1.

**a)** As software engineering covers the lifespan of a project – from its concept to its release, there is a need to communicate what a project does before coding it, especially if seeking funding to begin work on it. As such, one can use software modeling, which is a way of showing what a finished project will do via different models.

**b)** One of the foremost reasons of needing tools for software modeling is uniformity in communication, keeping a cohesive language. By having specific tools with consistent meanings, it ensures everyone is understanding what is being communicated, whereas without them someone might say something to a room and everyone hears something different, none of which what the speaker was trying to say.

**c)** User stories, which contain a clear narrative and image of a specific feature. Data Flow Diagrams, which show a simplified view of how information moves through the system. UML Use Case Diagrams, which show a user-focused flow of actions. And use cases, which detail step-by-step processes for how a specific subsystem works between the actors and systems.

**d)** While I feel like I’ve used words meaning ‘simple’ here a lot, software modeling is supposed to encapsulate the process – that is, ideally one is not adding anything that they have not modeled to the final project. By adhering to the models, one has a clear map to follow and a blueprint for how systems interact, helping allot time and anticipate where issues or bugs may arise within the project. Not only that, but by following the strict map, one can avoid the pitfalls of feature creep or bloat, which add unplanned complexity to the project, costing unpredictable and unplanned time to manage.

e) I feel like a UML Use Case diagram would be very effective for estimation for load on adding to an established framework as I am, as it lets me examine the flow actions via actors and lets me examine how the systems interact. It lets me see how subsystems interact and where issues might arise, as well as see how large subsystems are and how long their relative time would take.

2.

The first page specifies that Agile tries to be adaptive rather than predictive, and people-oriented rather than process-oriented. These principles emphasize these design ideas: Agile wants to be quick and functional, and wants to revolve around people and their interactions.

A few principles don’t feel like they can apply easily to my final project, but I suspect this is largely because I am in an individual group. As such, I chose ‘Working software is the primary measure of progress’ and ‘Simplicity – the art of maximizing the amount of work not done – is essential.’

The first one is clearly important to software project development. Everyone has big ideas, but consumers rightfully want results – if the code’s not working, it might as well not be there. This is easy to keep in mind for the project – while a lot of the language is about proposed changes, it’s proposed changes we should be able to implement, and failure to implement those changes is a reflection against the proposal. In the end, results matter.

The second one is also easy to justify as important to software project development, especially following the above. If results matter, it matters foremost to get what needs to work working, and anything else will get in the way. Simplicity is usually the clearest path to results. For example, my final project is adding a wishlist to the microservice frontend. I was considering a lot of ways to implement it, but the first way I tried was simply by creating another cart by adding ‘2’ at the end of the ‘email’ field. It’s dirty and not something I would do were I making a large project, but doing so allowed me to just re-use the shopping cart HTML and Javascript with minor edits rather than building a whole system from scratch.

3.

The article and video emphasize that architecture is a misnomer for the role in software development, and that something like ‘city planner’ or ‘guide’ is a more accurate analogy, roles that are more responsive to changes over time and flexible in their planning. Very often architecture is what is considered ‘important,’ and what’s considered important is because it’s hard to change. There was discussion about irreversibility of design, and how that should be eliminated in more iterative and adaptive design. After all, adaptability and responsiveness to changes are limited the more something has parts that are seen as uncompromising in their needs and use cases. While this is all well and good, there is some acknowledge that the more flexible and ‘simple’ a product is made to use the more complicated and costly it can be under the hood, and so software architects (or planners, or guides) need to identify which systems are constraining the usability of the system and which ones will remain functional if left more rigid.

I think my answer to this might be a little bit different because of being an individual group and as such adding to something that already exists. There’s already architecture there, but there is a need for refurbishing for my needs, and this relates heavily to video and article. When adding a wishlist, I was running into issues maneuvering to the shopping cart from it – because the way the microservice frontend is built is tied to refreshing the email address for it in the toShoppingCart function. This isn’t a huge problem for the frontend as-is because its use cases don’t run into many issues with it. For this reason, I considered its design integral and immutable, and tried to build the wishlist in a way that didn’t interfere with it. Ultimately, though, inspiration struck, and I treated as something flexible, something I could change. As the needs of the system changed, architecture that interfered with the plans is something can be demolished. As the paper says, in software engineering we aren’t limited by physics, we’re only limited by imagination. Reorganizing the code to more easily and consistently respond to new features wasn’t costly, and should be encouraged, rather than a strict adherence to treating the frontend as an unbendable skeleton.