# COMP 3023 Assignment 2

# Calculate the patient alert levels

**A diagram of a computer

Description automatically generated**

Figure 1: Strategy Design Pattern

**Design Pattern:**  Strategy Design Pattern

The system must be able to utilise the correct algorithm to calculate a patient’s alert level depending on their primary disease. To address this, I chose to utilise the strategy design pattern, which would make it so that each disease alert level algorithm were interchangeable

This design is represented in Figure 1. The AlertCalcStrategy is the common interface that all the algorithms inherit from. This class is used by the AlertCalcContext, which sets and executes the chosen algorithm.

**How it works:**

1. A patient is selected to have new vitals declared for by the user
2. The PatientManagementSystem calls the addVitals() method for the selected patient, passing in a the new vitals class and a boolean set to true
3. The addVitals() method adds the inputted vitals to the patient
4. The status of the inputted Boolean is then checked, if true, addVitals() will set a new alert for the patient based on the inputted vitals by calling the function
5. The alertCalc context and the specific diagnosis’s alert calculation classes are defined
6. The patients primaryDiagnosis is checked, addVitals() calls the setStrategy() function from the context, passing in a reference to the diagnosis strategy
7. After the context is set, addVitals() calls the executeStrategy() function from the context, passing in a reference to itself (a reference to the patient)
8. AlertCalcContext calls the calculateAlert() function from the strategy, which is inherited from the AlertCalcStrategy.
9. The specific strategy checks the vitals, and then calls setAlertLevel() from the inputted patient, passing in the chosen AlertLevel.

**Git commits:**

* I first implemented the strategy in commit ccb3514
* I then implemented the context for the strategy in commit 12c58f7
* I implemented the calculation for the strategies in commit d34278d
* I had a working finished FR1 in commit d653330
* Other commits: f0f93a3, e7865cb, 4243152

# Calculate the Alert level for all diseases a patient has

A diagram of a data flow

Description automatically generated

*Figure 2: composite design pattern*

**Design Pattern:** Composite Design Pattern

The system must calculate the AlertLevel for every disease that a patient has when new vitals are set for the patient, It is important that the system then sets the highest AlertLevel for the patient from all of the diseases. To address this requirement, I applied the composite design pattern, which wrapped all of the disease calculation algorithms in a common interface, making it simple and efficient to access and retrieve the alert levels for each individual disease.

The design pattern is represented in figure 2. The AlertCalcStrategy is the primary interface that all of the disease’s inherit from, this promotes easy expansion. The AlertCalcContext is the main interface from which the patient interfaces with.

**How it works:**

1. A patient is selected to have new vitals declared for by the user
2. The PatientManagementSystem calls the addVitals() method for the selected patient, passing in a the new vitals class and a boolean set to true
3. The addVitals() method adds the inputted vitals to the patient
4. The status of the inputted Boolean is then checked, if true, addVitals() will set a new alert for the patient based on the inputted vitals by calling the function
5. The alertCalc context and the specific diagnosis’s alert calculation classes are defined
6. The patients diagnoses are checked, addVitals() calls the setStrategy() function, setting the strategies to the relevant patient diagnoses.
7. addVitals() then calls executeStrategy()
8. executeStrategy() goes through every strategy and calls the calculateAlert()
9. calculateAlert() returns the alert level to the context
10. the context then calls checkAlertLevel() which checks using the int values of the enum Alert class, if the new alert level is higher than the already set one
11. if the new alert level is higher, checkAlertLevel() sets currentAlertLevel to the new alert level
12. once all of the strategies have been called, the context returns the currentAlertLevel
13. AddVitals() then calls setAlertLevel() with an input value of the returned alert level

# Adapter Pattern

The adapter pattern was chosen so that existing patient files and new patient files can be loaded. Allowing possible incompatible interfaces to work together.

# Observer Pattern

The observer pattern was chosen, as it makes sense for an alert system to be flexible and scalable. This makes it so that new stakeholders can be added to the system easily.