REVIEW FEEDBACK

Jack Shields 11/03

11 March 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: You're improving quite well. Your information-gathering is more comprehensive! Your test-driven development has also improved incredibly well. You've improved across all the areas. Well done on the improvements and the effort you've invested in improving your process. You can still improve by not leaving a refactor phase until no refactorings are possible. I encourage you to continue working hard!

I CAN TDD ANYTHING – Steady

Feedback: You derived your tests from your input-output table, which focused your tests on the band pass filter's behaviour from the client's perspective.

You followed the outside-in approach to test progression by incrementing on your algorithm while you fixed the number of frequencies in the soundwave. This allowed for a sensible direction in your algorithm's development in an incremental manner.

You adhered to the red-green-refactor cycle, making judicious use of the green phases to move.However, you could have used the refactor phases to keep your code clean.

I CAN PROGRAM FLUENTLY – Steady

Feedback: You used the terminal and your editor to set up, navigate and access your development environment. You used the terminal to run the Unix utilities, Git and RSpec.

You were quite familiar with Ruby's language constructs and could program your solution without the aid of documentation. You were familiar with the Array class and array processing using the language's primitives.

Your algorithm was sensible and iteratively developed. Perhaps, you could have considered Array.map to simplify your logic. Other than that, your Ruby was idiomatic.

I CAN DEBUG ANYTHING – Steady

Feedback: You were familiar with the common errors and could use the backtrace to direct changes in your implementation for your tests to pass. Due to your excellent test-driven development, you did not encounter any bugs that prevented your from continuing.

I CAN MODEL ANYTHING – Steady

Feedback: You constructed an input-output table to model the band pass filter's behaviour in terms of the input and output sound wave and limits, taking note of the mapping between the two.

Your filenames, method, parameters and variables adhered to Ruby's naming conventions. You also applied the best practice of naming your stateless method with a noun phrase.

Using a method was appropriately minimal and simple given that the problem was inherently stateless.

Your algorithm was quite sensible!

I CAN REFACTOR ANYTHING –Improving

Feedback: You used the refactor phases to remove hardcoding and were able to provide a reasonable solution. Perhaps, you could have considered the single-responsibility principle to simply your method's complexity. You could have also simplified the iterative logic to Array.map. These are both considerations you could have made during your refactor phases. I encourage you to make use of them.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Steady

Feedback: You adhered to the red-green-refactor cycle but left out the refactor phases, which would prove useful in simplifying your code for future iterations.

You prioritised core cases above edge cases, providing immediate value to the client. Your tests also proceeded in a stepwise manner, leading to a logical progression in development.

You might want to research the randomised population of an array and use a timing module from the standard library whose output you'd then use in assertions to test for the processing speed.

I USE AN AGILE DEVELOPMENT PROCESS – Strong

Feedback: You asked about the band pass filter's operation with gestures to clarify your understanding, which was brilliant for communicating with the client. You took down notes of all the relevant information the client provided you. Your analysis enabled you to uncover finer details relating to the default limits and the expected wavelength. Your information-gathering was quite comprehensive.

You used your input-output table for prompting the client for clarification. You also asked them questions through the session to ensure you met their needs.

I WRITE CODE THAT IS EASY TO CHANGE – Steady

Feedback: You committed every test-passing version of your solution with descriptive messages, which was useful for documenting your project's development history. You could improve it further by describing the scope of work for the commit instead of the test that is passing. The comments above your tests were good candidates for commit messages.

You did not test for intermediate steps, which decoupled your tests and code. You had the opportunity to refactor as a result. I recommend taking advantage of the refactor phases to improve your code's overall changeability.

You derived your program names from the domain's language, which clarified the intention behind your code. Your test names were descriptive, which helped with denoting your code's behaviour. Perhaps, you might want to group your tests to classify what they represent and improve the clarity of your code's usage. You used comments but comments are ignored by your test runner.

I CAN JUSTIFY THE WAY I WORK – Steady

Feedback: You vocalised your process quite well and your workflow was clear. You pitched your comments clearly enough for the client to understand your reasoning. Your reasoning was also logical. You can improve your reasoning further by justifying deviations in your process such as skipping refactor phases. You could improve your vocalisation further by updating the client regularly with the progress of work. That said, you did well!