REVIEW FEEDBACK

# Hamish Arro 04/08

**04 August 2021 / 09:00 AM / Reviewer: Ronald Munodawafa**

**Steady** - You credibly demonstrated this in the session.
**Improving** - You did not credibly demonstrate this yet.

## GENERAL FEEDBACK

Feedback: Hi Red Ponyta 99,

Despite the long time you've spent away from reviews you've remained steady across all the course goals. You can refine your information-gathering by considering finer details and edge cases before programming. You can also look into including refactor phases in your red-green-refactor cycle.

Well done on the review and all the best in your future endeavours!

Kind regards,
Ronald

## I CAN TDD ANYTHING – Strong

Feedback: You based your tests on the acceptance criteria, which kept them oriented towards the client's direct needs and focused on behaviour.

You adhered to the incremental red-green-refactor cycle, choosing simple tests in your red phases and only progressing slowly towards the more general solution that the client was seeking. You applied test triangulation quite well in this review. As a result, your red-green-refactor cycle iterations were quick and simple.

## I CAN PROGRAM FLUENTLY – Strong

Feedback: You were comfortable working with a development environment that was heavily based using a terminal and editor only. This opened you up to a lot of flexibility in terms of tooling. You might find it helpful to compose scripts for setting up Git and your process better.

You were very comfortable with Ruby's syntax and aware of the built-in methods for processing strings and arrays. As a result, you were able to construct a logical algorithm almost without referring to documentation. You had the language at your fingertips.

## I CAN DEBUG ANYTHING – Steady

Feedback: You did not encounter any major bugs. However, you did read the backtrace and based your changes in the green phase on the discrepancy between the expected and actual output. This demonstrated that you were able to make informed changes in a simple error-handling scenario.

You handled the only minor bug you encountered gracefully by looking up the correct syntax for the ternary operator. You took the time to go through the backtrace and understand the problem before acting. Well done!

## I CAN MODEL ANYTHING – Strong

Feedback: You modelled your solution as a single effective method named 'report', which was a sufficient abstraction for the problem's stateless nature. Your names adhered to Ruby's naming conventions and names best practices. You also provided an elegant algorithm for the problem.

## I CAN REFACTOR ANYTHING –Improving

Feedback: You kept refactoring for later than was necessary. You could have applied the single-responsibility principle to ensure that you cleaned up your code and removed any redundancies such as the counting of grades while they

were easy to do. You could have also considered using a hash to keep track of your grade counts. Refactoring in the refactor phase helps you control the quality of your code without incurring a huge development cost in the end.

## I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Steady

Feedback: You followed a regular red-green-refactor cycle. However, you could improve on it by including refactor phases instead of skipping them. This will help you keep your code maintainable.

You prioritised edge cases over edge cases, proving immediate value to the client. You also did so with a test progression that increased in complexity, allowing for a systematic approach to your development.

## I USE AN AGILE DEVELOPMENT PROCESS – Steady

Feedback: You noted the information the client shared in detail. You managed to gather main requirements such as the rages for the respective grades. You, however, needed to consider how the input might vary. You could have looked at test scores that fall outside of the range 0 - 100 or extra commas, for example. It would have allowed for the consideration of finer details and edge cases before you started programming.

## I WRITE CODE THAT IS EASY TO CHANGE – Steady

Feedback: You committed after every working change. In the one case you forgot to, you were quick to notice and commit as soon as you could. I recommend considering using pre-commit Git hooks to streamline your usage of Git in the context of the red-green-refactor cycle. Your commit messages were also meaningful and helpful though they could still be made conventional by starting them with a capital letter.

You tested at the implementation’s system boundaries and avoided testing for intermediate steps. As a result, your specification and implementation were decoupled, which gave you opportunities to refactor you should have considered to improve your code’s changeability.

You chose descriptive names that were derivative of the domain, making your code human-readable.

## I CAN JUSTIFY THE WAY I WORK – Strong

Feedback: You were engaging with the client and provided valuable justification for your steps. As a result, it was easy to understand why you deviated in certain situations and it was also easy to understand your thought process.