**Mindset Exercise Responses:**  
  
1) Choose **3** of 5 instances below where you can provide **specific examples.**

**Question**: Identified a problem or inefficiency in a process related to quality or development, and implemented a change to improve it   
  
**Response**:   
     In my previous job, the execution of UI automated tests was a major process in determining the release quality, however, due to the enormous number of tests in our suite, it took a great deal of time to run them. Often, they would fail due to their flakiness; stalling the release as there weren’t enough green on the release pipeline. Eventually it became a matter of clicking the re-run button until there were enough green runs on the release. I began looking into this issue soon and found that a lot of the tests were failing to create the basic test data (like creation of new organisations, discounts etc). The tests most times failed due to timing out, or because of using excessive UI automation. In a lot of teams, the data-setup would happen via db script but such was not the case here due to constrains that running db scripts on testing environments would need DBA intervention which was a costly affair in terms of time and effort.  My solution was to somehow remove the dependency on data-setups, therefore, I integrated the tests with the BFF API layer to be able to seamlessly create the data in without interacting with the browser interface.     
  
**Outcome**: The new changes to the way we setup data in our UI automated tests greatly improved our execution time and accuracy giving greater confidence in our releases.  

**Question**: Built or implemented new testing tools or frameworks    
**Response**:   
 My response to this question might seem like a continuation to the previous answer. :)   After I was able to prove that the BFF-API layer can be exploited for our automated testing benefits by exposing a vulnerability in the code, I began working proactively to gather more useful information like the location to the different API specs, their swagger definitions, etc., in order to design the testing framework which would enable us to write more API focused E2E tests. These tests would then become an integral part of our automated QA strategy in the team.

 For the purpose of this framework - I used my existing knowledge of Typescript which was luckily also the preferred language in writing our UI Automated tests, which resulted in a smoother transition of other QAs into writing them in future. I designed my framework in terms of various services and features fragmenting the code in a page object model style where different classes in my code represented the features, or the services.    
  
**Outcome**: The new API testing framework not only began replacing the existing lengthy UI Automated tests in a ration 3:1, but also gave more confidence to the team in the releases, gaining the much-required respect from the developers in the team in the QA efforts.  

**Question**:

Attempted to convince someone to change the way they were working,

even though they didn’t want to:   
  
**Response:**  
 You know where this is going on the basis on the previous 2 answers above, right? :)  This indeed, is an interesting story of my journey in a previous company and here is how it all began. Let’s assume the company name is X, and the team is Y for the purpose of anonymity.

After joining X, during the first few weeks in the team Y, I realized they invested heavily in writing and maintaining the UI automated tests. The onus of e2e testing fell largely on the QAs as the gatekeepers of the release.  
   
My responsibility in the role was to also suggest of using better practices and tooling for the purpose of testing automation.  After bringing up numerous impromptu discussions regarding the automation approach with the team, I realized they were convinced that the process is the best they can have and that there is nothing much to improve and there was also a time constraint attached to the releases barring them to think outside the box.   
I had learned the art of e2e API automation in the past and was keen to suggest and implement the same as a way of testing closer to the source code for an environment where unit testing and integration testing was solely a developer’s responsibility and happened only if they had time to invest in it.     
I came up with a Proof of Concept for using e2e APIs to test the applications and presented to the wider team along with the QAs. The approach was received well but only to an extend of thinking of it as means to improve the test execution speed and wasn’t necessarily looked at as a potential mainstay in the overall QA strategy.  
It was a disappointing end to my efforts, but that didn’t stop me from trying to reason with the team time and again of my suggested approach.  
One day, I found my first opportunity to prove that the e2e API framework could provide more than just the speed. Since I continued working on my API testing efforts proactively, I was able to find vulnerability in an API using my tests. It resulted in an immediate attention from the lead QA as I got their backing to actively start contributing to the framework.    
  
**Outcome**: It took me around 12 months of back and forth with the idea, but I finally got there. The lead QA finally gave me the much-required nod for my API testing approach and began actively involving themselves into writing new code. By the time I left the organization, the lead QA was the biggest contributor to the API testing repository.

**Question:** When you are testing a new feature, under what circumstances would you deviate from a test script while performing manual testing?

**Response:**  
  
1) When analysing a new system in test.

2) When testing requirements are unclear from the story, the normal action would be to use the exploratory testing approach to figure the right way for quality analysis.

3) When performing general exploratory testing.

4) When an unexpected issue occurs while following a test script, the deviation from the conventional script would occur to get to the bottom of the issue.

5) When writing automated tests in an agile environment which is dynamic by nature, a test script might soon become obsolete to follow as the system or the feature under test may have changed.

6) When debugging a production issue for a feature, the test script would normally become less important as more focus lays in looking at errors on the various monitoring tools in place, or speaking to the right people who first encountered the issue to try and form a basis for further actioning.

**Question:** If you joined a team of 10 developers as the only Quality Engineer and could

implement and change any process(es) you’d like, how would you ensure that the team delivers high quality software?

**Response:**   
  
I’d advocate towards having a **left shift approach**, i.e. shifting the responsibility of testing on the developers. For this to happen, I would consider the following points, but not necessarily in the same order:

1. Promote early check-ins with the developers and the business to ensure quality is not overlooked from the start.
2. Advocate for QA-Dev pairing up for exploratory testing, this promotes constant upskilling for developers who may not be as experienced in quality assurance.
3. Advocate for QA demos as a way to showcase feature dev work.
4. Insist on releasing smaller but more often releases to mitigate any risks.
5. Help setup test automation frameworks to facilitate a targeted regression testing strategy for every release.
6. Advocate towards making the code easily testable, e.g. ensuring that the code enables test automation at all levels, i.e. unit, integration, e2e.
7. Ensure the automated tests are easy to write and debug, for issues if any, arising on the build pipelines.
8. If found under the pressure of time, I'd request the entire team to join in the exploratory testing efforts to ensure quality is not overlooked.
9. Advocate for a feature service to enable releases to production behind the feature flag, this way, if issues are encountered, they stay confined in a boundary and we get ample time to refine them.
10. Promote occasional dark-launches for bigger releases, by this I mean, releasing feature internally to get it tested by other employees in the company as a way to gain confidence in the software.
11. Aim for running targeted automated regression at the e2e level. I’d also advocate at having a larger test coverage at the unit and integration levels and would advise against having too many e2e automated tests.