REVIEW FEEDBACK

Miranda Wilson 22/03

22 March 2021 / 09:00 AM / Reviewer: Pierre Roodman

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

GENERAL FEEDBACK

Feedback: You had a very well refined process in this review session. You were doing TDD in a very methodical manner with small increments of complexity on each RGR cycle. You are also very fluent with Ruby and RSpec and were able to solve all of the requirements for this program in a clean and elegant manner. You could start taking this process into the wild. Perhaps you could just attempt using Javascript in your future review sessions, just to give that some practice as well. Great job overall.

I CAN TDD ANYTHING – Strong

Feedback: You have followed a really good outside-in approach to TDD that tests for the behaviours of the program that you recorded in your input-output table and confirmed as being correct with the client. This ensured that your tests were also client-oriented and would provide value to the client with each test that passed. The logical progression of your tests was really well-considered and this drove the development of your algorithm really well in a natural manner.

You have followed the RGR cycle quite closely and were looking for refactoring opportunities on every refactor cycle.

I CAN PROGRAM FLUENTLY – Strong

Feedback: You seemed quite comfortable navigating your terminal and editor while setting up your environment for Ruby. You are well familiar with string manipulation methods and array methods and were able to use them correctly without any hassle.

You developed an algorithm that took care of all of the requirements of this exercise in an efficient and clean manner whilst making excellent choices on how to approach and solve the problem.

You had first attempted to do this in Javascript but had trouble getting the spec runner setup. Perhaps in the next review session, you could have this setup with your own test file and javascript file in order to make sure that you have the spec runner working correctly.

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 using puts statements and used IRB to test methods in order to check critical assumptions and were, therefore, able to identify any mistakes that you made in writing your code and were quickly able to rectify them.
Because you follow a methodical approach with a good TDD process, at no point did you have to do any complex debugging which is one of the major advantages of a good TDD process.

I CAN MODEL ANYTHING – Strong

Feedback: You modelled your solution with a single method responsible for the spell-checking. A method without a class was sufficient as state is not really necessary to complete this exercise. By starting with a single method, you left your algorithm open to adding methods later when refactoring in order to keep your main method adhering to the single-responsibility principle.

Your method names adhered to the Ruby naming convention of snake_case and were also actionable names that describe what the methods do in a way that reflects the client's domain.

The algorithm that you completed made logical sense and was able to adhere to the requirements as were provided by the client.

I CAN REFACTOR ANYTHING –Strong

Feedback: You have refactored your code in a way that makes the code really clean and readable without any code smells. You have cleaned up if-else blocks with elegant ternary operators and you were also able to identify methods that could be extracted in order to adhere to the single-responsibility principle. Great job.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Strong

Feedback: You have prioritised core cases over edge cases and this provided immediate value to the client.

Your tests progressed in a logical order and you were able to build up the logic in small increments not taking on too much complexity at any given point in time. This helped you to stay focused on the task at hand. You have really refined this process quite well and are making well-considered decisions for your test progression.

You are following the RGR cycle really closely which led to the development of clean and bug-free code.

You have also conducted good research that uncovered a really elegant way of handling punctuation using the chomp method.

I USE AN AGILE DEVELOPMENT PROCESS – Strong

Feedback: You did a great job gathering information about how the spell checker should work clearing up all of the finer details. You also made excellent use of an input-output table with your own examples in order to flesh out the behaviours a bit starting from the simplest cases to the more complex cases. This served you very well when you started creating your tests as you could just copy and paste these inputs and outputs into your tests thereby encoding the requirements as confirmed with the client into the tests. This ensured that your algorithm would adhere to all of the acceptance criteria. You also asked many questions on how the input might vary as you probed many of the edge cases.

I WRITE CODE THAT IS EASY TO CHANGE – Steady

Feedback: You did regular git commits on green and refactor phases in order to ensure that you had the latest working version available to you. This meant that if you made a change to the algorithm that broke the program that you were unable to undo, you could just roll back to a previous working version thereby helping increase the changeability of the code. Your commit messages also explain exactly what the scope of the work was that was covered in terms of the behaviours that were being tested which makes it easy for a client to see the progress of the program by reading the commit messages.

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. There is just one refinement that I suggest looking at which is to make your variable names reflect the client's domain a bit more closely. An example is "words" perhaps being sentence because that is what the client the input as being and just makes a bit more sense in terms of describing the data stored in the variable.

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 were generally quite vocal about your process, what you were doing and why you were doing it. There was an occasion where you made a decision about what test case you were going to introduce next without really explaining why you wanted to introduce that specific test (“a a”). In an interview type of environment, it is good to justify such decisions to the interviewer so that they can get some insight into how you make decisions and if those decisions are well-considered.