[From Phase II Requirements,

[Preliminary evaluation re: completeness, consistency, and quality of the detailed design model]

As the project has progressed, consistency has increased. The design class diagram has forced us to acknowledge our inconsistency across design diagrams; we retroactively made notes or made revisions to design diagrams from earlier in the process so that all classes, states, attributes, and methods made sense. We even discovered inconsistency within select UML diagrams, where methods or attributes overlapped one another. A large source of contention was with our system sequence diagrams, where we found that we had a large number of classes that individually did not do terribly much; we had to compromise on ways to make more useful classes without assigning too much responsibility to any one class, but we also had to ensure that we stuck close to our original design to avoid erroneously losing functionality with our projected design.

Speaking more of structure, we have collaborated with one another over whether attributes are unique, and have checked one another to ensure any changes made the diagrams—and thus the overall design—more simplistic and straightforward. However, with multiple people giving input, settling on a single logical decision has been difficult. At times, this led to a stray method or attribute being improperly placed in a completely unassociated class, such as the Employee class having to handle login and logout duties when we agreed our design would be based around a session handling login and logout.

Our naming schema had to be quite straightforward—if not a bit verbose—due to the fact that everyone in the group was required to understand how the design interconnected. Even if names changed over time, the names evolved in a manner that we could each easily understand the changes after time away from the group.

There were attempts at numerous points of the process to design software that did more than the minimal design for which we wanted to shoot. We agreed that a simple approach with uncomplicated inputs would be best for our project, so if we wanted to, say, search for a MIMS user or a patient, there would be no confusion regarding the ID or name used in the search process. We also ensured that IDs for different types of personas within the system would not be easily confused, to hopefully match our expectation of keeping things as obviously differentiated as possible. Keeping this in mind forced us to take into consideration each aspect of our selected use cases, promoting proper validation.

Our simplistic design also meant that ensuring the program was being built properly would be a bit less of a concern, though the actual implementation of our design may not be quite as simple as limitations are realized. Still, from the abstract design standpoint, no glaring omissions were found.

That said, our initial design seemed to ignore select specifics, such as indecision on whether to use vectors or dynamic arrays, or being unsure over whether to use primitive types or use included classes with additional functions. From this standpoint, the design was neither as complete nor as concrete as it could have been, though we plan to continually make notes of changes we make and retroactively create revisions to our design. In this way, we will ensure that we did not leave anything out, and we did not add too many extraneous capabilities or make our code too bombastic for the given task at hand.

We designed more with the concept of a tightly interwoven system, but by ensuring we kept classes lightweight with low coupling, we believe our design is flexible enough to be reused elsewhere with most of the maintenance focusing on building upon our design, not changing the design at its core. As diagrams are polished—in turn polishing our design—we see a capable, competent package that we could revisit in the future with additions that would not pose us a problem whatsoever.