Introductory Discussion on Design Pattern

What is a software design pattern?

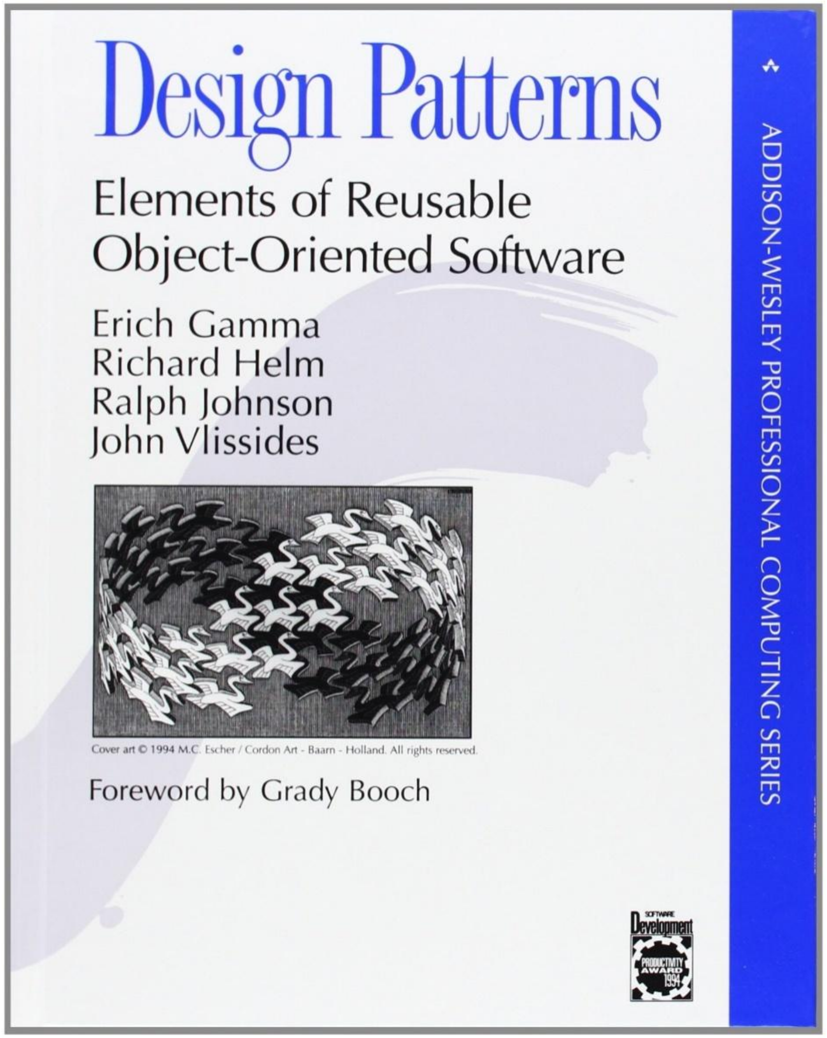
- A **standard** solution to a common programming problem
  - a design or implementation structure that achieves a particular purpose
  - a high-level programming idiom

- It is a description or **template** for how to solve a problem that can be used in many different situations

- A technique for making code **more flexible** or **efficient**
  - reduce coupling among program components
  - reduce memory overhead

- **Shorthand** for describing program design
  - a description of connections among program components

Gang of Four


Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Software Development Productivity Award 1994

- 23 design patterns
- Three categories:
  - ❑ Creational
  - ❑ Structural
  - ❑ Behavioral

Creational Patterns

- How objects are instantiated

- Five creational patterns
  - Factory method
  - Abstract factory
  - Builder
  - Prototype
  - Singleton

Structural Patterns

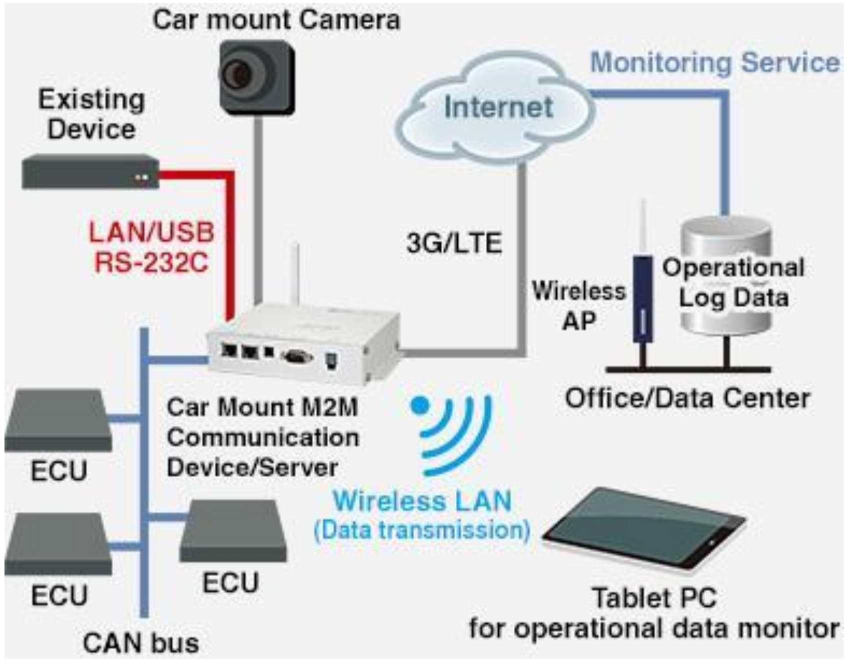
- How objects / classes can be combined

- Seven structural patterns
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Flyweight
  - Proxy

Behavioral Patterns

- How object communicate

- 11 behavioral patterns
  - Interpreter
  - Template Method
  - Chain of Responsibility
  - Command
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Visitor



Essential Elements of a Design Pattern

Intent	• Design goals of a pattern
Problem	• Describes when to apply the pattern • Explains the problem and its context
Solution	• Elements that make up the design • Relationships and collaborations among the elements
Consequences	• Benefits • Trade-offs

Why needed

- Requirements always change. Requirements change for a very simple set of reasons:
  - The users' view of their needs change as a result of their discussions with developers and from seeing new possibilities for the software.
  - The developers' view of the users' problem domain changes as they develop software to automate it and thus become more familiar with it.
  - The environment in which the software is being developed changes.

Why needed

- Rather than complaining about changing requirements, we should change the development process so that we can address change more effectively.
- The design of software should be able to accommodate many possible changes.
- Design patterns help to write codes that can accommodate changes easily.

Reasons to study Design Pattern

- Reuse existing, high-quality solutions to commonly recurring problems.
- Establish common terminology to improve communications within teams.
- Shift the level of thinking to a higher perspective.
- Decide whether I have the right design, not just one that works.
- Improve individual learning and team learning.
- Improve the modifiability of code.
- Facilitate adoption of improved design alternatives, even when patterns are not used explicitly.
- Discover alternatives to large inheritance hierarchies.

Modularity

- First approach to deal with change request is to write codes in **modular** way. Modularity definitely helps to make the code more understandable, and understandability makes the code easier to maintain.
- But modularity does not always help code deal with all of the variation it might encounter.

Other Approaches

- **Cohesion** refers to how closely the operations in a routine are related.
- **Coupling** refers to “the strength of a connection between two routines.
- Coupling is a complement to cohesion. Cohesion describes how strongly the internal contents of a routine are related to each other.
- Coupling describes how strongly a routine is related to other routines.
- The goal is to create routines with internal integrity (strong cohesion) and small, direct, visible, and flexible relations to other routines (loose coupling).