**When Patterns Backfire: Over-Engineering and Unnecessary Complexity**

Design patterns offer time-tested solutions to common design problems but using them in the wrong context can introduce more issues than they solve. A common mistake is applying a pattern simply because it's available, not because the problem requires it.

**Over-engineering for simple tasks.**  
Imagine building a small utility that formats dates in two styles. Instead of a simple conditional check or helper function, you implement a full Strategy pattern—defining interfaces, multiple classes, and maybe even a factory. While technically correct, this overcomplicates the codebase, making it harder to follow and violating the YAGNI (You Aren’t Gonna Need It) principle.

**Increased maintenance burden.**  
Each extra class, interface, or abstraction brings added responsibility. It must be understood, documented, and maintained. In smaller teams or projects with narrow scope, these additions often become overhead rather than assets.

**Harder for new developers.**  
Unnecessary patterns introduce abstraction layers that can confuse newcomers. Instead of focusing on the actual logic, they may get lost tracing through interfaces and class hierarchies that serve little real purpose.

In short, design patterns are valuable when used appropriately—but if the solution doesn’t naturally call for one, simplicity often wins.