FIT3077 Assignment 2 Design Rationale

Referring to the system class diagram, the use of abstract classes in the system is to allow for extensibility and flexibility within the system as they are points of hinges for new functionalities to be easily implemented. For example, if there is a new type of user, we can easily extend a new class from User to achieve that. This is workable because the dependencies and association of the classes in the system are mostly on their respective abstractions rather than any concrete classes. For example, the AbstractController would be dependent on the AbstractView and AbstractModel rather than any of the classes' concrete implementation. User, Monitor and PatientInfo classes were designed using Abstraction as well. This promotes the extensibility of the system and allows new functionality to be implemented. Hence, it can be said that the **Open-Closed Principle (OCP)** is practiced. Furthermore, the reason why such dependency on the abstraction is possible is due to the system design in following **Liskov Substitution Principle (LSP)** and **Dependency Inversion Principle (DIP)**, whereby the subtypes of the abstract classes conform strictly to the superclasses. As mentioned earlier, the system mainly operates by depending on the abstractions rather than concrete implementations, this can be seen in the AbstractController and AbstractModel classes where they rely on the abstract classes to function and set relevant properties rather than relying on concrete classes.
Lastly, the use of Model, View, Controllers packages are meant to represent the conceptual idea of splitting the GUI into 3 distinct parts (Model, View and Controllers). This adheres to the **Model-View-Controller design pattern**. The reason why we separated the GUI is to avoid the cyclic dependency to be present if the Model and View were to depend on each other. Refactoring the GUI to include the Controller type class ensures there would not be cyclic dependency and allow for ease of testing for different GUI components. However, the implementation of the Controller classes were slightly modified due to the Model used for patients. Patient Model here is a more atomised form compared to the commonly used model data. Therefore, the benefit of having the model controlled by the Controller class is largely negated.

The introduction of the **Observer design pattern** allows UI components to display the same Model data simultaneously. This allows for future extensibility of functionality where we might have to display the same Model for different UI components. For example, having to create two separate windows with the same data but highlighting different patient data (blood pressure and cholesterol levels). However, as mentioned previously, even when the PatientModel is updated, only the controller classes are updated to communicate to the views the update to the list of patients to be displayed in the tables.

Server and Calculation classes are in their own respective packages because not all packages will need to depend on these two classes. Since it will not be reused together, it should not be grouped together with other classes. Hence, it can be said that the **Common Reuse Principle** is practiced. This decreases unnecessary dependencies between packages/classes as well as promoting higher package cohesion. **Single Responsibility Principle** is also practised here because Server and Calculation class each has its own specific responsibility for their classes. This result is higher encapsulation and lower coupling. Model, View, Controller and Observer are also grouped together in their own packages to follow the **CRP** and **Common Closure Principles (CCP)**. This ensures that the packages are reusable and the right classes are tightly coupled together in the same package.

To conclude, the reasoning behind why such design principles and patterns are practiced is to set up vital points of hinges as well as allow for reusability of code that can be extended for future addition of functionality without having to change much of the current source code.

References:

Assignment 2 Source Code
Assignment 2 Class Diagram
FIT3077 MVC Lecture
FIT3077 Design Patterns 1 Lecture (Observer)
FIT3077 OOP1 (Design principles)
FIT3077 OOP2 (CRP and CCP)
Observer Pattern reference