REVIEW FEEDBACK

David McGregor 23/04

23 April 2020 / 08:00 AM / Reviewer:

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

GENERAL FEEDBACK

Feedback: Overall really nice, this was the most complete solution I have ever seen in a review, so you are to be commended on that. There were one or two points where refactored on a green step (ie didn't do the simplest thing to make the test pass, instead, building a refactored solution immediately), but other than this you process is really strong.

I CAN TDD ANYTHING – Steady

Feedback: You ran RSpec nice and regularly as part of your red-green-refactor cycle and to see whether your implementation solved the test. It was great that your tests were based on overall behaviours (acceptance criteria) of the system. Your test progression was excellent, based on iteratively increasing cases in terms of behaviour, starting with the simplest cases and iteratively increasing the complexity to more complex cases. You were very conscious of your red-green-refactor cycle and made (most) of the tests pass in the easiest way possible.

There were a couple of instances where you combined your red and green steps, thus not making the test pass in the easiest way possible. The most notable of these was when you opted to download a long list of English words in a text file from the internet to use as your dictionary. This was not the easiest thing you could do to get your test to pass, so this should have been done on a refactor step. This would have been a really excellent refactor. You could have simply added the necessary additional words to your array to get the test to pass simply. I will discuss other cases of this in the 'refactor' comments section, as this is the proper place for them.

Comments on punctuation approach:
You got this to pass by simply adding “mat.” to your dictionary, this was a very simple way to get the test to pass on a green step, which was good. You introduced another test to generalise against before finishing your refactor.

I CAN PROGRAM FLUENTLY – Strong

Feedback: You were very comfortable with Ruby syntax for if statements, the use of map and a number of in-built functions. When unfamiliar with the syntax for any in-built methods you checked useful sites like StackOverflow. You were comfortable with using an external text file containing many English words as a more complete dictionary - this was great to see. You dealt with checking for incorrect types really efficiently - you were clearly very familiar with this step, and you were making progress towards dealing with punctuation, there were some promising approaches that you tried.

I CAN DEBUG ANYTHING – Steady

Feedback:

I CAN MODEL ANYTHING – Steady

Feedback: You modelled your solution after a single method encapsulated in a class, this was a simple enough place to start and also provided a great foundation for expanding complexity later on in the software's life-cycle. You followed naming convention best-practices and named your class as a noun and your method as a verb.

I CAN REFACTOR ANYTHING –Improving

Feedback: You did several really good refactors as part of your process, namely, refactoring away from hard-coded input, generalising the return formatting of incorrectly spelt examples and the introduction of an array of correctly spelt words to check against. The introduction of the external

dictionary file was a really good idea and was a really good refactor, it just needed to be done on the refactor step of your cycle.

One refactor I would have liked to see (after the test before your “Good cat” example), was swapping the check to be for correct words, rather than incorrect words, in the if-statements. I also would have liked to have seen an earlier refactor to have a single dictionary with all words downcased in it, before the introduction of the external dictionary.

After getting the test to pass by adding “mat.” to your dictionary, you then refactored your code to make use of .scan with a regular expression to deal with the punctuation in a better way. You forgot to run rspec after introducing this, and instead opted to introduce a new test with an exclamation mark. In terms of your red-green-refactor cycle, you needed to get this test back to green before introducing another test.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Strong

Feedback: You have a nice methodical approach to your development. You moved from requirements gathering to developing a good set of inputs and outputs that drove your testing. You clearly put a lot of thought into this step. Your readme was extremely well laid out, with very clearly demarked sections. You made use of excellent test progression, including leaving edge cases for last. You made good checks of program operation using ‘puts’ and used two panes for easy reference between your code and tests.

I USE AN AGILE DEVELOPMENT PROCESS – Strong

Feedback: You conducted a really strong information gathering session. You immediately asked for an example of the input and output, which allowed you to determine the general program operation. You asked an excellent question about whether there was a choice of a dictionary to use, which established that you only needed a working set of words. You also considered a good real-world consideration, in terms of asking about capitalisation. You also considered non-string input and asked how to account for that. Other edge cases you could

have asked about were about punctuation, but the client ended up suggesting this near the end since you conquered the task so quickly. You created a really excellent IO table, in which you listed examples which started out with the simplest of cases and became iteratively more complex - which formed a nice basis for your tests.

I WRITE CODE THAT IS EASY TO CHANGE – Strong

Feedback: I was very happy with your Git usage. You initialised Git early on, committed after every test went green and supplied useful commit messages, making it easy for future developers to come in and change a feature.

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.

Your code was also laid out neatly, and documented in a neat readme.

I CAN JUSTIFY THE WAY I WORK – Strong

Feedback: You let me know when you expected errors and why errors were occurring. Your process was easy to follow. Really nice thoughts about possible approaches, and justifications for reverting to a previous approach. Your comments were pitched at exactly the right level.