While design patterns offer powerful solutions to recurring problems and promote reusability, they are not always the best choice in every situation. As discussed in class, one clear example where design patterns may not be beneficial is when they lead to unnecessary complexity; especially in simple applications or early-stage prototypes.

In a lecture, we talked about how some patterns, like Template Method or Singleton, introduce additional layers of abstraction or require more classes to be defined. This abstraction can be overkill if the problem being solved is straightforward. For instance, using a Singleton pattern just to restrict instantiation when a static variable or method would suffice can clutter the design and make it harder for others to read and maintain. In the lecture, this was referred to as "overengineering”. A situation where the desire to “do things right” with patterns actually slows down development and introduces rigid structures that aren’t needed.

Design patterns should serve the problem, not define it. Using them without a clear motivation violates the KISS (Keep It Simple, Stupid) principle discussed in the checklist of principles and strategies. This is especially risky for beginner developers who may not yet have the experience to apply patterns appropriately and might misuse them, leading to tight coupling or reduced clarity.

In short, while patterns are valuable tools, they should be applied thoughtfully. When the added abstraction outweighs the problem’s complexity, it’s better to stick to simpler, more direct solutions.