Group H

Pattern in Action Prototype

Colby Frison
Emily Locklear
Grant Parkman
James Totah
Antonio Natusch Zarco

CS 4213 - Software Design Patterns
Fall 2025

AGENDA

01 **Strategy**
- Why We Chose Strategy
- How Strategy Works

02 **The Problem**
- Problem Description

03 **How We Applied Strategy**
- General Explanation of Our Pattern & Prototype
- Before & After Diagrams
- How It Improved Maintainability, Clarity, or Flexibility

04 **Demonstration & Closing**
- Video
- Questions
- Thank You

01

Our Selected Design Pattern

Why We Chose Our Pattern

- We wanted to be able to refactor our ModelManager class in a way that reduced tight coupling and strengthened decision flow.
- The Strategy Pattern allowed us to do that in a way that was simple to us.

How Our Pattern Works

The Strategy Pattern is great for:

- When you want the same functionality with different non-functional aspects
- When you want to be able to easily add or remove subclasses
- Maintaining readability

02

The Problem

The Problem

- The ModelManager class in our SmartWrite application suffered from flaws typical of monolithic design.
- It was difficult to test and maintain.

03

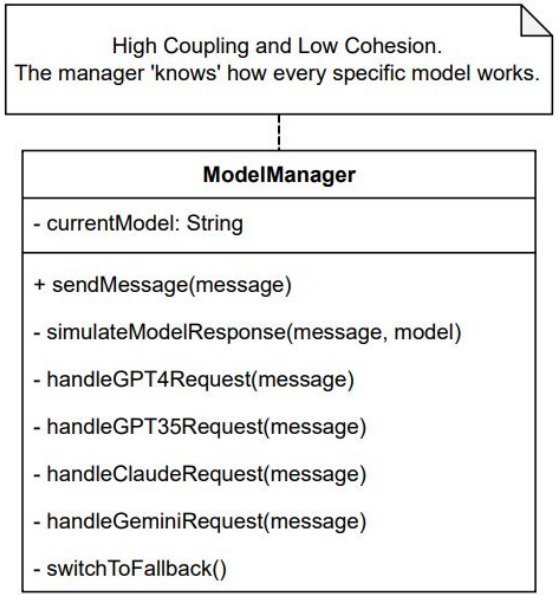
How We Applied Our Pattern

How We Applied Our Pattern

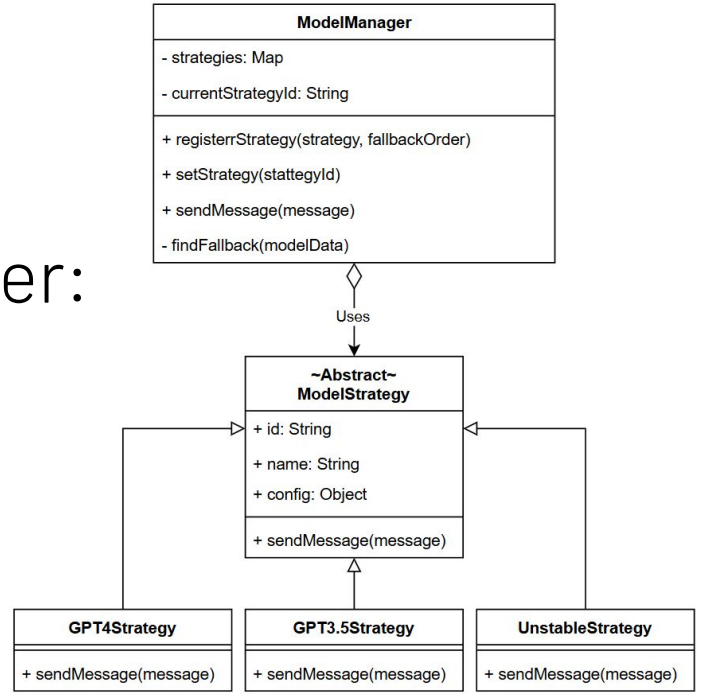
- *ModelStrategy* was added.
  - It defines the contract that all models adhere to.
  - Individual classes contain the specific logic for each API.

