REVIEW FEEDBACK

Lucy Stringer 17/12

17 December 2020 / 09:00 AM / Reviewer: Ronald Munodawafa

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

GENERAL FEEDBACK

Feedback: You currently have a good basis for forming a solid development methodology. You have a sense of test-first development. The process could be refined further by being more iterative and incremental. Taking more time to analyse the requirements the client has given you would also help inform the development process better. That said, I was happy to see the direct use of the example the client provided. It showed an explicit focus on the client's needs. Well done!

I CAN TDD ANYTHING – Improving

Feedback: The first green phase attempted to split the string resulting in a lot of time taken when the green phase should aim to pass as quickly and simply as possible. Hard coding the return value would achieve this aim. Introducing some control flow with the hard coding would allow for simpler progressions. The refactor phases would then be used for eliminating any duplicated code based on concrete patterns appearing in the code. Following this cycle would establish a steady red-green-refactor cycle.

All the tests except the second were based on the acceptance criteria. Those tests helped allocate development time efficiently for the client's needs. However, the second test was focused on intermediate steps removed from the specified behaviour taking up development time that could be used for the client's needs. That said, the other tests were behaviour-based and client-oriented.

I CAN PROGRAM FLUENTLY – Steady

Feedback: The use of the terminal and editor allowed for easy use of the development toolbox. The use of Ruby's control flow features led to correctly composed tests and a logically flowing algorithm. Some assistance was needed in terms of verifying the behaviour of String.split. The employment of a few debugging techniques helped bring light to this. However, some practice on a code challenge platform would help with the familiarity with Ruby's standard library. The final product of the development session made logical sense.

I CAN DEBUG ANYTHING – Steady

Feedback: Using puts to view the state of the grade counts during runtime was an excellent way to verify the expectations surrounding the implementation's behaviour. Using of the REPL was also good as it allowed for the debugging outside of the runtime environment. Using the REPL for experimentation with unfamiliar methods would help resolve uncertainty much quicker, e.g., when dealing with String.split.

Due to a stable debugging process, the bug with the splits of the grades string was resolved successfully and the development of the product continued.

I CAN MODEL ANYTHING – Steady

Feedback: The system was modelled as a class with a method that would perform the grade counting. The class and method names adhered to Ruby's naming convention of classes being PascalCased and methods being snake_cased.

The string splitting, grade counting and string interpolation were a natural approach to solving the problem leading to an elegant algorithm.

I CAN REFACTOR ANYTHING –Improving

Feedback: The refactor phases were not made use of. The change in the model from a method to a class would have been better suited for the refactor phase if the reason would not tighten the coupling between tests and implementation. Refactoring in each development iteration improves the quality of the codebase at any point in time during the process.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Improving

Feedback: The overall development approach was certainly test-first. However, it could be improved by breaking down the behaviour to be tested for and making use of more examples such that making the tests pass as simply as possible would suffice for complete development. This would make development more incremental and iterative. The tests would then become an acute measure for development progress.

Analysing the behaviour of the product in terms of its inputs and outputs would help reveal any underlying edge cases. This would also help with prioritising of tasks and allocation of development time.

To help not forget details such as committing, taking the time to prepare for a development session would help with the maintenance of any checklists or documentation used in the process. That said, the overall development process showed signs of a good methodology.

I USE AN AGILE DEVELOPMENT PROCESS – Improving

Feedback: Taking note of the information the client provided and asking about the inputs and outputs captured the basic requirements of the system. Taking note of the example the client provided and using it in an input-output table helped clarify the requirements. However, explorating the input’s variation may have been needed to capture finer details and edge cases.

I WRITE CODE THAT IS EASY TO CHANGE – Improving

Feedback: Source control management could have been used to document the development history of the project which would provide a reliable reference for rollbacks and a solid basis for future change.

The second test tested for attributes that had to be exposed. This made refactors such as changing from variables to a hash impossible. Tests can be decoupled from the implementation by only testing for behaviours and avoiding testing for state. In other words, testing for the interface provided by a given object helps ensure that changes in the implementation can take place without requiring changes in the tests and vice-versa.

The tests' names indicated the behaviour they were testing for making the usage of the product clearer. Names such as 'grade_counter' were part of the ubiquitous language making the intention behind the code easy to discover. This helped improve the changeability of the code.

I CAN JUSTIFY THE WAY I WORK – Strong

Feedback: Verbalising the development process helped the client keep track of the workflow. At all times, it was clear why certain actions were being done because the justifications given for each action were sound. The use of the client's exact example during testing was an effective way to tune the workflow directly to the client's needs.