Quality assurance

In this section we will introduce the reader into the importance of quality and how we focused on it when working on the GTL project.

Why is quality important?  
Before we go into detail about the importance of quality, lets first talk about what quality actually means. Quality in software refers to:

* The structural quality of the software –which embodies the robustness of the codebase of the software. Maintainability, supportability, the number of bugs or likeliness to crash are all taken into consideration. In short quality code has reduced cyclomatic complexity and is testable, so when it is delivered in the form of an application it can properly be expanded on and maintained years after its initial release
* The perceived quality of the software – in this case we focus on the satisfaction of the initial requirements given by the product owner and how highly the target group rates the final product. Of course, perceived quality can only be achieved if the code of the application is up to scratch as it directly influences the perception of the software.

A great way to tie up structural quality and perceived quality is by utilizing the acronym FURPS, which is a way to classify and validate the software against the gathered criteria. FURPS is short for:

* Functionality – refers to the features the target audience wants to see in the release

Functionality represents the main functional requirements in the system, in our case (regardless of priority/risk severity) it would be: Borrowing a book, returning a book, producing statistics, etc.

The rest of the URPS attributes represent the other non-functional requirements which are architecturally important.

* Usability – how effectively can the target group use the application, focusing on the learning curve of the user interface and how good is the UX (user experience)

As our project does not really focus on the UI of the application, we will not go in detail about the UX and usability

* Reliability – how robust must the application when interacted with, we must also take into consideration the up time, the down time and the quality of the data produced/received

Considering the environment and the use case of the application reliability is a high priority, its one of the main reasons we chose the V-Model (the development process we will go into detail later in the report), as we felt it would yield a higher quality product than its Agile counterpart, the four quadrants.

* Performance – how resource intensive is the application when used by the customer and their users

As with the usability, we cannot accurately give a number or estimate as to how resource intensive the final application will be as we do not have a fully developed UI and have not finished developing every user story.

* Supportability – how easily the application can be maintained after its initial release by (possibly) future development teams

The application was built with supportability in mind, hence the reason for the usage of code standards, numerous tests, which test most aspects of the software

While FURPS is used to ensure the overall application measures up to the global criteria of the project, we must also focus on the individual code snippets of the application. Sadly, it is not feasible for us to test every part of the application, but we can define a test coverage by selecting the most important parts of it (done in risk assessment) and focus on making the vital parts of the application as pervious to errors as possible. For the rest of the application not tested as rigorously we maintain quality by using coding standards and constantly review each others, work before approving adding it to the application.

Cyclomatic complexity

One of the most important metrics for code quality, it refers to the complexity of the code, how many decisions must it go through for it to execute a specific function, the higher the number, the worse it is, as the higher number of decisions an application must make the greater the risk of it to produce an error. If we convert a method’s code to a flow graph, the number of linearly independent paths will be the cyclomatic complexity.

The value to complexity ratio is the following:

1 – 10 is a low risk program  
11 – 20 is a moderate risk program  
21 – 50 is a high-risk program  
anything greater than 50 is a highly unstable program and should be refactored for simplification

As shown our application scores favorably when looking at the maintainability index and the cyclomatic complexity. When looking at the cyclomatic complexity score, they mostly range from 1 to 9 which are considered to be low risks, with one method being an exception with a moderate risk level of 13, meaning the application’s code is of high quality.

