REVIEW FEEDBACK

Peter Allen 16/04

16 April 2021 / 11:00 AM / Reviewer: Ronald Munodawafa

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

GENERAL FEEDBACK

Feedback: You have improved your code changeability! The areas you could improve on include refactoring, leading your green phases with your red phases and test variation. I believe that practice will help you improve on these aspects. I wish you the best!

I CAN TDD ANYTHING – Improving

Feedback: You derived your tests from your input-output table, keeping your tests client-oriented and behaviour-based.

Your first tests were incredibly simple, starting with a single word and proceeding to multiple words. These tests focused on correctly spelt words. While your tests were simple, there's nothing that these tests imply about correctly spelt words in the general case given their similarity. Someone else using your tests could have ended up with a method that only recognises the word 'a'. By varying your test cases, you are inducing generality in your code. You can do this via a technique known as test triangulation, which you can learn about here https://www.tddfellow.com/blog/2016/08/31/getting-stuck-while-doing-tdd-part-3-triangulation-to-the-rescue/

I CAN PROGRAM FLUENTLY – Steady

Feedback: You were comfortable with including the terminal in your development workflow, making excellent use of RSepc’s test runner, Git and the Unix utilities.

You were familiar with Ruby's language features, needing to occasionally look up features such as string interpolation. Your program was logically flowed, showing the ability to develop algorithms and process strings and arrays using the corresponding classes.

I recommend continuing regular practice with Ruby to keep the language at your fingertips.

I CAN DEBUG ANYTHING – Improving

Feedback: You read the backtrace messages of your failing tests. You paid attention the expected vs got output and used them to justify the changes you made to your method. You also interpreted the reasons for the failure correctly.

I recommend including print statements and IRB as part of a feedback loop that reduces your bug search space and tests your hypotheses concerning your code. It would help you avoid dealing with too much uncertainty that gives way to random bug fixes. This may have helped you pass your first four tests quicker.

I CAN MODEL ANYTHING – Steady

Feedback: Your solution was a method, which was suitably minimal for the problem’s statelessness.

You named your method ‘spellcheck’, adhering to Ruby’s convention of snake_casing method names and object-oriented programming’s best practice of assigning actionable names to methods.

Your algorithm was sensible.

I CAN REFACTOR ANYTHING –Improving

Feedback: You refactored by changing your algorithm in a manner that did not simplify your program necessarily. You mentioned that this was due to needing to refactor. I recommend applying the single-responsibility principle when you can to improve your code's changeability.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Improving

Feedback: You followed a regular red-green-refactor cycle, leading some of your green phases with your red phases. To help you tighten your process around this, I recommend only writing as much production code as is necessary to pass your test. You can employ test triangulation to help you select tests that generalise your code.

You used your tests as a measure of progress, running them on every change you made. Your refactor phases could have included more refactorings to clean up your code in iterations. Your tests also proceeded logically.

You prioritised the core cases before the edge cases. As a result, you provided immediate value to the client.

I USE AN AGILE DEVELOPMENT PROCESS – Steady

Feedback: You took note of the information the client shared with you meticulously. You asked questions surrounding the inputs. You asked about the input's format and the representation of the custom dictionary. You asked about edge cases such as the empty string. You asked for examples of input and output. You constructed an input-output table, planning your tests with the client's input. Your information-gathering was comprehensive.

I WRITE CODE THAT IS EASY TO CHANGE – Steady

Feedback: You committed after most green phases, providing reference points for reverting undesirable changes. You committed some green phase code along with refactorings. I recommend committing after every green phase and again after every refactoring to keep track of all your working states.
Your commit messages could have only described the implementation work that you'd done to clarify the value you provided in that commit. Starting commits with a capital letter would also make them more noticeable.

You only tested using the specified inputs and outputs, decoupling your tests and code. You had the opportunity to refactor because of the flexibility afforded by a decoupled architecture.

You made sensible choices for the name in your program, making it clear what they represented.

I CAN JUSTIFY THE WAY I WORK – Improving

Feedback: You explained the steps you took and justified the deviations you made. This highlighted the reasoning behind your workflow. I recommend also providing updates on the completion of work. It would help the client keep track with your progress.