Basic principles of System Design

System design work usually takes the form of a diagram: boxes and arrows describing the main parts of the product, and how they all connect to each other. As we've written before, a good system design defines the following:

- **Elements:** What are the core elements or objects in the system?
- **Interconnections:** How are the elements connected? What are the relationships? What are their inputs and outputs?
- **Purpose:** What does this achieve?

What are the benefits of System design?

When you start to move your workloads to the cloud or build your applications, a major blocker to success is lack of documentation of the system. Documentation is especially important for correctly visualizing the architecture of your current deployments.

A properly documented cloud architecture establishes a common language and standards, which enable cross-functional teams to communicate and collaborate effectively. It also provides the information that's necessary to identify and guide future design decisions. Documentation should be written with your use cases in mind, to provide context for the design decisions.

Over time, your design decisions will evolve and change. The change history provides the context that your teams require to align initiatives, avoid duplication, and measure performance changes effectively over time. Change logs are particularly valuable when you onboard a new cloud architect who is not yet familiar with your current system design, strategy, or history.

Principles of good system design:

1. **Keep it as simple as possible to address today's known problems:** Don't add complexity to the system to solve hypothetical problems we might face in the future. Thinking ahead is great, but we shouldn't take on the burden of planning for eventualities that might not even happen. Let's not try to boil the ocean. If we can make the system more open-ended without adding a ton of complexity to the system or adding lots of work, that's great. But otherwise let's stay focused.

2. **Ensure that it's legible:** A successful system should be easy for users to understand when they interact with the product. They will need to look at the UI (when we design it) and be able to roughly figure out what the parts of the system are.

3. **Move complexity to infrequently used parts of the system**: We will accept less efficient workflows and higher learning curves for one-time tasks if it means we get to make common tasks faster and simpler.

4. **Don't take on non-core problems:** Don't get distracted by the implementation details. Remember, you're not designing the entire product when you're designing the system: you're designing the major interrelated parts of it. The system design needs to describe the shape of the solution. It doesn't need to answer every detailed design question.

5. **Build it to scale from simple use case to complicated use case:** One of the dangers of moving upmarket (i.e. focusing on larger customers) is that you can alienate and create unnecessary complexity for your smaller customers. It's too easy to design just for the needs of your most demanding users. The most elegant systems allow for simple use cases that then scale up as needed.

6. **Prioritize adjacency to the existing system:** All other things being equal, choose a system design that's most similar to what we currently have: less for us to build, less change for our users to adapt to.