REVIEW FEEDBACK

Peter Allen 25/03

25 March 2021 / 02:00 PM / Reviewer: Pierre Roodman

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

GENERAL FEEDBACK

Feedback: Your TDD process has really improved from the first review session. There are just small refinements that can be made such as refactoring on refactor phases and adding Git commits to your process as well. Well done.

I CAN TDD ANYTHING – Strong

Feedback: You based your tests on the acceptance criteria, which focused them on how the bandpass filter should behave with the different input variations based on the acceptance criteria thereby making them client-oriented. Good job. This also helps to ensure that tests are properly decoupled from the code.

Your test progression made great sense and each test broke the assumptions of the previous tests with the next simplest test case. This served to drive the implementation logic in an incremental manner that did not introduce too much complexity at any given time.

You have mostly followed the RGR cycle, but there were some refactor phases where there was an opportunity to refactor, but you decided to leave it for a later stage. I will discuss this further in the “I have a methodical approach to solving problems” section of the review.

I CAN PROGRAM FLUENTLY – Steady

Feedback: You were comfortable with using your editor and the terminal to set up, access and navigate your development environment. You were able to run RSpec and IRB well from the terminal.

You have a very good understanding of Ruby language constructs, syntax and built-in methods. The one thing that perhaps you could practise a bit with is the use of the map method.

You have made good choices on how to approach the problem and what data structures could get the job done as well as using default parameters and passing keyword arguments in your tests to check that they work as expected.

I CAN DEBUG ANYTHING – Strong

Feedback: You were able to debug your code really quickly when running into problems as you used tools that are at your disposal such as reading the backtrace properly and interpreting where the error occurred and why it occurred very well. You also printed out code to the terminal and used IRB to test that methods behave as you assumed that they would. You were therefore able to identify any mistakes that you made in writing your code and were quickly able to rectify them. There was no point where you were doing random changes to code in order to try and fix bugs.

I CAN MODEL ANYTHING – Steady

Feedback: Starting off with a single function in your algorithm was a simple place to start as it was not really necessary to create a class because this exercise does not require state in order to complete. This also allowed for extracting functions in order to adhere to the single-responsibility principle, which unfortunately you did not do although the opportunity did arise to do so.

You stuck to Ruby naming conventions having your class in UpperCamelCase and your method in snake_case. Your algorithm also made sense and you were able to complete the requirements quite well.

I CAN REFACTOR ANYTHING –Improving

Feedback: You were generally writing clean code, but there were opportunities to refactor on the refactor phases that you did not take advantage of and opted to do so later. One particular part of the algorithm that could have been refactored was the conditional logic that could have been extracted into a new method. This method could even have been used in a map method in order to manipulate the soundwave. You also made sure that when you introduced the default parameters, that your tests that passed limit arguments to the main method were adjusted accordingly.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Steady

Feedback: You have prioritised core cases over edge cases, thereby delivering immediate value to the client.

Your tests progressed in a logical order and at no point introduced too much complexity to the algorithm on any iteration of the RGR cycle.

You did not prioritise refactoring in your RGR cycle but opted to refactor at a later stage in order to get more test cases completed. I would encourage you to refactor when you identify it in order to avoid complex refactoring later which has an increased chance of introducing bugs.

I USE AN AGILE DEVELOPMENT PROCESS – Strong

Feedback: You did an outstanding job of gathering information and reifying the requirements and how the program should behave. You were able to clarify the finer details and also asked about some of the common edge cases for the datatypes being inputted and after a short while, you had all of the main requirements and the common edge cases recorded.

Your input-output table was also well-considered and you were able to base your tests on the recorded acceptance criteria and build a good test progression off of the table.

The one thing that you perhaps could have recorded, which I was hinting at when I said that a soundwave needs to have at least 1 frequency, was how to handle an empty array.

I WRITE CODE THAT IS EASY TO CHANGE – Improving

Feedback: You have not committed to Git on green and refactor phases. I suggest adding Git to your development process. Using Git to commit your work means that it is easy to roll back to an earlier version if something goes wrong thereby making your code easier to change. The commit messages also serve to document the changes made in terms of features completed. This will help a client to be able to keep track of the progress of the development of the program. You should commit on every green phase and refactor phase of the RGR cycle.

You named variables and your method so that it was clear what they represented and what tasks they took care of. This contributed to making your code readable, which in turn makes code easier to change. A small refinement here would be trying to have your variables reflect the client's domain more closely. An example is "freq_array" rather being named "soundwave".

You had your test suite properly decoupled from your implementation by making sure the tests were based solely on acceptance criteria, and not reliant on the current implementation. This makes changes to the code much easier as they will not break your test suite. This was very well done.

I CAN JUSTIFY THE WAY I WORK – Steady

Feedback: You are generally really vocal and kept me updated with your progress and your decisions. I would, however, encourage you to justify any reasons for deviating from the process such as skipping a refactor and ensure that your reasoning is sound.