek to manage the architecture according to these requirements. It can’t be managed what is not measured. But what can we measure in software architecture? While it was identified [4] the main quality attributes of the software overall, we would like to stop on those that are crucial specifically for mobile application development.

Performance. Performance pertains to the system's reactivity, which encompasses the time needed to react to stimuli (events) or the number of events processed during a specific period [4]. Performance holds significant importance in mobile software architecture due to the perpetual resource constraints of mobile devices, including limitations in terms of microprocessor power, storage capacity, and battery life .

Reliability. Reliability denotes the system's capacity to sustain continuous operation throughout its operational lifespan, often assessed through the metric known as mean time to failure [4]. For mobile systems we should consider that the network could be unstable, and some operations could fail due to the aforementioned constraints.

Availability. Availability refers to the percentage of time the system remains operational [4]. Availability is a critical consideration in mobile application architecture. Ensuring the availability of a mobile application means that it should be accessible and operational for users whenever they need it. Mobile applications can face challenges related to network connectivity, server availability, and other factors that can affect their accessibility. To provide a positive user experience, mobile app architects need to design for availability, which may include strategies like offline capabilities, redundant servers, and efficient error handling to ensure the app remains usable even under less-than-ideal network conditions.

Security. Security involves assessing the system's capacity to withstand unauthorized access attempts and service disruptions, all while maintaining its functionality for authorized users [4]. Security of Mobile apps involves implementing robust measures to protect user data and the application itself, ensuring that it remains resilient in the face of potential security risks.

Modifiability. Modifiability refers to the system's capacity for swift and cost-effective modifications, which is one of the key attributes for a large team, which is working on a mobile application.

Portability. Portability refers to a system's adaptability to function across diverse computing environments, which can encompass hardware, software, or a blend of both [4]. For a mobile architecture, it could address new versions of OSs or the device's screen size.

As stated in the work [2], quality attributes form the basis for architectural evaluation, but simply naming the attributes by themselves is not a sufficient basis on which to judge an architecture for suitability. In a perfect world, the quality requirements for a system would be completely and unambiguously specified in a requirements document.

Modern Mobile Applications Architectural Styles

Over the years of mobile app development, the projects have rapidly become more and more complex and harder to maintain. Companies behind major platforms - Google and Apple have sought to develop guidelines for native app architectures. At the same time, developers have adapted existing architectural patterns for mobile platforms and continued to evolve them to adjust to growing complexity. In this section we will briefly review some of the most popular patterns: MVC, MVP, MVVM, and Clean architecture-based patterns.

Model-View-Controller (MVC)

Model-View-Controller (MVC) architectural pattern was first introduced by Trygve Reenskaug while working on Smalltalk based system [3]. MVC architecture operates with three entities: model, view and controller. A model object encapsulates the data and implements the logic for manipulating the data. It can be as simple as storing a string, or more complex, like storing an object with multiple value fields or retrieving data remotely. A view object is responsible for the user interface. View object receives updates of model's state and updates the user interface accordingly. Controller object receives user input events from the view and changes the state of the model [5].

Over the years, the MVC architecture gained a lot of popularity among developers thanks to its clear separation of concerns [6].

MVC implementation in iOS

When Apple first published its iPhone SDK, MVC was the recommended architecture for native apps. However, the implementation Apple has proposed was different from the original pattern [7].

View doesn't receive updates of the model directly. Instead, the controller receives updates of the model and then updates the view accordingly. Ideally, View and Model should not even be aware of each other and should not be coupled in any way.

This form has a few advantages, mainly increased maintainability. Decoupled models and views should make it easier to introduce changes to one of these components without changing the other. Additionally different teams may be able to work on business logic and user interface concurrently which can increase overall productivity and time to market.

MVC and its variants [8] have also become very popular with Android developers [9] due to its excellent separation of concerns and increased maintainability.

Model-View-Presenter (MVP)

Model-View-Presenter (MVP) pattern was introduced in 1990s and was one of the first alternatives to MVC [10]. MVP has a lot in common with MVC. Similarly, MVP also has 3 entities: Model, View and Presenter.

In MVP, the Model is again responsible for data access and manipulation. The view is dedicated to user interface and presentation logic. Finally, MVP introduces a new concept: Presenter. The presenter acts as an intermediary between the Model and the View. It responds to the user input from the view and updates the Model. At the same time, the Presenter listens to changes in the Model's state and updates the view if necessary [11].

The View in MVP is completely separated from the Model, which is similar to the Apple's version of MVC we mentioned earlier. While MVP is a little more complex than MVC, it offers a better separation of concerns, which makes business logic more testable [11].

Model-View-ViewModel (MVVM)

Model-View-ViewModel (MVVM) architecture was introduced by Microsoft in the early 2000s and was used in Silverlight and WPF [12]. It became prevalent in native app development for both Android and iOS.

MVVM operates with three objects: Model, View and ViewModel.Similarly to MVC and MVP, the Model is responsible for data access, data persistence, and communication. The View is dedicated to the user interface and should not contain any logic. The new object ViewModel is the core of the MVVM pattern, and its place is between Model and View. The view is bound to ViewModel via data binding. The data binding updates the View automatically when ViewModel's state changes. A View can have multiple references to ViewModel. However the ViewModel should not have any dependencies on View. This is a big difference compared to MVC or MVP, where presenter or controller will set view in code [9]. This way MVVM offers even greater separation of concerns and greater testability of ViewModel [11].

