1. Using an example, explain why the design activities of architectural design, database design, interface design, and component design are interdependent.(examples must be different from books examples)

The example I will use is the design activities we use at my full stack developer job, Hobby Lobby corporate offices. The following will apply to a large number of our in-house apps, but I will use the inventory management app for this example.

**Architectural Design:** The architectural design of our apps takes a very specific route. This process includes managing how the app will run and how all of its components will be connected. This includes things like what kind of backend we’re using (NETCore or Elixar), what kind of security or authentication will be built into the app, and how the app will “talk” to our other services. Essentially, it’s the boring stuff that happens behind the scenes.

**Database Design:** This step is very important. There are not many apps at Hobby Lobby that don’t use a database. The database design includes deciding what database to use, if existing tables should be used or if new tables need to be created, what kind of attributes need to be in those tables, and how that data is going to be retrieved from the database. I finished an app recently where I had to create an API in order for my app to retrieve the information from the database. The choice to create an API was a design decision that goes along with the database design activities for this specific app.

**Interface Design:** This step is also very important. This step dictates what the user is going to see and interact with. There are almost unlimited design choices for this step. Should the app be single page or multipage? Should we use toggle switches or dropdown selection? Should we use icons or buttons? From working, I’ve gathered that interface design mostly boils down to personal preference and is also mainly impacted by each app’s specific needs. However, it’s still very important. One could have an amazing app, but if the UI is terrible, then no one is going to want to use the app no matter how great the logic is underneath.

**Component Design:** In my example, component design mainly includes deciding what components will be in the app and how they will be used. Should we app functionality for the user to search the inventory list with a search bar? Should we only allow the user to view one item at a time? These are all components that go into the app and decisions must be made on how they will function.

In this example, all of these design components are very much interdependent. Changing one aspect of one component will change parts of other components. For example, if someone wants to implement a certain functionality of a component, then they may have to change or make changes to the backend. Changing the backend may change how the database connections are working. Changing the database may change what data is received back. Changing the data received may lead to changes in the UI to support the viewing of this new data. I could give countless more examples, but its apparent that all of the design components of an app are dependent of each other and changing one could very well affect one or more other components.

1. Imagine that a government wants a software program that helps to keep track of the utilization of the country’s vast mineral resources. Although the requirements put forward by the government were not very clear, a software company was tasked with the development of a prototype. The government found the prototype impressive, and asked it be extended to be the actual system that would be used. Discuss the pros and cons of taking this approach.(2 for each)

**Pros:**

1. One pro of making the prototype without a lot of clear direction is that it allows the software developers to be creative without all of the design constraints of a normal project.
2. Another benefit of this is that sometimes an app can be “over planned”. Sometimes so much time is taken planning an app that maybe it would have been faster to just start on the app and then reassess it as time goes on.
3. I see this one a lot at my work, but sometimes the users don’t quite know what they want. I’ve learned that sometimes users think they know what they want, but in actuality they really don’t. It can be better sometimes to take your own direction with the app and then occasionally show the user your progress and get their feedback on what you’ve implemented in the app.

**Cons:**

1. An obvious con of this is the lack of clear planning. Sometimes jumping into an app without a plan can work (if it’s a simple app). Other times, this makes the developer have to do twice the amount of work. Say for example the developer works two weeks on a specific feature of the app and then shows it to the user and the user dislikes it. Then that’s two weeks of wasted work that could have been avoided if better planning was taking place.
2. Another reason lack of a clear direction is bad is that the developers have no clue of the future of the app. Maybe a developer will make an app a specific way based on certain future assumptions. Maybe those assumptions turn out to be incorrect and now the app isn’t going to be poised for success. An example is maybe a developer knows that an app can support X number of customers at once. Maybe the user knows that in a few months, the app will need to be able to support 2X people. If the user doesn’t relay this information to the developer, then the developer may not make the necessary changes to support the growth and scalability of the app. The app will end up failing with the large influx of new customers.