TEST AUTOMATION

5.1 Identify and apply review criteria to selected parts of the code and identify issues in the code

Various criteria could be used to review different qualities of the code. For example, understandability and learnability are criteria used to evaluate the usability of a system. For the PizzaDronz system, I will review its testability and try to resolve any issues which arise. Additionally, there are various methodologies to review code qualities. This includes code reviews, inspections and walkthroughs. I will review the code testability using code reviews, or peer reviews, as it is less time-consuming than inspections but more thorough than code walkthroughs. The results of the inspection are shown below.

Testability – how straightforward is it to test?	Yes/no, add supporting comments if needed
Project has unit tests	Yes
Project has integration tests	Yes
Project recommends tools to check test coverage	Yes
Project has automated tests to check test coverage	Yes
A minimum test coverage level that must be met has been defined	Yes
There is an automated test for this minimum test coverage level	Yes
Continuous integration is supported – tests are automatically run whenever the source code changes	No
Test results are visible to all developers	No

Although there are many more criteria missing from the table above, it contains relevant points which could be planned or executed with my current resources. Additionally, the rows which were marked “no” were mostly due to a lack of a CI pipeline, which will be tackled in the parts below.

5.2 Construct an appropriate CI pipeline for the software

Before constructing a CI pipeline, I will lay out the general stages of a CI/CD pipeline, which consists of 4 stages: source, build, test and deploy.

In the source stage, whenever a change is made in the system’s source code, data, environment or configuration, a new instance of the pipeline will be activated. The main purpose of this stage is to maintain the system in a state that can be released to an environment. Generally, this stage could be accomplished using a repository and a version control system. Thus, I will create a repository on GitHub for this project.

In the build stage, the CI pipeline helps build and compile the application, create packages from the source code and provide early feedback to developers. This aids in maintaining the system in a state which could be released into an environment. Generally, this stage is accomplished by choosing a CI engine, compiling the code and running the system. Using GitHub Actions as my CI server, I will first set up a pipeline as code in the GitHub repository. This allows the creation of audit trails for easier version tracking and eases collaboration between developers. Then, I will implement a job in the pipeline that compiles the source code. Following that, I will run unit tests and code analysis to ensure there are no code smells or deeper faults. Lastly, the pipeline will generate a versioned and built artefact, which will be published to a store for testing or deployment.

In the testing stage, most CI pipelines evaluate the build of the system and publish the corresponding results. After that, the system is released for production. This stage ensures that any changes made to the source code do not break any functionality and ensures the code's safety for release. A few example tests would be integration, performance, compliance, system and acceptance testing. To achieve this, I will utilize the JUnit plugin to integrate within the pipeline, as I am most familiar with it. After running all tests, I will publish the test report and code coverage report to ensure the results of the testing are easily available in the pipeline run. The last important step would be to set a minimum code coverage before releasing a software build. This serves as a quality check and will fail the stage if code coverage drops below the set threshold.

The last stage, deployment, is covered in the CD part of the pipeline. It mainly involves delivering and deploying the final build by hosting the system in the cloud. Hosting the system could be done by using server providers like firebase or Heroku. I will not go in-depth into this as the question only asks for CI pipeline and doesn't include CD, but in summary, CD involved deploying and obtaining feedback and statistical data from the deployed system.

5.3 Automate some aspects of the testing

There are a few ways in which the testing stage of the CI pipeline could be automated. Firstly, the unit tests written for white-box testing could be automated using JUnit. Some scaffolding will be used to set up test parameters and mock objects that simulate the functionality of undeveloped code. The unit being tested will then be executed with the predetermined parameters. Lastly, an assertion comparing the expected and actual output of the system will be used to automatically check if the system functions successfully as intended.

Additionally, the synthetic generation of scaffolding like mock data, for example, could be efficiently generated using automated processes. Ideally, this would be done by fitting real data to a known distribution to produce the most accurate synthetic data, or even through deep learning techniques like variational autoencoders. Although this would greatly improve the efficiency of the testing stage, it also bears some risks, which were covered in "test planning" and "test evaluation" documents.

5.4 Demonstrate the CI pipeline functions as expected

Various issues could be identified by the CI pipeline. Firstly, any bugs caused by changes in the system's source code will always be caught in the CI pipeline. Previous test suites created will always run during the build stage of the CI pipeline. Thus, if there are changes to previously developed code that changes the module's functionality, the system will fail at the testing phase of the CI pipeline. additionally, the CI pipeline can help improve the development of code by identifying inadequate test suites more quickly and efficiently due to the feedback loop of the CI/CD pipeline. After deployment of the code and acceptance testing is conducted, developers will work on patching bugs missed during initial development and adding the corresponding test cases to the automated tests. Then the system is deployed again, and the loop continues.

Lastly, the CI pipeline can catch systems which fail to satisfy the predetermined adequacy criteria. In the testing phase, there could be automated tests set to verify the performance of the system. For example, having an assertion to the minimum amount of code coverage achieved or an assertion for checking the minimum response time of the system. With the CI pipeline, systems below the minimum performance level will not be mistakenly or unknowingly deployed.