Scenario 1:
You are brought on to help with the latest LFL VR masterpiece that has already been in development for 6 months. Your support will help take it from a working alpha to a feature complete beta. You are familiar with the functionality as you have played around with the experience casually and even understand the overall structure from a few offsite lunches with the lead engineer. After being assigned a few “simple” bugs you get to work. Immediately you realize a core piece of the code that can be improved for efficiency is being used throughout the project and has been in the code base for almost 5 months.

● Questions:

○ How would you attempt to understand the design of a large code base in a short period of time?

○ What steps will you take to understand the ramifications of the potential refactor?

○ How do you balance the task of fixing the bug with the investigation?

1. To be able to understand the overall design of a large code base, the best thing is to work on a bug that involves a lot of code in that project. When you start to work on that bug, you would immediately start to see how it affects the project at a high level. You would then dive into the system that is at fault and understand how that system was made by asking yourself: “what was the reason behind adding this system in the first place”, “What did their timeline look like”, “What did their testing / implementation plan look like”. When asking yourself these questions you get to know the risk and effect of the system’s addition in the first place and what is necessary in order to improve the system while maintaining its original function. So when you ask yourself those questions you get to know its original function, playing your mind in the engineers who designed it in the first place. When doing this you start to develop an approach of your own, you start to envelope the system itself, making your understanding greater than before because then you can ask questions to the developers of that system at a deeper level to get the most understanding from them as well. You could also kick start this by going into any documentation regarding this feature and maybe any archived conversations or meeting notes where they discussed the system at all. To understand the ramifications of the potential refactor, you would start off by trying to understand the risk of the refactor's negative effects. One is if the refactor doesn't go well by taking up too much time or resources or if it actually does more harm than good to the project if it ends up causing other systems to fail at the expense of that single feature improvement. It might be helpful also to get the user end’s effect on this by not just understanding the technical weight this has on the project but the product weight. What I mean is how many users will be affected by this feature since they will be the ones noticing the greatest difference in their user experience. So getting a product manager involved to understand feature analytics and feedback is crucial if it is a user facing feature. For the technical side you would have to lay out a map of all the areas this code touches and how many classes / files are needed to work on in order to give a rough estimate of time and resources needed to do the refactor. (amount of code * requested level of quality of code) + testing and feedback = amount of resources needed, roughly, to ship the improved system. On balancing the bug with investigation, as a developer I tend to try to both at the same time but I prioritize the bug fixing since it would take priority since that is the immediate “help” the team is waiting

on since it has the greatest short term value on the project. As I start fixing the bug, I would note key parts of the classes / systems I touch and how it connects to other parts of the project that need to be improved. By the time I'm done with the bug I would have a list of nice to haves to necessary changes I saw fit while working on the bug, especially since some of these requests would have made the next engineer to work on the next iteration of the system less painful. All of these changes could be weighted and prioritized by leads to get their perspective on how manpower should be applied to each need and want since this change could affect other people's work.