***Level of the chosen requirements***

# **Requirements considered**

* **Requirement 22:** “*The system should be able to plan and plot a flightpath for delivering orders in 60 seconds or less”*.
* **Requirement 14:** *“The system should be able to retrieve data about different components (such as location of the University’s Central Area, locations of No-Fly-Zones, restaurant locations and order information) from a given REST server”*.
* **Requirement 20:** “*The system should output user friendly messages in case a validation check for the command line input fails”*.
* **Requirement 18:** “*The system should validate each order for the given date, and, in case some of the order information is not valid, label it with the appropriate order outcome depending on which information was invalid”*.
* **Requirement 13:** *“The system should ensure that the drone never enters a no-fly-zone”*.

# **Level of each requirement**

* **Requirement 22:** *system level*. To verify this requirement, we will essentially need to run the whole system several times in order to determine whether or not the average runtime falls within our threshold of 60 seconds.
* **Requirement 13:** *system level*. This is because we can check this requirement manually after the GeoJSON output file is produced using the geojson.io website.
* **Requirement 18:** *integration level*. This is because, for this requirement, we will need to combine data retrieval from the REST server (to get the orders for a given date) with the validation methods inside the Order class in order to establish whether or not an order is valid.
* **Requirement 14:** *unit level*. This is because one class can be used to handle data retrieval from the REST server, and the methods in this class can be tested to ensure appropriate functionality.
* **Requirement 20:** *unit level*. This is because the input validation is done inside the *App* (main) class, meaning that either manual testing or an automated test checking the methods in this class will suffice.