Inspections and automated testing process

Review and Inspections:

Reviews may have set activities to go through as part of the process, since we do not have another programmer reviewer to ask for help, we will review our code ourselves. In our case, we want to **systematically check the code** for its functionally, where any issues that arise be noted of. Any issues that arise should be dealt with and documented. The last step is to modify the code to address the issues that were discovered.

A review technique as part of the code review process is the **checklist for review**. This will go through a simple final inspection of the software. As part of the code review, this technique ensures that the code quality is as good as possible. Examples include: **meaningful variable and function name**, making sure **all data errors have been dealt with**, if **all exceptions are handled,** **correct typing in the programing code** and **well formatted code**. Through the review, some issues that were identified included the incorrect naming of some of the variables in the code with ambiguous names, which was found and fixed.

# Constructing the pipeline and Automated Testing

I did not have time to fully set up the Continuous Integration(CI), however, the following steps show how I would go about constructing the pipeline for my chosen project.

1. Set up the version control system:

Version control system (VCS) keeps track of all changes made to the project. This is easily done by creating a repository on GitHub or GitLab. In this case GitHub was chosen

1. Select a continuous integration (CI) tool:

After the VCS is setup, a CI tool will be chosen to automate code building and the running of all test files whenever a change is committed to my project. I opted to use Jenkins, since it’s well known, and its an open source CI tool that is simple to use.

1. Download and set up Jenkins:

Install and configure Jenkins on my device and set up users and permissions.

1. Create and configure a Jenkins job:

Create a Jenkins job for my project. The job would be configured to build when changes are pushed to my VCS repository. The configuration would specify the necessary build steps in the Jenkins job configuration, this includes going to the VCS repository for checking code, running ‘mvn clean install’ command for compiling, packaging and testing of the code, and archiving build artifacts.

1. Configuring the test suite:

The Jenkins job configuration we created earlier contains the following:

* using the ‘mvn test’ and ‘mvn verify’ commands to run the unit and integration tests.
* running functional tests, plus end-to-end tests using a separate set of commands.

1. Deployment of the staging environment:

Once the build is successful and all tests pass, I would deploy the software to a staging environment (for further testing) using the Jenkins Deploy. Further testing may include performance tests which I could recommend the SOAPUI tool.

1. Release and monitor the tool:

If the tests do pass, the set up Jenkins job will now release the code changes to the production environment. The environment will be checked for any faults. When faults are found, notifications can be set up to warn developers.

# Demonstrate the pipeline functions as expected

Whenever a change is made, the build server from Jenkins configuration will pull from the newest code version to run the test on the CI pipeline. The tests include all levels (**unit, integration and system**) and will also include performance attribute tests. Encountering any failed tests or issues, the build will fail, the developers will be warned that issues have been found. Otherwise, the server will deploy the code to a staging environment, and the developers will be updated.

Below are some issues that the pipeline may detect that will cause faults when changes are made to the project:

* Failing of the **basic functional tests** (including unit, integration or system), this would cause the pipeline to not build and fail.
* Simple **syntax** errors will cause the build to fail, as the java program will fail to compile when trying to combine the source code and the dependencies.
* Wrong **dependencies** – again, the build will fail when trying to compile, as the maven dependencies code not support the currect code changes
* **Poor code formatting or violations** – this could be detected naming violations, not following conventional formatting etc. Extra tools may be needed to be added to Jenkins for this detection.
* **Poor Performance** – This, again, requires the installation of an additional tool, such as SOAPUI. The pipeline may fail depending if the performance is longer than expected, perhaps due to unnecessary loops or unoptimal code.

Once the system updates the developer of the pipeline situation, if there are issues with the automated tests, the developer will be able to find and solve whatever fault that has risen.