UDC 004.2

ANALYSIS OF APPROACHES TO IMPLEMENTATION OF A MOBILE APPLICATION FOR ANDROID OS

ALIAKSANDR MOROZOV, RICHARD BOGUSH
Polotsk State University, Belarus

In this article we are going to review the architecture and general approaches to implementing a mobile TV- and video-viewer application for the Android platform. We are going to list the major functions of the software required to develop the mobile application.

In this article we are going to review the architectural approach that may be used for developing a mobile application intended for the Android operating system. We describe the advantages and the drawbacks of that approach, giving explanations and analyzing the chosen architecture [1].

The project development process involves adding a great number of code lines. If we work with a single development pattern the code become hard to work with, as the developer has to check constantly where a certain function begins and keep in mind the constant code replications. [2] To make that work easier, various application architectures were developed.

There are quite a few approaches to developing complex systems with robust architectures. Despite some insignificant differences, those approaches have a lot in common. Of course, all of them establish the methods to divide the application into separate modules. As a minimum, each system has modules containing the application's business rules and data display modules. In the end, each approach may be used to develop a system that meets the following requirements:

– Architecture must be independent of various frameworks;
– The system must be tested;
– The application must be independent. [3]

Independent of Frameworks. The architecture does not depend on the existence of some library of feature laden software. This allows you to use such frameworks as tools, rather than having to cram your system into their limited constraints.

Testable. The business rules can be tested without the UI, Database, Web Server, or any other external element.

Independent of UI. The UI can change easily, without changing the rest of the system. A Web UI could be replaced with a console UI, for example, without changing the business rules.

Independent of Database. You can swap out Oracle or SQL Server, for Mongo, BigTable, CouchDB, or something else. Your business rules are not bound to the database.

Independent of any external agency. In fact, your business rules simply don't know anything at all about the outside world. [4]

The schematic approach to organizing a mobile application code is presented on Figure 1.

The Dependency Rule. The concentric circles represent different areas of software. In general, the further in you go, the higher level the software becomes. The outer circles are mechanisms. The inner circles are policies.

The overriding rule that makes this architecture work is The Dependency Rule. This rule says that source code dependencies can only point inwards. Nothing in an inner circle can know anything at all about something in an outer circle. In particular, the name of something declared in an outer circle must not be mentioned by the code in an inner circle. That includes, functions, classes. variables, or any other named software entity.

By the same token, data formats used in an outer circle should not be used by an inner circle, especially if those formats are generating by a framework in an outer circle. We don't want anything in an outer circle to impact the inner circles.

Entities encapsulate Enterprise wide business rules. An entity can be an object with methods, or it can be a set of data structures and functions. It doesn't matter as long as the entities could be used by many different applications in the enterprise. If you don't have an enterprise, and just write a single application, then these entities are the business objects of the application. They encapsulate the most general and high-level rules. They are the least likely to change when something external changes. For example, you would not expect these objects to be affected by a change to page navigation, or security. No operational change to any particular application should affect the entity layer.