Clean Architecture based architectural patterns

Clean Architecture is a layered architectural pattern proposed by Robert C. Martin [13]. This architecture provides an improved separation of concerns by using four concentric layers: Entities, Use Cases, Interface Adapters, Framework and

Drivers.

The central idea behind Clean Architecture is that the inner layers (Entities and Use Cases) are independent of the outer layers. Business logic and core use cases are not tightly coupled to specific UI frameworks, databases, or external services. The dependency flow moves inward, from the outer layers to the inner layers. The inner layers should have no direct dependencies on the outer layers [13]. This separation enables easier testing, maintainability, and adaptability [14].

Naturally, Clean Architecture derived patterns became extremely popular with mobile developers. On iOS the most popular patterns are VIPER [15] and VIP [16]. Android and cross-platform frameworks like Flutter have their own examples of Clean Architecture implementation [17].

Conclusions and future work

In this work, we discussed what is Software Architecture and Quality Attributes for Mobile applications and what are the main styles used in Mobile application development. We discussed the history of architectural styles and patterns and their implications in the modern world.

In the [18] it was described the variability inherent in the design of the software architecture and how that variability argues for the use of measures that are tailored to the context of the specific organization. For instance, different roles seek to have different points of view and therefore different metrics, specifically, there was elaborated on Software Architect and Project Manager roles.

That's also true for different domains – there is no architecture of size-fits-all, no silver bullet, we should carefully integrate our requirements into Quality Attributes and therefore pick the right architecture for the solution.

A more in-depth examination underscores the significance of a comprehensive assessment of mobile software architectures with a focus on their thorough implementation to ensure the robustness of these attributes and more.

Such examination should first of all rely on practical comparison of the different architecture codebases. Furthermore, we have posed an essential research question: Is there a suitable method for selecting the right architecture based on the specific requirements of a given project? As the field of Mobile Application Development continues to expand, there is a growing need for a more in-depth exploration of this topic, particularly concerning the criteria used to evaluate the quality attributes of Mobile Software Architectures.

Literature:

1. Laricchia, F. (2023, September 28). *Global smartphone penetration 2016-2022*. Statista. https://www.statista.com/statistics/203734/global-smartphone-penetration-per-capita-since-2005/

2. Clements, P., Kazman, R., & Mark. Klein. *Evaluating Software Architectures: Methods and Case Studies, Addison Wesley., Dec 6, 2001.*

3. Trygve Reenskaug. *The Model-View-Controller (MVC). Its Past and Present* Java Zone, Oslo 18–19 September 2003

4. Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice*. Addison-Wesley Professional.

5. Fowler, M., Rice, D., Foemmel, M., Hieatt, E., Mee, R.,, Stafford, R. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.

6. Campos, E., Kulesza, U., Coelho, R., Bonifácio, R., & Mariano, L. (2015, April). *Unveiling the architecture and design of android applications*. In *Proceedings of the 17th international conference on enterprise information systems* (Vol. 2, pp. 201-211).

7. Apple Inc. Concepts in Objective-C Programming, Model-View-Controller [Online]. 2012. Available: https://developer.apple.com/library/content/documentation/General/Conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html

8. Sokolova, Karina & Lemercier, Marc. (2014). *Towards High Quality Mobile Applications: Android Passive MVC Architecture*. International Journal On Advances in Software 1942-2628. 7. 123 - 138.

9. Lou, T. (2016). *A comparison of Android Native App Architecture MVC , MVP and MVVM*.

10. Potel, Mike. (1996). MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java", Taligent Inc.

11. García, R. F. (2023). *IOS architecture patterns MVC, MVP, MVVM, Viper, and VIP in swift*. Apress. ISBN: 978-1-4842-9069-9

12. J. Grossman, *Introduction to Model/View/ViewModel pattern for building WPF apps*, Microsoft, 8 October 2005. [Online]. Available: https://blogs.msdn.microsoft.com/johngossman/2005/10/08/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps/

13. Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Boston, MA: Prentice Hall. ISBN: 978-0-13-449416-6

14. D. Bui (2017). *Reactive Programming and Clean Architecture in Android Development*

15. J. Gilbert and C. Stoll. *Architecting ios apps with viper*, objc, vol. 13, 2014. [Online]. Available: https://www.objc.io/issues/13-architecture/viper/

16. R. Law, *The Clean Swift Handbook*, 2019. [Online]. Available: https://clean-swift.com/handbook/

17. S. Boukhary and E. Colmenares, *A Clean Approach to Flutter Development through the Flutter Clean Architecture Package,* 2019 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 2019, pp. 1115-1120, doi: 10.1109/CSCI49370.2019.00211.

18. Chastek, Gary J.; Ferguson, Robert W. (2018). Toward Measures for Software Architectures. Carnegie Mellon University. Report. https://doi.org/10.1184/R1/6585371.v1

Article sent: 20.10.2023