5.1.

Examining the code overall has allowed for me to check for errors even before testing and therefore to change the code. I had realised that not all my exceptions were explicitly handled and therefore if not reviewed, could’ve led my system to crash. Furthermore, I had noticed that my code for one of my classes could’ve been formatted better so that it could be read more clearly by other potential software engineers.

5.2.

The CI/CD pipeline for the pizza delivery drone system consists of:

1. **Commit**: In the initial step, the software’s source code is managed by using a version control system like GitHub which will store the code in a central repository where changes can be tracked and commits can be changed.
2. **Build**: Next, automated build tools like Gradle/Maven can be used to build the delivery system. As a result, the software will be tested and compiled, and it will be packaged into a deployable artifact using the build tool.
3. **Unit Testing**: The software should be tested after it has been built. Individual software components might be tested individually by the system using unit tests. These unit tests are run automatically to ensure that the code is operating as intended.
4. **Integration Testing**: The software should be integrated and tested on the target platform following the unit tests. This should also be done once all of the unit tests have passed, so that the integration testing can work. The testing of my software can be performed on several configurations and operating systems to ensure that it is successful on multiple platforms.
5. **Deploy**: After the integration tests have also passed, further Quality Assurance methods can be performed alongside other tests after the deployment of the system, to the staging environment.
6. **Performance Testing**: After the deployment stage, performance testing should be performed to ensure that the software can manage the anticipated load through tests such as stress testing, spike testing, load testing and scalability testing. This would be especially useful as it would consist of testing quality assurance requirements, where the users may have needs that should be incorporated in the system.
7. **Review**: The software can then be reviewed externally where errors and anomalies in the code can be checked by one/several people and discuss with the developer who will then go and fix the issues with the code.
8. **Production**: Lastly, using a release management tool, the pizza delivery system will be ready to be released for production and monitored continuously.

5.3.

The features of my system is directly accessible through an API so that tests can directly access them without the need for a graphical user interface (GUI) which would be more costly to automate. I would automate:

**Order Validity**: Automating the orders so that it can be tracked to allow for the user to receive the correct pizza would ensure that the software does exactly what it should do.

**Flightpath**: Automating the flightpath to allow the drone to remember previous flightpaths for similar deliveries and consider other factors such as weather in curating the optimal path.

**Drone Attributes**: Automating the drone’s attributes for example, when it starts on the flight path, reaches Appleton Tower, and hovers so that the drone’s movements can be continuously monitored.

5.4.

The CI pipeline for my system consists of automating the integration of code changes, as shown in the following stages:

1. **Code compilation**: The most recent code is compiled.
2. **Unit testing**: Automated unit tests are then tested on this code to ensure that each section of the code work correctly.
3. **Integration testing**: Automated integration testing checks the behaviour of different units of the system altogether. It allows for the latest code to be integrated with the existing software, this ensures that this code interacts well with the rest of the system.
4. **Deploy**: After the automated unit and integration tests have passed, further Quality Assurance methods can be performed alongside other tests. This comes after the deployment of the system, to the staging environment.
5. **Performance testing**: After the deployment stage, performance testing should be performed to ensure that the software can manage the anticipated load through tests such as stress testing, spike testing, load testing and scalability testing. This would be especially useful as it would consist of testing quality assurance requirements, where the users may have needs that should be incorporated in the system.
6. **Security testing**: Automated security tests are incorporated to tests for potential security issues and vulnerabilities.
7. **Production**: Lastly, the latest code can be used in the production environment once all of the tests have passed.

Issues including defects, compatibility problems, and performance issues would be found by the CI pipeline. This is because the CI pipeline requires all tests to pass in each step. The process would immediately halt and the developers would be informed if any problems were found.