REVIEW FEEDBACK

Holly Duckett 09/03

09 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: Your process demonstrated very good adherence to agile and test-driven development principles and your process vocalisation was also quite good! You could improve further by paying attention to your red-green-refactor cycle iterations' complexity and refactoring in the refactor phases. That said, your overall your process was good!

I CAN TDD ANYTHING – Steady

Feedback: You based your tests on the input-output table, allowing you to test for the band pass filter's behaviour from the client's perspective. Your development process was therefore oriented towards the client's direct needs.

You followed the inside-out approach to your test progression by focusing on a single behaviour before moving on to the next and integrating them afterwards. For example, you focused on filtering with the minimum frequency before proceeding to testing for filtering with the maximum frequency. You were able to integrate your filter's behaviour quite well.

You adhered to the red-green-refactor cycle quite well. Your green phases could have been made simpler by starting with simpler tests such as single-frequency sound waves and by using multiple tests. This is because, as is mentioned by Robert Martin, "As your tests become specific, your code becomes generic." In test-driven development, you generalise your solution by test-driving it using multiple tests making incremental changes in each iteration. Instead of applying an iterative higher-order function and

introducing conditional branching, you'd perform these two actions in two separate green phases.

I CAN PROGRAM FLUENTLY – Steady

Feedback: You were comfortable with using the terminal and editor to access and navigate your development environment. You were able to run RSpec and use Git from the terminal.

You were so familiar with Ruby's language constructs you did not need to use documentation to implement most of your solution. You were also familiar with the Array class and array processing in general. As a consequence, you developed a logical algorithm.

Your overall approach was fine. Perhaps, you could have considered using Array.map given it's a more natural method for the problem rather than Array.each. It would also simplify your code. That said, your Ruby was generally idiomatic.

I CAN DEBUG ANYTHING – Steady

Feedback: You were familiar with the most common errors and could easily target the information on the actual vs got values from your tests. You correctly used them to locate the bug with your tests that involved testing against a 2D array rather than a 1D array. Your debugging was systematic.

I CAN MODEL ANYTHING – Steady

Feedback: You constructed an input-output to model the band-pass filter's behaviour and interface. Your input-output table also included descriptions of the behaviour that each entry represented, making your input-output table excellent for deriving tests. You could improve your input-output tables further by including simpler examples of the behaviour such as empty

soundwaves and account for aspects of the requirements that could influence the interface, e.g. default filters.

Given the stateless nature of the problem as implied by your input-output table, a single method was minimal and appropriate. It would allow for simple access to the band pass filter.

You paid attention to whether the names you'd use were single words or multiple words and used this information to influence your naming scheme. You named your method, its parameters and your files according to Ruby's naming conventions. The noun phrase you used for the method was also suitable for the method given that it was stateless.

The iterative transformation of the sound wave was a naturally-fitting solution to the problem, which made your algorithm quite sensible.

I CAN REFACTOR ANYTHING –Improving

Feedback: You did not refactor during any of the refactor phases of your red-green-refactor cycle. You could have considered the single-responsibility principle to reduce the complexity of your method and help it read more declaratively. You could have also considered refactoring the Array.each to Array.map since the latter is directly specialised for problems of this nature. You could have also grouped your tests. Refactoring helps you deliver clean code to your client.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Steady

Feedback: You adhered to the red-green-refactor cycle for the most part, but did not refactor. There were opportunities to refactor that you missed. Unless they are design or architectural changes that would require changes in both your specification and implementation at the same time, I recommend only skipping the refactor phase if no refactorings are possible.

Your task prioritisation was great because you focused on the core cases before considering more specialised scenarios and on correctness before efficiency, providing immediate value to the client.

Your tests progressed in a stepwise manner, which was logical and well-suited for the problem's complexity.

You conducted research on how to test for the speed that a full second of music would take to filter using your band-pass filter. You could have considered using a timer or stopwatch library to help you time the method call and assert against the time taken rather than the result of the method. You did a great job in figuring out how to test for your method's processing speed.

I USE AN AGILE DEVELOPMENT PROCESS – Strong

Feedback: You asked for examples of the input and output soundwaves and used them as the basis for exploring the band pass filter's behaviour. Your attention to detail helped you discover finer details such as default and custom filters and edge cases the client hadn't thought of such as equal upper and lower limits. Your information-gathering was comprehensive as a result.

I WRITE CODE THAT IS EASY TO CHANGE – Steady

Feedback: You committed your code after every green phase, documenting your project's development history quite nicely. I encourage you to consider using capital letters to start your commit messages so that they are more noticeable when reviewing your commit history. It would complement your helpfully descriptive commit messages very well.

You only tested using the specified interface, decoupling your specification and implementation. You had opportunities to refactor that you should have considered taking up so that you could work with simpler code that was easier to change after refactoring.

You based your names on the client's vocabulary, which was an excellent way to name your variables, method and parameter. Your code's intention was clear as a result. Your test names were also descriptive of the behaviour they represented, clarifying your band pass filter's usage.

I CAN JUSTIFY THE WAY I WORK – Steady

Feedback: You vocalised your process, making it clear what you were doing. You provided valid reasons for the actions you took. An improvement to consider in the way of your process' reasoning is by justifying deviations such as skipping refactor phases. Overall, your workflow was logical. I also appreciated the updates on work progress you shared with the client.