REVIEW FEEDBACK

Jason Wong 05/03

05 March 2020 / 11:00 AM / Reviewer:

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

GENERAL FEEDBACK

Feedback: Thanks for coming in today. Overall this was a typical first session, so don't be too discouraged. I think there's still some misunderstanding around how to use tests to drive your implementation. This, along with basing your tests on acceptance criteria and not intermediate steps, would be a good thing to focus on for next time. I look forward to our next session and hope my comments have been helpful.

I CAN TDD ANYTHING – Improving

Feedback: You weren't entirely familiar with the testing syntax, it may be a good idea to have a template if you will be using this syntax often. It would be better if each 'expect' were within its own 'it' block so we have a better idea of which test is failing.

Your testing approach could use some refinement. The outputs you were checking against in the first two tests were arrays, rather than strings. By testing the outputs of methods, you weren't testing the actual acceptance criteria, and this is called a 'structure first' approach. It is important to always test against the acceptance criteria so that your tests are not dependant on implementation details. In this regard, tests should all have only made use of the 'report' method, and you could have created the evaluate method as part of a refactor later on if it was necessary.

Your next test was a case with "Green, Amber". I would say that this was a little complex to start with, as it is important to deal with all the

simplistic cases first. This case pushed you to introduce iteration very early into the program.

You then removed your first test, I'm not quite sure why. If you implement a test, it is important to keep it as it should be there to test whatever feature it was put there to test and to remain in place to check that this remains true. If you are going to remove a test, or modify it, it probably wasn't based on acceptance criteria and thus should not be there in the first place.

I like to recommend this video, it gives a nice insight into a good TDD process as is worth a watch: https://www.youtube.com/watch?v=B93QezwTQpI.

I CAN PROGRAM FLUENTLY – Steady

Feedback: You seemed quite comfortable navigating your terminal and editor while setting up your environment. You seemed pretty comfortable with Ruby, and clearly have a good grasp of core programming concepts such as object-orientated programming, variables and if statements. You were also familiar with in-built methods such as split and string interpolation.

I CAN DEBUG ANYTHING – Nothing here

Feedback:

I CAN MODEL ANYTHING – Improving

Feedback: You modelled your solution as two methods. I would think that the 'evaluate' method was not really necessary at the stage of program we were working at and would suggest you have waited until a refactor necessitated a second method later.

I CAN REFACTOR ANYTHING –Improving

Feedback: You refactored on the Green step of the Red-Green-Refactor cycle. After introduced a test for the“Red, something” case, you immediately began to develop a solid (refactored) solution to the currently failing test instead of finding the quickest solution during the Green step and then using the Refactoring step to solidify the solution. This sort of methodology tends to produce over-engineered solutions and a fair amount of debugging.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Improving

Feedback: Your testing seemed a little ad-hoc, rather than methodical. I would have preferred to see you developing your tests from simple cases and iteratively moving to more complex cases. This is important so that there are no gaps in your tests.

You dealt with the ‘uncounted’ edge case before the full program was developed. Edge cases do not provide value to the user until core cases have been addressed.

I USE AN AGILE DEVELOPMENT PROCESS – Improving

Feedback: Your information gathering process was sufficiently good for you to grasp the main acceptance criteria. You asked some questions about the input what the input and output, about formatting and about types to ensure you were on the same page as the user. You tended to clarify points about the input and output with verbally with examples which was nice as the client could provide information as to how to adjust from this point to their desired requirement.

You asked good questions about edge cases and established which edge cases could be dealt with in the same manner. You were able to use the provided example to draw observations as to how the input might vary, which you clarified with the user.

I would like you to pay particular attention to establishing the type of the input and output during the information gathering, as this is very important to both the client and to how you develop a solution.

It would be nice to see the information that you've gathered in the README as you gather it. I'm not sure if this was due to a screen-share issue or if you recorded your notes elsewhere. This is a nice step as it really ensures that you and the client are on the same page from the word go.

It may be nice to make a table of inputs and outputs to help with your testing in the future. In this, you should include simple cases and more complex cases that allow you to see the features of the program. This can often help lead a discussion about some of the finer (less apparent) details as well as provide a good basis for your tests.

I WRITE CODE THAT IS EASY TO CHANGE – Improving

Feedback: You initalised git before starting, but then didn't commit to git after each test went green. Ideally, you would commit on every green test, before the refactor, as part of a Red-Green-Refactor cycle. This creates a paper trail for future developers to use if they wish to modify a feature in your system. It also means that if something goes wrong with your code, there is a working version to fall back to.

Your first two tests were too tightly coupled to your implementation. This creates problems when making changes to the implementation as often times you will have to change your tests along with the implementation. Try to decouple your tests by basing them on overall behaviours (or acceptance criteria) rather than basing them on structure(or implementation details). Eg. If we wanted to change how the evaluate method worked, we could not do so during a refactor because it would break these tests, whereas if the tests were based on acceptance criteria, from an abstract point of view we don't really care about what steps were taken to get the output as long as the output was achieved.

I CAN JUSTIFY THE WAY I WORK – Improving

Feedback: You briefly updated me as to your achievements and what you were going to do next. Slightly more vocalisation would be nice to help me follow your process. Eg. ‘The test is now written and it fails because there is no method, so now I’m going to go create that’. This would allow me to understand your decision-making process a bit better. This can be very important especially in the technical interview space where interviewers are not only assessing your coding but your reasoning behind your coding as well.