REVIEW FEEDBACK

Sean Edwards 01/03

01 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: This was a really good second review session. You were able to form a really good algorithm and your process is quite refined and mostly sticks to good TDD principles. I suggest just being conscious of your first few tests taking on too much complexity at once as well as not refactoring on refactor phases when there are refactors that could be done in order to adhere to the single-responsibility principle.

I CAN TDD ANYTHING – Steady

Feedback: Your input-output table was serving you well as you were able to base your tests on the behaviours as was agreed upon with the client. Your tests were therefore client-oriented and were able to provide immediate value to the client as each test passed.

You took a good outside-in approach to the problem where you were introducing tests cases in an iterative manner in order to drive the logic in a naturally evolving manner. You were not focused so much on how the implementation logic would look at the end, but just focused on making the current test pass in the simplest way possible before introducing a new test that would iteratively scale up the complexity of the algorithm

I suggest though that your initial test be perhaps a bit simpler for example a single value array that tests either the lower and upper limit of the filter, then moving onto another simple case and identifying the duplication which would lead to the first refactoring of the code as the first tests did introduce a bit more complexity than perhaps you should have as you were working with creating an array, adding a loop and conditional statements all in one test. In a problem where you are not as familiar with how to solve the problem, this could lead to overengineering or the introduction of complex bugs. Simpler test cases, in the beginning, could help to mitigate that because you will only be focusing on a small piece of logic or transformation to the code as is necessary.

You did not refactor on all of the refactor phases, which can introduce complex refactoring later on and also allows for complex bugs to arise. I suggest refactoring on every RGR cycle, if a refactor is possible, in order to develop clean code throughout the process in a manner where the likelihood of complex bugs arising is less likely.

I CAN PROGRAM FLUENTLY – Strong

Feedback: You are very fluent with Ruby and RSpec syntax and language constructs as well as the use of the terminal. You were able to quickly pass tests without too many issues and solved problems quite easily developing a logical algorithm in the process. You were really familiar with default parameters in Ruby as well as many of the built-in methods that you were using.

I CAN DEBUG ANYTHING – Strong

Feedback: Debugging went very smoothly as you read the backtrace properly and you were never stuck at any point for too long as you were able to figure out quickly what the problems were. Great job.

I CAN MODEL ANYTHING – Strong

Feedback: You used a single method which was a great place to start as a class was not required due to state not being required to solve this exercise. You have followed Ruby naming conventions using snake_case for variables and method names. Your method name was also based on the client's domain and adhered to conventions by making it contain a verb that describes the action that was being taken i.e. filter.

Your algorithm made logical sense and you were able to finish almost all of the requirements. You could just finish off in your own time by testing if 44100 frequencies are processed in 100ms.

I CAN REFACTOR ANYTHING –Steady

Feedback: Although you did not refactor on every refactor phase, you did not have any magic numbers in your code or variable or function names that could have been changed. There was an opportunity to extract a method for the error handling that validates if the input is correct. Extracting this method would make your main method have less responsibility, thereby adhering to the single-responsibility principle.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Steady

Feedback: You have prioritised core cases over edge cases which has the result of bringing immediate value to the client.
You have for the most part followed a methodical approach to the problem sticking mostly to an RGR cycle. I do suggest not skipping refactor phases though as I have mentioned in the “I can TDD anything” section of the review.

You did very good research on how to raise exceptions and test for them in RSpec. Research is a very import aspect of a good development process as it is better to use your resources making sure how to do something than do random attempts at getting it to work.

I USE AN AGILE DEVELOPMENT PROCESS – Steady

Feedback: You did an excellent job getting clarity on the program’s main requirements and what a bandpass filter is meant to do. You asked relevant questions about whether the user sets the limits which brought a good understanding of how to approach the problem because you were well informed. You also touched on a common edge case for the data type which is an empty array. I suggest that you perhaps explore a few more edge cases such as negative integer frequencies, strings instead of integers etc.

You made a good input-output table that represented the main requirements quite well and you also added some of your own examples to base your tests on.

I WRITE CODE THAT IS EASY TO CHANGE – Strong

Feedback: You were committing regularly to Git at the end of the green and refactor step of your red-green-refactor cycle. This ensured that the latest working version of your code was available which helps with being able to change your code because if the program runs into problems, you can roll it back to a previous working version helping in making your code easier to change. This also means that your commit messages document the context of the changes made keeping the client updated with the progress of the program. You could look at using commit messages that are more client-oriented in order for the client to understand the progress of the program and what behaviours or features have been completed. An example of this would be the commit message: “Fourth and fifth test written and passed.” The message does not really cover the scope of what behaviour was completed. Perhaps you could detail what behaviour was added that is now working.

You chose sensible names for the methods which were informed by the client’s domain making it easy to understand what it does. This makes code easier to read and understand and subsequently also makes it easier to change because of that.

I was pleased that 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 test suites will not break due to changes in implementation logic or refactoring of the code.

I CAN JUSTIFY THE WAY I WORK – Steady

Feedback: You were very vocal about what you were doing and why you were doing it keeping me updated whenever you had completed a feature or requirement. This is really good practise for when you are in an interview environment. I would just suggest that if you deviate from the process such as not refactoring on refactor phases, that you justify your reason for doing so and ensure that your reasoning is sound as this will help to keep you to stick closer to the RGR cycle.