Josh Edwards

Open Ended Questions

Section 1

Q1)

1. Utility
    a. What do we do that isn't done elsewhere?
    b. Can we combine this function with an existing codebase, or is it better to be standalone?
2. Performance/Scalability
    a. Are we keeping data sent back and forth to a bare minimum for functionality and performance
    b. How do we handle multiple users simultaneously
        i. Multi-threading?
        ii. Multi-instance with load balancer?
        iii. Some other queuing?
3. Usability
    a. Does this application make its functionality plain to see?
        i. If not, why not – do we need to build in more documentation?
        ii. If so, how do we make sure the users are getting everything they need
            1. Feedback url/email?
    b. Have different platforms been considered?
        i. Mobile
        ii. Desktop
        iii. Some other interface – like a car, or some manner of integrated system?
    c. Has this been reviewed by UX so we have confidence that it is going to do what we need to before we push it out the door?
4. Maintainability
    a. Does the code meet coding standards of the practice?
    b. Has it been peer-reviewed?
    c. What's the process for reporting bugs (Jira tickets? Email?)
    d. Is there a plan in place to revisit regularly to make sure it's still meeting the needs of the target audience?

Q2)

Honestly, I wait to see what the major publishers are putting out for books, I follow Manning Publications on Twitter and when I see that a book has been put out for a given framework, I consider it. I like to let frameworks sit a little while so other people can work out the kinks, I like to use frameworks to write interesting software, rather than debugging someone's framework and having to hack together my applications. I also, of course, talk with friends and colleagues and follow various folks I've listened to at conferences.

Section 2

Q1)

I'd love to say that I go Test Driven Development every time; but more often than not, I find myself back-filling unit tests and integration tests. I do take testing into consideration and will rewrite functions to be more testable as development continues. I think the best practice is to keep every function short and discrete; this increases the likelihood that it will be testable via unit testing and integration testing later. Of course, there's also a good round of human functional testing as well; but I've found for maintainability into the future having the automated tests is critical to making sure you are returning APIs that are consistent release to release.

Q2)

Order and consistency. If code is laid out in one way for half the functionality and another way for the other half, it becomes harder to see where there are redundancies that could be reduced, and maintenance becomes nightmarish. Also, comments! Even if you're the most amazing and fluent programmer ever, writing a comment to describe how a particularly tricky piece of code is navigated will help you or your future coworker immensely. You don't need to go overboard, and in fact, you should avoid excessive comments (yes, we know that i++; is incrementing the iterator, don't comment that!); excessive comments actually make the code harder to read, particularly if it's a simple code flow that's a common pattern; but when a piece of code required you to go to stack overflow, make note of it, add the url to where you found the answer in a comment, if nothing else, it will give some tips to future-you of where to look for other tricks (or certain posters whose advise is poor to avoid).

Section 3

Q1)

I love the challenge of taking a set of tools and making something useable from it. I love to find novel ways to use existing tools; and I love seeing it all come together to form a useable application that improves folks' computer experiences.

Q2)

Honestly, I wrote and have been maintaining a keyword search application (called FastJump) internal to MITRE for the last 9 years. It's simple and it's integrated with every page of our intranet via a common header. It's probably not even noticed by most people most of the time, but every communication we have tells people to "FastJump: <some term> for more information." It's become so ingrained in our daily usage that it's achieved a level of invisibility, and it's so stable and performant that it only ever goes down if there's a major network outage (I'm lucky to be able to count the number of times I've seen that in my tenure on one hand). It has very simple interaction and it just works. We continue to evolve the workflows on the backend for the administrators to be able to use it easily, but the users have had 1 consistent interface that's been steady for my whole tenure at MITRE. I'm proud to have built an app that gets used without thought because it's intuitive and just works.