Reviews, Inspections and Automation

This document will conduct reviews, inspections and design how an automated testing process would be put in place for the Pizza Dronz project.

Review criteria

A review of the system will check many of the measurable quality attributes that are difficult to quantify, including code maintainability, testability and readability. There are several different ways that a code review could be undertaken but I've chose to use a mixture of the following three.

Peer review

This involves getting another developer to visually review your code, and identify any problems that it may have, or areas that could improve. This could include how maintainable, readable and testable they believe it to be. This is one of the less time-consuming and easier methods of reviewing a system.

Tool assisted review

This strategy involves using automated tools to go through the code, and then suggest areas that could change. This could be as simple as suggesting clearer indents or better variable names to improve readability and uniformity throughout the code.

Checklist review

A checklist review involves having a list of predefined guidelines and structures that your code must follow, then going through the code and ensuring that these rules are adhered to. This will both help to find bugs, but also improve coding practices ensure that the code is formatted neatly and is readable. This review could be carried out by yourself, or by another team/developer.

Any information gained from the reviews was taken into consideration, and were acted upon if necessary. For instance, anything that did not conform to the checklist review was changed so that the code was uniform throughout. Additionally, through peer review I found that some of my variable names were confusing, which would impede maintainability, so these were changed. The tool assisted review helped to find anything that I missed during a checklist and peer review, such as strange indentation at places that made the code harder to follow.

Construct a CI pipeline

The following will explain how I would've constructed a CI pipeline, if I had the time to do so.

The first step is to choose a Version Control System that will house and manage my source code. This is so that whenever a change is made to the system, a new instance of the pipeline can be activated. For the Pizza Dronz project, I would create a GitHub repository and I would store all of the code there.

Next, a CI/CD platform must be chosen, which will build and compile the application, whilst also running the appropriate tests and analysis on the code which will then be stored as feedback for the developers. As GitHub is being used as the VCS, it makes sense to use GitHub Actions as the CI server. Using this, a pipeline will be created in the repository that will compile the source code, run unit tests and code analysis to ensure that the source code is not faulty, and then create a build artefact that will be stored somewhere in the repository for the developers.

At this stage the pipeline will test the build automatically, and then make the results available for the programmer. Only after this is done, is the system released. This stage is vital, as it ensures that any code changes do not impede on the previous functionality of the system, and does not introduce unwanted bugs. It will run all manor of unit, integration and system level tests and test code coverage. In this case, JUnit can be used within the pipeline to create these tests. We can then impose restrictions for whether or not the code can be released, such as a minimum code coverage threshold. If these are passed, then the build will be released and the results of the tests and the coverage will be revealed to the developer.

This pipeline will need to be monitored every so often to ensure that it is working correctly and is not publishing incorrect test results to the developer. If any faults are detected, these should be fixed before continuing to push code to GitHub.

Automating some aspects of the testing

One thing that could be automated is the creation of synthetic data for the system. This would be vastly beneficial, as it is easy for a developer to miss or overlook certain possible scenarios when developing simulation data. If the synthetic data was generated automatically by taking in real world data and then using some machine learning techniques to produce accurate synthetic data, and maybe uncover some edge cases that would not be covered otherwise. However, this would take a great deal of time to create.

Unit tests could also be automatically generated throughout development, which would lead to grey box and white box unit tests being available. These could be made by JUnit, and would test new functions as they are added to the code. This has the potential to

speed up test generation significantly, but may also result in the generation of unit tests that either do not work in a desired way or do not tests the requirement thoroughly enough.

Demonstrating CI pipeline functions

The CI pipeline has not been constructed, so it cannot be demonstrated working, but I will describe the ways in which it will function if the CI pipeline were to be implemented.

When code is modified or added to the GitHub repository, the pipeline will compile this code and check for any issues in the source code. Therefore the first way to demonstrate that it is functioning is to see if it catches bugs in the source code successfully. If this is the case, it will not be released and a bug report will be available to the developers.

Then the code will undergo testing. The pipeline will always run previously made test suites whenever there is a code change, and so this will quickly discover whether or not a bug has been introduced to previous functions.

Due to the continual feedback loop that the pipeline provides, better test suites can be identified and added more efficiently. Once testing is completed by the pipeline, developers can quickly see the test reports and fix whatever issues have arisen in the code. Once this is done, they can push the new code and this will once again go through the pipeline, thus continuously giving feedback on what should be improved or changed.