LO1

**Functional:**

* The drone must collect the pizza at the restaurant and return to Appleton Tower in less than 2000 moves.
* The drone must pick the optimal flightpath.
* The drone must not fly in no-fly zones. safety
* Once the drone re-enters the central area, it must not leave the area again.
* The drone must hover when collecting the order so that the order can be stored.
* The drone must hover when delivering the order so that the order can be picked up by the customer.
* The drone must deliver the order at Appleton Tower.
* The drone must begin the order process at Appleton Tower.
* The calculated flightpath must lead the drone to the correct restaurant and safely back to Appleton Tower.
* The customer’s card details must be validated and then stored resulting in a successful purchase.
* The customer should receive the correct order.
* The drone must complete the delivery according to the oldest order in the queue.
* The customer must be able to place their order on the app.

I would go about testing the functional requirements above by performing unit tests as it can test individual components of the software e.g., validation of the card details. I would start by unit testing this order validation requirement to find errors easily, covering every single module. This form of white box testing allows for me to delve into my code and cover the initial bases of the requirement with low maintenance. White box testing for unit tests is essential to cover the individual components of the code. As the developer, performing white box testing would be essential as I am already familiar with my code, allowing me to test the smaller details of the code functionality. However, unit tests are more limited in scope than system tests as they focus on one module and therefore it is hard to see how the components needed for the functional requirements are integrated together with other classes. As a result, I would perform integration testing so that these modules can be combined and tested together. However, integration testing is very high maintenance and finding errors is much more difficult since the behaviour is observed by inputs and outputs. In addition, black box testing would usually be performed for integration tests, where a separate group of testers who don’t know the system would generate responses to test actions without bias of what to test. However, this cannot be achieved because a limitation would be the fact that I would be performing my own integration tests having had developed/designed the software.

**Chosen Requirement (Functional) :**

* The customer should receive their order and it must be correct.

Order validation is an important requirement for my pizza delivery system since it consists of a plethora of sub-requirements essential to the working system. This includes checking the exceptions: Valid But Not Delivered, Invalid Card Number, Invalid Expiry Date, Invalid CVV, Invalid Total, Invalid Pizza Not Defined, Invalid Pizza Count and Invalid Pizza Combination Multiple Suppliers.

I would start by unit testing this order validation requirement to find errors easily, covering every single module. This form of white box testing allows for me to delve into my code and cover the initial bases of the requirement with low maintenance. White box testing for unit tests is essential to cover the individual components of the code. Unit testing using boundary, valid, invalid and absent test data would ensure that there is coverage over these smaller components. However, unit tests are more limited in scope than system tests as they focus on one module and therefore it is hard to see how the components in the order outcome are integrated together with other classes. As a result, I would perform integration testing so that these modules can be combined and tested together. However, integration testing is very high maintenance and finding errors is much more difficult since the behaviour is observed by inputs and outputs. In addition, black box testing would usually be performed for integration tests, where a separate group of testers who don’t know the system would generate responses to test actions without bias of what to test. However, this cannot be achieved because a limitation would be the fact that I would be performing my own integration tests having had developed/designed the software.

**Quality Assurance:**

* Even if there are other issues with the drone, the drone should never go through no-fly zones. (Safety)
* Even if there are environmental issues, the drone should never go through no-fly zones. (Safety and robustness)
* The drone must not drop the order. (Safety and robustness)
* The drone must terminate its flightpath when faced with harsh conditions to prevent entering no-fly zones. (non-functional, safety and robust (subset of dependability))
* The flight path must be generated in under 60 seconds. (Performance testing)
* The drone should be able to detect obstacles in its path.

In the software development cycle, the quality assurance of the requirements above would be most appropriately tested using system testing. This is because a quality assurance team would be assessing the way various components of the system interact with each other in the complete, integrated system. As a result, the quality of the system can be maintained through continuously scheduled system tests so that the entire application can be checked for any errors that may appear and affect the entire system. Since the application is being tested, I would be able to see how it works from the perspective of a user, preventing future crashes and therefore how the application should behave overall. Furthermore, system testing of the quality attributes would allow me to check if changes made in my code have created an error somewhere else in the system. As a result, system testing would allow for me to test the whole application with all the software components interacting with one another. However, a potential issue would be that executing continuous tests is time-consuming. In addition, I do not have a quality assurance team to test the system accordingly, therefore the quality of the system may not be as high due to my lack of experience in comparison. Of course, system testing also means that such tests cannot be performed on individual software components, so the requirement would have to adhere to when the system is fully completed. Performance testing can be used to test the quality attributes of these requirements, for example testing how reliably the drone would be able to detect objects in its path and how dependable the drone is on terminating its flight when faced with harsh conditions. Stress testing in performance testing is also imperative in assuring the application’s robustness under extremely heavy load conditions, increasing the quality of the system. This would be particularly useful with the penultimate QA requirement as it measures performance time.

**Chosen Requirement (Quality Assurance):**

* The flight path must be generated in under 60 seconds.

System testing would allow me to test this requirement using the entire application, timing how long it takes for flight paths to be generated when orders are placed. I would use performance testing to test this requirement as it requires a threshold of 60 seconds. By using this form of testing, I can detect whether the flight path generator can handle a high volume of inputs. Load testing would allow me to stress test the flight path generator through increasing the number of orders from differently located restaurants to simulate a high volume of requests that may potentially occur in the application. However, performance testing of this requirement in the real world is expensive and time-consuming as the environment with which to test must mimic the real-world conditions. In addition, performing only stress testing may produce false positives, as even if the system proves to be able to handle the high-volume load and still generate a flight path in under 60 seconds, the test cases still do not represent the real-world environment. For example, though the flightpath may be generated in under 60 seconds, realistically the drone may still face performance issues in other ways. I would also perform unit test on this requirement to ensure the flight path works accordingly before performance testing.

**Qualitative:**

* The customer’s card details should be stored securely. (Qualitative non-functional) Can’t measure, what is secure
* The customer should receive the order in a reasonable amount of time.
* The drone must pick the order and deliver the order efficiently whilst hovering. (Qualitative non-functional)
* The drone should minimise disruption by flying at a certain height.
* The app should be easy to operate for customers and restaurant employers.

I would go about testing the qualitative requirements that include the security of card details by deploying security penetration testers. This would allow for a range of security vulnerabilities to be detected and therefore provide a higher quality system. It would also be important to be savvy about the latest data breaches so that the system can be better protected from ever-evolving adversary attacks and conforms to up-to-date data protection practices. However, security penetration testing would not be possible for this assignment as it would only be me performing tests as opposed to highly qualified security specialist. In addition, monadic testing can be used for the qualitative requirements that involve user-focussed design. This is because it allows the isolated feedback from the testing to promote a more realistic design of the system with reduced bias. However, the issue with monadic testing is that it would require a high research cost as there must be more people involved in the sample to divide them into multiple groups. This is also unrealistic for this task as it would require many people, however, in a world where this system is fully deployed, I would carry out monadic testing for its ability to address general problems, which are apparent in the vagueness of qualitative requirements. Here, system level testing may be fitting too as it requires checking whether the system delivers the level of service required by the user. For example, ensuring that the app is easy to operate could require extensive system-level testing.