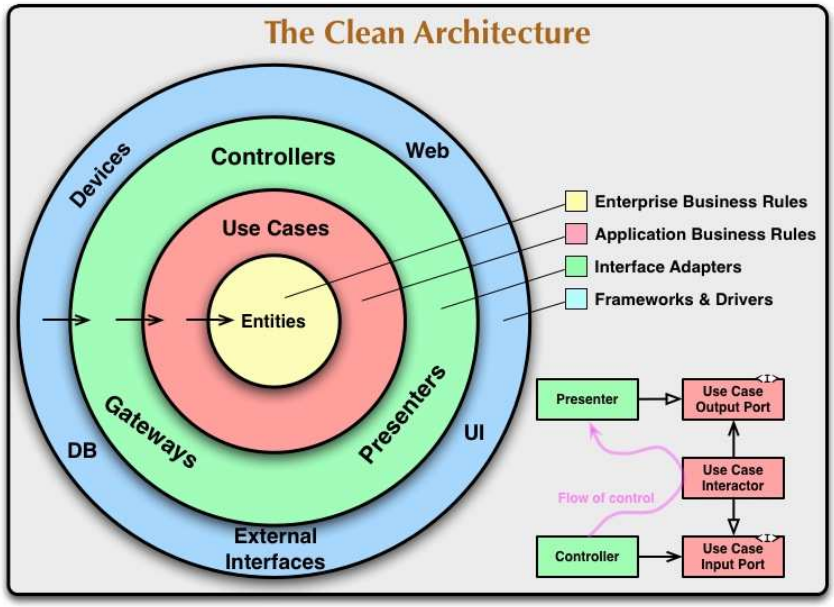

Figure 1. – Clean Architecture Diagram

Use Cases. The software in this layer contains application specific business rules. It encapsulates and implements all of the use cases of the system. These use cases orchestrate the flow of data to and from the entities, and direct those entities to use their enterprise wide business rules to achieve the goals of the use case.

We do not expect changes in this layer to affect the entities. We also do not expect this layer to be affected by changes to externalities such as the database, the UI, or any of the common frameworks. This layer is isolated from such concerns.

We do, however, expect that changes to the operation of the application will affect the use-cases and therefore the software in this layer. If the details of a use-case change, then some code in this layer will certainly be affected.

Interface Adapters. The software in this layer is a set of adapters that convert data from the format most convenient for the use cases and entities, to the format most convenient for some external agency such as the Database or the Web. It is this layer, for example, that will wholly contain the MVC architecture of a GUI. The Presenters, Views, and Controllers all belong in here. The models are likely just data structures that are passed from the controllers to the use cases, and then back from the use cases to the presenters and views.

Similarly, data is converted, in this layer, from the form most convenient for entities and use cases, into the form most convenient for whatever persistence framework is being used. i.e.

The Database. No code inward of this circle should know anything at all about the database. If the database is a SQL database, then all the SQL should be restricted to this layer, and in particular to the parts of this layer that have to do with the database.

Also in this layer is any other adapter necessary to convert data from some external form, such as an external service, to the internal form used by the use cases and entities.

Frameworks and Drivers. The outermost layer is generally composed of frameworks and tools such as the Database, the Web Framework, etc. Generally, you don't write much code in this layer other than glue code that communicates to the next circle inwards.

This layer is where all the details go. The Web is a detail. The database is a detail. We keep these things on the outside where they can do little harm. [5]

Application architecture is one of its most important characteristics, defined in the very beginning of an application's development. Changing architecture is a complex and long operation involving the already existing code which may lead to many new bugs negatively impacting the application's functions. Therefore, one has to be careful when choosing architecture back at the application design stage.

In this article we reviewed one of the approaches to the mobile application design, namely the Clean Architecture approach, defining the layers separating the mobile application's components, which, in turn, improves the code's testability. The modules become independent from one another, thus giving the developers

an opportunity for their independent development and reuse, as well as simplifying introductions of new features to the project.

REFERENCES

1. Анализ современных средств для разработки мобильных приложений под ОС Android [Электронный ресурс]. – Режим доступа: https://cyberleninka.ru/article/n/analiz-sovremennyh-sredstv-dlya-razrabotki-mobilnyh-prilozheniy-pod-os-android/. – Дата доступа: 02.02.2019.
2. Паттерны проектирования встроенных систем [Электронный ресурс].– Режим доступа: https://cyberleninka.ru/article/n/patterny-proektirovaniya-vstroennyh-sistem/. – Дата доступа: 05.02.2019.
3. Clean architecture for Android with Kotlin: a pragmatic approach for starters [Electronic resource]. – Mode of access: https://antonioleiva.com/clean-architecture-android/. Date of access: 10.02.2019.
4. A Guided Tour inside a clean architecture code base [Electronic resource]. – Mode of access: https://proandroiddev.com/a-guided-tour-inside-a-clean-architecture-code-base-48bb5cc9fc97/. – Date of access: 11.02.2019.
5. Basic concepts of software architecture patterns in Android [Electronic resource]. – Mode of access: https://android.jlelse.eu/basic-concepts-of-software-architecture-patterns-in-android-c76e53f46cba/. – Date of access: 16.02.2019.