Automated Testing Processes

Construction of an appropriate CI pipeline for the software

The following steps show an example of how I would set up a Continuous Integration (CI) pipeline for my Informatics Large Practical (ILP) project "PizzaDronz".

1. **Set up the version control system.**
   I would set up choose a version control system to track all changes to my project. This can be done by creating a repository for the ILP project on a hosting platform like GitHub or GitLab.
2. **Choose a continuous integration tool.**
   I would choose a continuous integration (CI) tool to automatically build the code and run tests whenever a change is made to my project. In particular, I would use Jenkins as it is a popular open-source CI tool that is easy to use, supports a wide range of plugins, can be installed on Windows, MacOS and Linux and has a large community that provides support if needed.
3. **Install Jenkins.**
   I would install and configure Jenkins on my machine and set up users and permissions.
4. **Create Jenkins job.**
   I would create a Jenkins job for my project and configure it to build when changes are pushed to my version control system repository.
5. **Configure the build.**
   I would specify the necessary build steps in the Jenkins job configuration. These will include checking code from the version control system repository, running 'mvn clean install' to compile, package and test the code and archiving the build artifacts.
6. **Configure test.**
   In the Jenkins job configuration, I would specify the necessary steps. These will include running unit tests and integration tests using 'mvn test' and 'mvn verify' respectively as well as running functional and end-to-end tests using a separate command, such as 'mvn test -P functional-tests'.
7. **Deploy staging environment.**
   Once the build and tests pass, I would use the Jenkins Deploy Plugin to deploy the software to a staging environment for further testing. The further testing can involve performance tests (using Apache JMeter as my testing tool) among others.
8. **Release**
   Once the tests pass, the Jenkins job would release the code changes to the production environment.
9. **Monitoring and alerting**
   The production environment should be monitored for any issues and alerts (such as emails) should be set up to notify the developer.

Demonstration that the proposed CI pipeline functions as expected

The build server will pull the latest code from the version control system each time a change is pushed and run series of tests on the code, whenever it is built. These tests would include different levels: unit, integration, system, performance, code quality, etc. And then, if all tests pass, the build server will deploy the code to a staging or production environment, while also notifying the developers. If not all tests pass, the build will fail, and the developers would be notified.

The kinds of issues that could be identified by the proposed CI pipeline would discover, include (but are not limited to):

- Failed tests. The failures will be identified by, for example unit, integration, system tests among others that can be found in the testing repository.
- Syntax errors. These prevent the code from compiling and would be identified by the CI pipeline.
- Deployment failures. These could be problems that prevent software from deploying successfully to the staging or production environments. This can be done by using the Jenkins Deploy Plugin, which will fail the build if it discovers any issues during the deployment process.
- Misconfigured/incompatible dependencies. The code changes could introduce problems that cause conflicts with the dependencies of the project. This will be identified by Maven inside the POM file, and if any issue is found the pipeline would fail the build.
- Performance issues. The code changes could cause the system to perform poorly. These will be identified by the Apache JMeter as my testing tool.
- Code style violations. The code changes could contain inconsistent formatting or naming conventions and that will be identified by tools such as Checkstyle.
- Configuration errors. The code changes could introduce errors in the configuration files. The pipeline will use tools like Ansible or Puppet to check the configuration files for errors, and if any errors are found, the pipeline would fail the build.

In all cases, the pipeline will identify the bug and will fail the build, while also notifying the developer. The developer would be expected to fix the issue and upload the changes.