Paul Martin | Can apply to jobs | May 14, 2019 1:30pm

General feedback

Please see the goal-specific feedback below :)

I can TDD Anything — Strong

You began programming in short, tight TDD loops to get the basic structure set up. You were running your tests every few seconds, which was really impressive. Your first test used an extremely simplified version of the test cases - an empty string. It's hard to think of a simpler step to take so this was great TDD. You slimed the answer to this test, which got you to a passing first commit. You then gradually built up your test suite with increasingly complex cases, which rapidly made progress through the application. When I challenged you about an earlier test case, you rolled back to that test case using the `x` prefix.

I can program fluently — Strong

It was a good sign that you had added a command specifically to change the colour mode of your terminal. You were moving very fluently around the command-line, using the `-l` flag with `ls` to get a clearer picture. You had no problem whatsoever with basic testing or Ruby syntax. When it came to implementing the algorithm, you were moving comfortable around the string, using conditionals and indices to find the correct character. You used a REPL to support the algorithmic design.

I can debug anything — Steady

You hit one small bug around a matcher, which you'd named `toBe` instead to `to_be`. You read the error and rapidly fixed it up. When you hit a small bug around evaluating an unusual string, you solved it by repeatedly referencing the command-line tests.

I can model anything — Steady

When modelling the program, you got a high-level picture of the input and output of the program, along with the type signature. When modelling the algorithm, you used a tight test loop to begin developing the arithmetic engine.

I can refactor anything — Steady

You were careful to refactor your test messages as you went through, in order to get to a readable version of the passing tests.

I have a methodical approach to solving problems — Strong

You started by reifying the fuzzy requirements into a clearer set of requirements. You mentioned several edge-cases sprang to mind but decided to pick a simplified version of the problem – simply handling basic addition. This was a good sign of methodicality in breaking down large problems. You went to testing next, which was a great sign. You then headed into tight TDD loops. You were moving extremely capably around the process, very fast. You continued to gradually complexify the application, always on a rapid programming pathway. I really liked how you added a "2+" test case, but you expected something that I as a user would not have expected. I called you up on this and you rolled back in the process, commenting-out code that you didn't want and making use of the `x` prefix. Do watch out for introducing edge cases a little too early. Introducing the " 2+" string was quite a trick case to handle. You used Stack Overflow to get a picture of precisely what you wanted to do.

I use an Agile product development process — Steady

You asked good questions to get a picture of the input and the output. You quickly got an understanding of the user requirements and began programming in tight TDD loops. You committed frequently, with clear messages that would give a neat story of how the application had come about. If you have the benefit of having the user nearby, it's a good idea to check acceptance criteria frequently – at one point, you programmed some acceptance criteria that weren't accurate for me as a user.

I write code that is easy to change — Steady

The code you wrote was very thoughtful, especially regarding the tests. Your test cases were built up carefully and your commit messages gave a clear and coherent story of how the app came to be.

I can justify the way I work — Steady

You gave a good justification about how you would approach the problem given the time limit you had. You justified a vacuous test by explaining that it was about evaluating basic arithmetic. You justified introducing a conditional by differentiating between simplicity and ease. During a tech interview, I'd strongly advise narrating your process out-loud as it gives an insight into your working processes which, as an interviewer, I didn't otherwise have access to. While you program in a very intentional way, I think adding some narration – at least a quick overview of your intentions every so often – would only stand to enhance this approach.

I grow collaboratively — Steady

I manage my own wellbeing — Steady

I can learn anything by myself — Steady