Rationale

With the new incoming requirements, we decided to keep the **MVP design pattern** as it has a clear separation and brings more decoupling between domain layer and user interface[1], which is very suitable for testing, maintaining, and extending new functionalities without much modification on existing classes. However, it has higher complexity which takes more learning time for maintainer and new developer to be familiar with the structure and more compiling time as more classes created by the new requirements.

To support different kinds of Monitors for the new requirement, we use **Factory Method pattern** to create different monitor by reusing existing Cholesterol request structure without breaking existing monitor code, it can avoid tight coupling between containers(creator) and the concrete monitors(product), but it may become more complicated as we need to introduce new subclass for the new monitor.[2] Here we also **refactor** the MonitorContainer and its subclass to apply this pattern by using **Move Method** to move the specific request methods to the concrete Container, **Pull Up Field** to put the reusable attributes into abstract Container. Actually there are a lot of places we've used refactor, such as put getter and setter for most of data holder classes, make attributes private to improve security(**Self-Encapsulate**) etc.

In order to avoid cyclic dependencies, we use **The Acyclic Dependencies Principle** in the Contract Interfaces, so that the Presenters and the Views have agents to communicate with each other. Each Contract also apply **The Single Responsibility Principle** as it only responsible for its corresponding Presenter and View, which means that we can quickly locate the bugs or change requirements in the responsible classes. With the use of **The Interface Segregation Principle**, both Contracts and Callbacks have unique methods for the classes that depend on them, only depend on the methods that they use. These principles both support good extension because of modular design

The Model package is robust and stable as it only request the data from server and not depends on other package in the system, so **The Stable Dependencies Principle** is applied between it and Presenter package. we want Presenters to be unstable as new requirements change we will be able to modify.

Inside the Model package, we use The **Open Closed Principle** to add new classes in ServerSource for the new query instead of modifying the existing classes, which has a good extensibility.

Reference:

[1] Daoudi, A., Moha, N., ElBoussaidi, G., & Kpodjedo, S. (2019). An exploratory study of MVC-based architectural patterns in android apps. Proceedings of the ACM Symposium on Applied Computing, 147772, 1711–1720. https://doi.org/10.1145/3297280.3297447

[2] Factory Method. (n.d.). Retrieved from https://refactoring.guru/design-patterns/factory-method