REVIEW FEEDBACK

Iain Aitken 02/04

02 April 2021 / 10:00 AM / Reviewer: Pierre Roodman

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

GENERAL FEEDBACK

Feedback: There are some real improvements that you are making in terms of test progression as well as adding Git commits to your process. You taking the time to consider what your next step is going to be. There are, however, times where you take some time contemplating your next step even after voicing out valid approaches. Being confident in your decision making often is a result of having a bit more experience. I would suggest that you practice your TDD process a bit more before booking your next session in order to build up that confidence in your decision making.

I CAN TDD ANYTHING – Steady

Feedback: You have maintained the good habit of testing for behaviours in order to avoid tight-coupling of tests and code, as well as making the tests client-oriented.

Your test progression is improving and you are now making well-considered decisions as to what your next test is going to be and how that affects the transformation of the algorithm that you are creating, if it is simple enough or if there is too much new complexity introduced. There are times when you are caught up between deciding between different approaches, whether to go depth-first or breadth-first such as whether to introduce an incorrectly spelt word after the correctly spelt word or whether to use two correctly spelt words. I would suggest that you just decide on one and go with it. So long as you are doing so in an iterative manner, both are valid approaches. If you find that the approach is not working for you, you can always roll back to the test where you made the decision and then take another approach. I also

encourage you to perhaps practice doing some exercises from CodeWars, but to do them in your own IDE and TDD them. The more you practice the TDD process, the more confident you will become in the decisions that you make.

You generally stick quite closely to the RGR cycle, but you do not always refactor on a refactor phase. I would suggest that you try to identify any methods that may be extracted such as the logic that checks if a word is spelt correctly or not.

I CAN PROGRAM FLUENTLY – Steady

Feedback: You are quite familiar with the terminal and Ruby and RSpec syntax. You are also quite familiar with built-in methods for array and string manipulation and your algorithm was making logical sense. You did have a bit of trouble with the usage of the map method though but were eventually able to identify what you were doing wrong after some debugging. I recommend practising a bit with array manipulation and even looking into how to use a function on each element of the array in the map method's block.

Your algorithm was making sense and you were well on your way to fulfilling the requirements.

I CAN DEBUG ANYTHING – Steady

Feedback: You have a very good debugging process. You are reading the backtrace really well and you are able to interpret the reason for the error and where the error is occurring. You also use puts statements in order to get visibility into the code. I would suggest that you try to do some more research when you are faced with a problem that you are sure you are doing correctly. Make sure that the way you are using the methods that you are using is correct and look at the variations of using the built-in methods by looking at the documentation. Looking into the documentation for Ruby would show that there are two ways to use the map method (map or map!). Using it with the exclamation mark changes the existing array and without the exclamation mark returns a new array (therefore the need to assign it to a variable.) https://docs.ruby-lang.org/en/3.0.0/Array.html#method-i-map

I CAN MODEL ANYTHING – Strong

Feedback: You were able to justify your usage of a class for this exercise by anticipating that at some point the client would be providing their own dictionary and that you would then be able to expand the logic in order to accommodate that by using a class. You have started with a single method that returns the manipulated string which is a simple enough place to start.

Your class and method names were in adherence to Ruby conventions by using UpperCamelCase and snake case respectively.

Your algorithm was making logical sense and you were able to pass many of the requirements by the end of the review session.

I CAN REFACTOR ANYTHING –Improving

Feedback: You wanted to move on to new test cases when your tests went green and whilst you were taking some time to consider the next test case, I reminded you that there was hard coding that needed to be generalised first before moving onto the next test case. I suggest that you look at any duplications in your code first before moving onto the next test case so that you have generic code for the simple cases before moving onto more complex where you will then need to focus on two levels of complexity instead of one.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Improving

Feedback: You have now started testing for core cases before edge cases which allows you to deliver immediate value to the client.

Your tests are also progressing in a logical order and do not introduce too much complexity to focus on at any given point in time.

You do tend to focus a bit on what the next test is going to be before refactoring the code that has already been written. This means that you could still work on sticking to an RGR cycle a bit more closely. I recommend that

you have some sort of reminder such as a sticky note on your monitor. The note could perhaps list out what is expected on each phase of the RGR cycle until it becomes second-nature to you.
You are doing some research, but I also recommend using official documentation when conducting research in order to make sure that you have the latest information about the language you are using and the version you are working with.

I USE AN AGILE DEVELOPMENT PROCESS – Steady

Feedback: You have reified the main requirements well with the client and understood the behaviour of the program quite well. You also took note of the requirements as the client provided them. I suggest that some of the variations of the input be ironed out during the information gathering stage a bit more such as proper nouns, abbreviations etc. You did, however, ask about some of the edge cases that could arise and clarified the information about the dictionary.

I would also encourage you to make use of the input-output table to create examples of the inputs and outputs ranging from the most simple cases to the most complex. This can help to determine what your test progression could look like whilst at the same time you are clarifying with the client what the outputs would look like for those cases. One could simply copy and paste the inputs and outputs into your tests as well.

I WRITE CODE THAT IS EASY TO CHANGE – Strong

Feedback: You have added Git to your development process and are now committing after green and refactor phases. This means that you are now increasing the flexibility of changing your code because you can roll back to a previous working version if something goes wrong. Your commit messages are quite clear and a client would be able to read the messages and follow the progress of your program. Just bear in mind that the convention for Git messages is that they start with a capitalised word.

You have maintained the good habit of testing for behaviours in order to ensure that your code and tests are decoupled.

Your variable names and method names are still descriptive and they also accurately reflect the client's domain by using the terms that the client has used.

I CAN JUSTIFY THE WAY I WORK – Steady

Feedback: You are quite vocal with your process and you are communicating your thought process well. When you do so, you give insight into how you consider what to do next and that you are not just making decisions without contemplating the implications of the decisions. You were also giving me updates as to what progress you were making. I would suggest that you justify any deviations from the process such as skipping a refactor phase. This also helps you to adhere to the process more strictly because you need to evaluate your reasoning as well.