REVIEW FEEDBACK

Ben Gittins 24/08

24 August 2020 / 10:00 AM / Reviewer: Katherine James

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

GENERAL FEEDBACK

Feedback: Great review Ben. You made use of good test progression, were aware of the complexity that tests demanded and justified the decisions that you made excellently. Although slightly inclined to refactor the code a bit on the green step of your red-green-refactor cycle, you were aware of when this had happened. You are conscious of your process and demonstrated good ability today, handling the review well. This is a good place to be at. Really well done and all the best for the interview tomorrow!

I CAN TDD ANYTHING – Steady

Feedback: You were familiar with the testing syntax and used clear test descriptions for your it-blocks.

Testing approach: Your tests were based on the acceptance criteria (on the expected input and output of the system). This is great as you can change how your implementation works, without breaking the tests. This promotes flexibility in your implementation, which is good from an agile point of view.

Test progression: You used good, iterative testing in today's review in which resulted in you built up some basic logic for distinguishing between the two input cases (correctly and incorrectly spelt) first, before moving onto applying this logic to longer inputs. This was excellent, as it meant that you effectively created the logic for 'check a single word' before applying this logic to each element in the array. It was really good to see that you caught yourself when the code was getting a bit too complex and that you stepped back, refactored and then stepped forwards again.

You then moved onto a test with more words to drive the introduction of a word bank to check against. This was good. For interest, note that this functionality could also have been added in earlier if you had tested for another correctly/incorrectly spelt word pair before incrementing the complexity to multiple words. Your decision to introduce a test to handle capitalisation after the dictionary had been established was excellent.

I CAN PROGRAM FLUENTLY – Steady

Feedback: You were nicely familiar with navigating your programming environment and went through the setup process fluently. You had a good handle on Ruby and its in-built methods, demonstrating familiarity with a number of these during the session (split, downcase, split, include?, concat and join), all of which you were familiar with the syntax of or able to verify quickly using the stack trace information if not 100% correct. You were familiar with working with arrays using an each loop and building up a new array as output and converting this to a string using join. You consistently were able to smoothly meet the goal set by the new failing test in code.

I CAN DEBUG ANYTHING – Steady

Feedback: You seem to have a good debugging process. You seem to understand where to find the most important information in the stack trace, the line number where the error occurs and the error message itself. You seem to have an understanding of the most common error messages and what they mean and also used prints to get visibility into your code.

I CAN MODEL ANYTHING – Steady

Feedback: You initially modelled your solution after a single method encapsulated in a class. This was a good place to start, as it was sufficiently simple, yet also provided a great foundation for expanding complexity later on in the software's life-cycle, which you utilized in your refactors at the end of the session. You included a constructor for creating

the instance variable containing the populated array of correctly spelt words and extracted methods from your main method to conform to the single responsibility principle. This was a really good design for this task. You named your class SpellChecker and your method spellcheck. These were both suitable choices.

I CAN REFACTOR ANYTHING –Steady

Feedback: You seemed to have a slight tendency to refactor on the green step of your red-green-refactor (RGR) cycle, but you were aware on this both times that this happened in the review. The first instance of this was jumping straight to string interpolation, the second came about in refactoring the algorithm on the test with multiple words (where you caught yourself and stepped back away from this). Other than this, you stuck to your RGR cycle nicely and executed a number of excellent refactors at the end of the review for readability and the single responsibility principle, extracting a method which could be extended for parsing the input from a string into checkable words.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Steady

Feedback: I found your process to overall be very methodical today. I really liked how you compared the notes to the tests a the end, how you justified each decision you made and how you ran a solid RGR cycle. Your tests progressed in complexity iteratively, as did your code.

I USE AN AGILE DEVELOPMENT PROCESS – Steady

Feedback: You conducted an excellent information gathering session, in which you reified the main requirements and confirmed these using examples in an input-output (IO) table. You asked about output type and formatting requirements and gave and overview of the main requirements. You had a good grasp of what sort of finer details might be relevant to this task and asked about these to establish how these should be handled, such as an empty string, capitalisation and punctuation. You asked if there was a particular structure

required for storing correctly spelt words and made a good choice for this. You also gave a final run overview of the requirements, which was excellent to confirm that the client and yourself were on the same page.

I WRITE CODE THAT IS EASY TO CHANGE – Steady

Feedback: Your code was nice and easy to change. 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. You also made a good choice selecting an array for the word bank, as new words are very easily added to this. You committed to Git whenever your tests passed and supplied a descriptive commit message to document your changes, such that these messages documented the changes in your code. You also committed after refactors, describing the reason for the change. You chose good variable names such that it was clear what they represented. This makes the code easy to read and thus change.

I CAN JUSTIFY THE WAY I WORK – Strong

Feedback: You vocalised your process so it was always clear what you were doing and why, pitched your comments at a good level and provided sound reasoning for each decision you made. There were a couple of really excellent justifications you made today, beyond justifying the changes you made to your code and continuously updating me as to what you were working on. These included justifying your choice of using a class, adding each new test case in when you did, selecting the array to store correctly spelt words and advantages and disadvantages thereof. You also gave good insights into your understanding of single responsibility, by highlighting what the responsibilities of the method were and also that you were aware of the level of complexity each test should drive.