How We Applied Our Pattern

Before:



After:



How We Applied Our Pattern

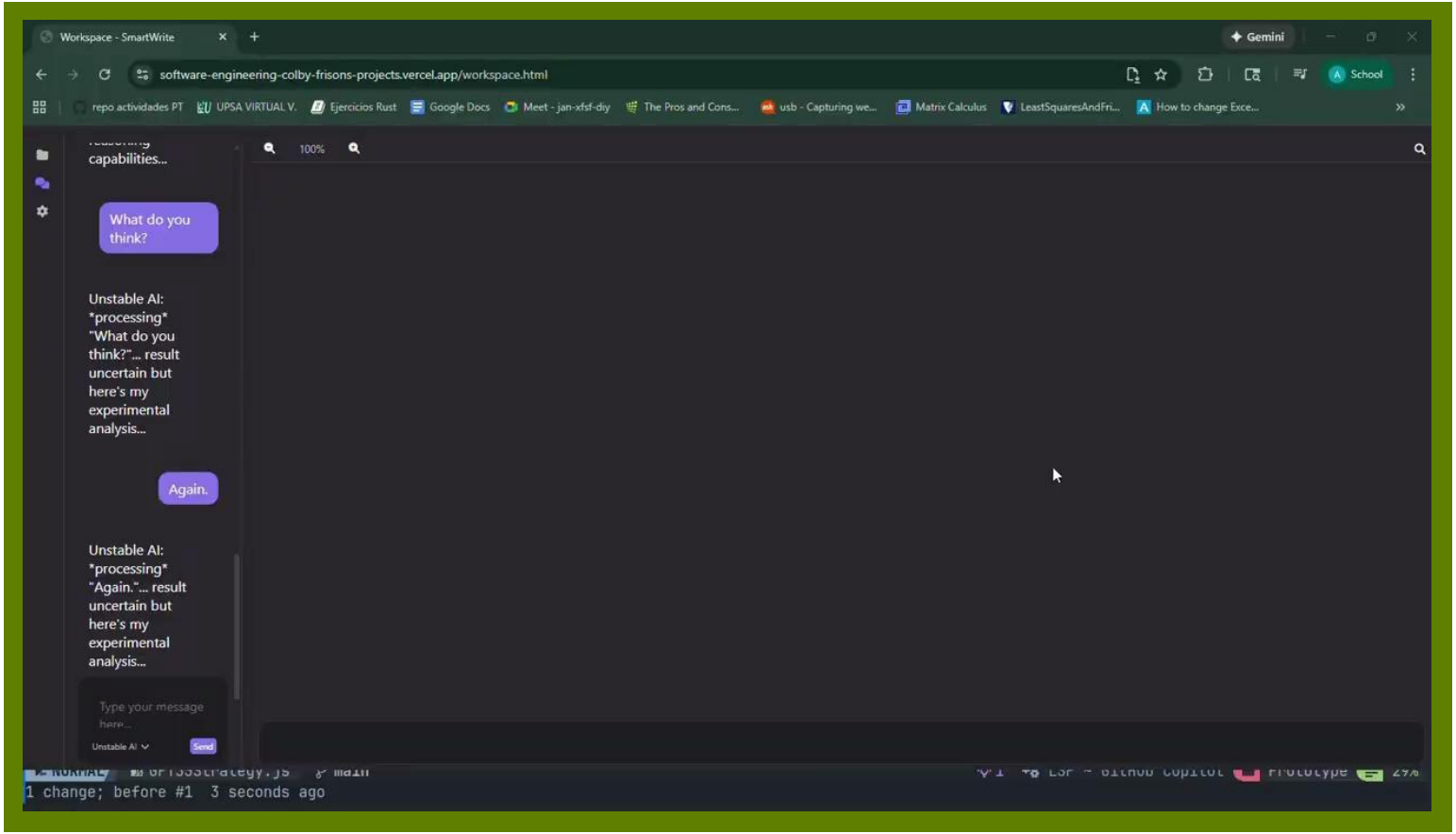
- Maintainability
  - Easy to add remove subclasses based on need.
- Testability
  - *UnstableStrategy* was designed to fail, allowing us to verify the fallback logic without relying on unstable external APIs or modifying production code.
- Separation of Concerns
  - The strategies focus on connectivity so the ModelManager can focus on reliability.

04

Demonstration and Closing

Pattern in Action Prototype

Play video



THANK YOU

Questions?