Grad student course project

Design Patterns as Language Constructs

Proposal

> *June 12th, 2025*

Overview

Team Members: Simon Walker

This research explores the integration of design patterns as explicit language constructs. It investigates how existing languages (such as C# event delegates or Rust's algebraic type system) reduce boilerplate, improve clarity, and enforce pattern integrity in a non-redundant way. The topic intimately connects core OO design principles to language design. Major elements include a comparative analysis of pattern implementation, how it is affected by language design, impact on software quality, and trade-offs between expressiveness vs. rigidity.

Design patterns are key for object-oriented design, but often suffer from implicit, difficult to refactor implementations that obscure intent. This topic bridges the gap between theoretical design patterns and practical language design, addressing critical software engineering values: clarity, maintainability, and scalability. The central question is: can generic design patterns be made into explicit language constructs that are clear and easily maintainable? The project is motivated by the potential to reduce cognitive load and accelerate robust system design.

Major presentation elements

- Introduction
  - (Briefly) the importance of design patterns
  - Limitations of traditional pattern implementation
    - Verbosity of boilerplate
    - Obscured intent
    - Maintenance overhead
    - Lack of assistive tools
- Solution: Design Patterns as Language Constructs
  - Theoretical benefits
    - Clarity (which in turn helps extensibility)
    - Conciseness
    - Reusability
  - Why the theoretical benefit is not just an ideal, but of tangible benefit to engineers
- Case studies and existing research
  - Language design features to directly implement design patterns explicitly
  - Examples of existing languages with explicit keywords for common design patterns
    - I expect to go through a large series of examples because the project is a survey of the many possibilities, not only an argument for the value of this type of language design.
  - I may go through a few specific cases for in-depth analysis
- Can more abstract (non OOP, architectural) patterns be implemented too?
- Challenges
  - Does a fixed implementation prevent necessary customization?
  - How is the risk of language bloat mitigated?
  - Community adoption
    - Even a theoretically "perfect" language is at the mercy of marketing and the politics of tech giants. -> However, some projects show promise: language transpilation (into existing languages)
- Conclusion