REVIEW FEEDBACK

Amit Tauro 06/01

06 January 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 a very steady development process, which appears to be maturing quite well! Well done on the improvements! To continue refining your development methodology, I encourage you to systemise how you evaluate design strategies. Other than that, you are doing very well. Keep it up!

I CAN TDD ANYTHING – Steady

Feedback: You ran your tests to ensure they failed and used the failing tests as a basis for development, forming an incremental red-green-refactor cycle. You focused on testing from the client's perspective by only testing for behaviours with no assumptions about the implementation's logic. You also based your tests on the acceptance criteria. You met the client's needs quickly as a result.

I CAN PROGRAM FLUENTLY – Strong

Feedback: You comfortably navigated your development environment using your editor and terminal. You used Ruby's language features and standard library to process strings and arrays and compose your RSpec tests. Your competence as a programmer enabled you to provide a logically flowing implementation of your algorithm. I also appreciate your use of 'pry' for debugging; it was a fantastic inclusion in your toolbox.

I CAN DEBUG ANYTHING – Strong

Feedback: You read the failing tests' messages and used them as the basis for determining what minimal changes were needed to make your tests pass. You also expressed hypotheses concerning the expected behaviour before running your tests, demonstrating that you maintained a keen notion of your program's correctness.

You also used 'pry' to make your program's runtime state visible and form a feedback loop. It was an excellent way of testing your hypotheses!

I CAN MODEL ANYTHING – Steady

Feedback: You modelled your system as a class with a single method client code could call to evaluate the sum string. Using a class was appropriate as it allowed room for extensibility. Your class and method names followed Ruby's naming conventions of classes being PascalCased and methods being snake_cased.

Your algorithm to the problem was elegant.

I CAN REFACTOR ANYTHING –Steady

Feedback: I appreciate the refactors you performed during the session. You added contexts to your tests to make them maintainable. You also applied the single-responsibility principle through the introduction of private methods. You could have avoided your replacement of these private methods with the constructor taking on the methods' responsibilities since it increased your solution's complexity. You might find it helpful to evaluate the quality of your refactor in terms of cohesion (which you want to maximise), complexity (which you want to minimise) and coupling (which you also want to minimise). Doing so gives you a good starting measure for considering whether a refactor is necessary.

Last but not least, I observed that you completed your algorithm, you spent considerable time grappling with the implementation complexity. You could curb this by refactoring in each refactor phase after every green step; it helps

you proceed with clean code. Overall though, you demonstrated that you have good refactoring skills.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Strong

Feedback: You dealt with core cases before handling the edge cases, providing immediate value to the client. You ordered your tests in increasing complexity and grouped them by behavioural commonalities using contexts. Your red-green-refactor cycle was regular and iterative. As a result, your problem-solving is systematic. I also appreciated the care taken to observe finer details within your development process. I encourage you to continue refining your development toolbox.

I USE AN AGILE DEVELOPMENT PROCESS – Steady

Feedback: You asked about the inputs and generated an example to uncover the product's general behaviour, including finer details such as the spacing between operators and operands. You listed examples in terms of inputs and outputs, allowing the client to chime in when necessary. It also provided a sound basis for your tests. Your requirements-gathering was comprehensive.

I WRITE CODE THAT IS EASY TO CHANGE – Steady

Feedback: You committed almost every green and refactored version of your solution. The exception was the first green phase which you did not commit. Committing every working version helps you document your project's history and provide a reference point if necessary to revert to a particular commit. You could improve your commits' messages by not mentioning what tests are passing but rather what behaviour the commit is introducing or changing.

While using the attribute reader for debugging, I hope you consider excluding it when giving the client the complete product to ensure that there is no temptation for client code to become tightly coupled with your library. It helps avoid breaking client code if you change your implementation.

On that note, you kept your tests and implementation decoupled from each other, allowing refactors to take place without losing your tests' validity.

Finally, you chose program entity and test names derived from the requirements, making the intention and usage of your code easily discoverable.

I CAN JUSTIFY THE WAY I WORK – Steady

Feedback: You verbalised your thought process with clarity and explained why you took each action, making it easy to follow your workflow. For a non-technical client, you might want to update them on your work's progress in terms of fulfilling the requirements you gathered from them. It helps them understand your process in terms of the scope of work from their perspective. The client is central to agile development. That said, you were clear enough and easy to follow.