REVIEW FEEDBACK

Lucy Stringer 05/02

05 February 2021 / 02:00 PM / Reviewer: Ronald Munodawafa

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

GENERAL FEEDBACK

Feedback: Your programming has improved quite considerably from the last reviews. I appreciate the resourcefulness you employ in your process. I encourage you to spend more time on reading backtrace messages, your testing discipline and including refactor phases. Focusing on these will refine your brilliant process further.

I CAN TDD ANYTHING – Improving

Feedback: You made your first tests pass by hardcoding, following an outside-in approach. However, you switched to the inside-out approach in your second green phase by introducing logic that would apply the low-pass filter to multiple frequencies. You could reduce your process' complexity by consciously committing to either the inside-out approach or outside-in approach based on whether you have insight into the problem's solution. Being aware of the risk of overengineering with the inside-out approach might be a helpful consideration.

You followed a depth-first approach by ensuring that the low-pass filter worked before proceeding to the high-pass filter and integrating them as you went along. It might complicate your integration of the separate bits of logic in some complex scenarios, but offer you the benefit of being able to parallelise your development if you're working with a teammate. These are some considerations for deciding on the testing path you choose to follow.

You based your tests on the acceptance criteria which focused your development on the client's needs by ensuring that your tests were representative of the client's needs.

I CAN PROGRAM FLUENTLY – Strong

Feedback: You were capable of using the terminal to access your development tools. You were comfortable with Ruby's syntax enough for you to compose the RSpec tests and use features such as symbols to simplify your solution. Your ability to manipulate arrays and use Ruby's control flow features enabled you to implement a logically flowing solution.

I CAN DEBUG ANYTHING – Improving

Feedback: You did not read the backtrace messages when debugging the introduction of symbols, leading you to the territory of random bug fixes. I encourage you to pay attention to your backtrace messages and the information they provide such as line numbers to identify the assumptions you made when writing your code. You would then use print statements or IRB in a feedback loop to test for those assumptions and identify the source of bugs.

I CAN MODEL ANYTHING – Steady

Feedback: You developed your solution as a class, offering the benefits of extensibility and clean namespacing. You could have considered moving the specification of the parameters of the filter's limits from the method 'modulate' to the constructor to allow reuse of the same filter limits.

I also appreciate that you used pseudocode in some of your iterations to model your algorithm, which ended up being quite sensible and relatively straightforward with the array mapping.

I CAN REFACTOR ANYTHING –Improving

Feedback: I encourage you to take advantage of the refactor phases as part of the outside-in approach when you determine a common algorithm for the behaviour implied by the duplicated logic.

I also encourage you to consider applying the single-responsibility principle to reduce the complexity of your methods.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Steady

Feedback: Your test cases represented the requirements in increments of complexity, making your process incremental and allowing you to reduce the complexity of problem-solving you had to engage within each iteration.

While you adhered to the red-green-refactor cycle, you excluded the refactor phases. Refactor phases afford you the opportunity to remove duplication and clean up your code, which is helpful in easing your outside-in test-driven development and future iterations.

You successfully researched the use of symbols while debugging but did not use the information from the backtrace messages. Research's value is improved when we can use the details our development tools provide us with to inform our search. Ironically, it's easier to find what you are looking for once you know what it is.

I USE AN AGILE DEVELOPMENT PROCESS – Steady

Feedback: You asked about the inputs and outputs which was a good starting point for comprehending the requirements. You asked about the representation of sound waves and frequencies, making the requirements concrete. You asked about the default and custom limits, a finer detail of the requirements. You had prior domain knowledge that you tried to integrate into the problem, which was a brilliant way to meet the client on your understanding.

I encourage you to explore the input's variation by considering potentially invalid values that would still be the correct data type and extreme cases such as empty sound waves. It might help you discover edge cases.

I WRITE CODE THAT IS EASY TO CHANGE – Steady

Feedback: You committed every test-passing version of your solution which was helpful for documenting your project's development history. You might want to consider the convention of using a capital letter to start your commit messages. Nonetheless, they described the scope of work for that commit and were clear for that reason.
You also kept your specification and implementation decoupled. You therefore had plenty of room to refactor and could have used your refactor phases to make your code easier to extend and modify in future iterations.

You used sensible names that indicated what they represented, which was quite helpful in understanding the intention behind your code.

I CAN JUSTIFY THE WAY I WORK – Improving

Feedback: Your comments were loud and clear, making it easier to understand what you were doing. You could improve this aspect by providing reasons for those actions including deviations from the process such as skipping refactor phases or not reading backtrace messages. It improves the client's understanding of your workflow and helps you tighten your process' reasoning.

I also suggest you update the client on the progress of work whenever your product meets an acceptance criterion. It helps the client understand your work in relation to the completion of work.