Engineering of Software Subsystems
Anatomy of a Pattern

- Component
  - operation
  - Concrete Component
    - operation()
  - Decorator
    - operation()
    - Concrete Dec #1
      - addedState
      - operation()
    - Concrete Dec #2
      - addedBehavior()
      - operation()
    - Decorator::operation()
      - addedBehavior()
What Are Patterns?

Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.

Christopher Alexander

A pattern is a *general* solution to a *problem* in a *context*

- **general** -- outline of approach only
- **problem** -- a recurring issue
- **context** -- consider the expected design evolution
Patterns allow us to gain from the experience, and mistakes, of others.

- Design for re-use is difficult
- Experienced designers:
  - Rarely start from first principles
  - Apply a working "handbook" of approaches
- Patterns make this experiential knowledge available to all
- Support evaluation of alternatives at higher level of abstraction
The most important piece of information about a pattern is its intent.

- The intent provides the general indication of when a pattern may be appropriate.
- Some intentions may be familiar sounding while others are not.

*Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.*

*Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.*
The main classification for Gang-of-Four design patterns is by purpose of the pattern’s intent.

- Creational: intent is mainly about creating objects
- Structural: intent is mainly about the structural relationship between the objects
- Behavioral: intent is mainly about the interactions between the objects
A second dimension for classification is binding time.

- Using inheritance is *compile-time (early) binding* or class-based
- Using delegation or composition is *run-time (late) binding* or object-based
- Creational
  - *class* => defer creation to subclasses
  - *object* => defer creation to another object
- Structural
  - *class* => structure via inheritance
  - *object* => structure via composition
- Behavioral
  - *class* => algorithms/control via inheritance
  - *object* => algorithms/control via object groups
To apply a pattern you need to, at least, know structure, participants, and collaborations.

- **Structure**
  - *The static class relationships between elements of the pattern.*
  - *Note: GoF structure is not standard UML notation*

- **Participants**
  - *Each class/object in the patterns*
  - *The responsibilities for each class/object*

- **Collaborations**
  - *General description of interactions between participants*
  - *Sequence diagram defining interactions*
The consequences describe the nuances of the pattern usage.

- How does the structure support the intent of the pattern?
- What are the trade-offs in pattern usage?
- Where are the variation points?
The implementation details can be language specific.

- Pitfalls to avoid when implementing the pattern
- Hints and techniques for applying the pattern
- Language-specific design choices