REVIEW FEEDBACK

Iain Aitken 26/03

26 March 2021 / 09:00 AM / Reviewer: Joshua van Staden

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

GENERAL FEEDBACK

Feedback: Well done on getting through your second code review. It was great to see you adopting a simpler test progression that drives small and incremental improvements to your system. It was also great to see you asking about edge cases in the system. While you have improved from your last review, I feel that there is still some degree of improvement to be made. Firstly and most importantly, avoid testing edge cases before core functionality. Additionally, I would like to remind you to use Git as part of your process (once it becomes a habit, it will be much easier to remember). Otherwise, good job and I look forward to seeing future improvements.

I CAN TDD ANYTHING – Steady

Feedback: You were sufficiently familiar with the testing syntax and framework.

Your tests were based on the expected inputs and outputs of the system. This is great as it promotes flexibility in your code, while keeping it focused on the behaviour of the system.

The tests for core functionality began with a single value between the default values. This was a great place to begin, as you are able to use the default values to make test progression simpler. You then progressed outwards in a depth-first approach by adding complexity to an input case before moving on to the next input case.

It seems that, from your depth-first approach, some issues presented themselves in your test progression. This is due to the fact that you had a multiple-element array of values within the bounds passing before you moved onto a single element outside bounds passing. Moreover, you had this passing simply by returning the output. This could have been avoided through a refactor, or by simply inspecting the first element of the array. To avoid this in future, I would recommend following a breadth-first test progression. This involves covering all of the input cases before adding complexity to your code. In this case, you would first test an array of a single element within bounds. Your next test would test an array of a single element below bounds, then one above bounds. Then you can move onto multi-element arrays.

I CAN PROGRAM FLUENTLY – Steady

Feedback: You seemed comfortable enough using the terminal and editor to set up your environment. You also seemed sufficiently knowledgeable in basic programming concepts, such as variable declaration and method definition. You seemed to show a good knowledge of the in-built methods in Ruby, clarifying some syntax online when needed.

I CAN DEBUG ANYTHING – Nothing here

Feedback: No bugs seemed to present themselves during this review, so I was unable to get a sense of your debugging.

I CAN MODEL ANYTHING – Improving

Feedback: You modelled your solution as a single effective method encapsulated in a class. This is a fine choice of model, and provides plenty of room for expansion later on in the software’s life cycle.

In general, I would have advised against passing the track to the initializer. This is due to the fact that, in order to filter a new track, you would have to create a new object.

Your class name, Frequencies, while following Ruby convention, doesn’t precisely describe what it is modelling. Because the class’s purpose is to provide a Bandpass filter, a more appropriate name would be BandpassFilter. Your method name, filter, is sufficiently descriptive to illustrate its purpose.

Your algorithm followed a logical procession and made sense.

I CAN REFACTOR ANYTHING –Improving

Feedback: An opportunity for a refactor presented itself after the second core test went green. At this stage, your system was able to handle an array of multiple values as long as all of the values were between the upper and lower limits. This was achieved by simply returning the input array; however, this would not be the correct behaviour going forward. A good candidate for a refactor would have been to simply iterate through the input array and return all of the values unmodified.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Improving

Feedback: You regularly ran your test suite to observe your error messages changing.

You mostly followed a regular RGR cycle, however, there didn’t seem to be a step for checking for refactors in your process. In future reviews, it would be great to hear you sounding out a check for refactors. This gives you a reminder to refactor, and justifies any skipped refactors.

Your tests focused on passing edge cases before implementing the main functionality. Unfortunately, this does not provide immediate value to the user. Generally, the user would value core functionality being implemented into the system before edge cases. This is due to the fact that most use cases would be simple enough to be handled by the core functionality, so that code would be executed more than the code written for edge cases.

You changed previous test cases when you added in the lower and upper limit arguments. Because not specifying the lower and upper limits is a valid use case, I feel that this could have been avoided by specifying the values from the beginning of the implementation.

I USE AN AGILE DEVELOPMENT PROCESS – Steady

Feedback: You began your information-gathering by first clarifying the basic expected requirements of the system, ensuring that you have a good understanding of the inputs and outputs. Once you had a good understanding, you moved onto exploring the input space to find out how to better suit the solution to the client’s needs. This helped you to gather some finer details and edge cases within the system. This is great and helped you to better suit the client’s requirements.

While your Agile process is very much improved, I feel that there is one further refinement that can be made: using an input-output table to form a representation of the acceptance criteria. This has a quick and clear way of communicating your understanding with the client and clarifying any differences.

I WRITE CODE THAT IS EASY TO CHANGE – Improving

Feedback: I would have liked to see you use Git as part of your process. Using Git means that you keep a working history of your code. It also keeps a record of all of the changes made during development.

I was pleased to see that you had your test suite decoupled from the implementation. This was done by testing on the expected inputs and outputs of the system. This promotes flexibility in your code, and so any changes made to the structure of the code will not warrant a change in tests.

You wisely chose good variable names to reflect the information that they hold.

I CAN JUSTIFY THE WAY I WORK – Steady

Feedback: You sounded out your logic and process so that it was easy to follow you during this review.