Tactical Design Patterns in .NET: Managing Responsibilities

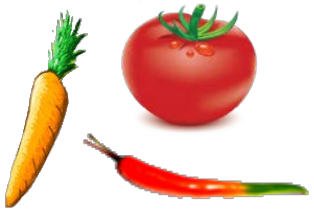
The Right Time to Apply a Design Pattern

Zoran Horvat

@zoranh75 | www.codinghelmet.com

Things to Keep in Mind

- Design patterns are not coding recipes
- Implementation details differ
- Otherwise, patterns would become ready-made components
- In rare cases design patterns were successfully implemented as reusable components



About This Course

- This course is not about design patterns
- The course is about how to *apply* design patterns
  - Programmers often complain that design patterns do not work well in practice
  - Design patterns may cause undesirable effects
  - The problem is that design patterns are often applied too early
  - This may lead to abandoning the whole idea of design patterns
- Be patient
  - Design patterns are not just mechanically applied to the design
  - Instead, design evolves into design pattern

Design Patterns and Class Responsibilities

- Responsibilities are fundamental ingredient of object-oriented design
  - When *not* done right, everyone is dependant on everyone else
  - When done right, classes are small and focused to their primary responsibility

The Single-Responsibility Principle (SRP):

A class should have only one reason to change.

Robert C. Martin

Favor object composition over class inheritance.

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

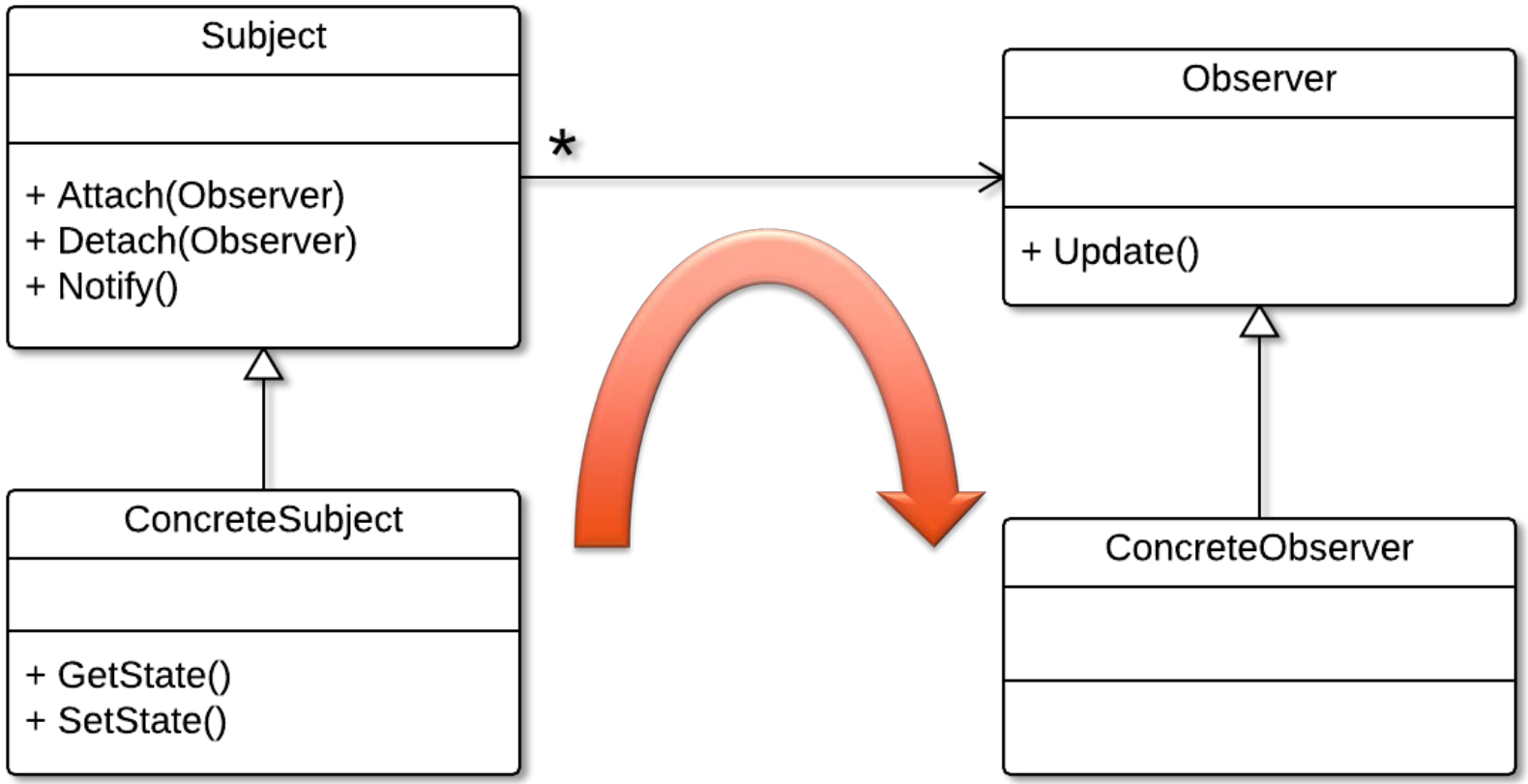
Before Watching this Course

- You should have functional understanding of at least these design patterns:
  - Composite
  - Visitor
- Composite design pattern
  - Convenient way to compose larger objects
- Visitor and Mixin
  - Convenient way to implement collaborations between objects

Evolving Design Patterns

- Do not just inject the design pattern into the solution
- Instead, evolve the solution to accommodate the design pattern

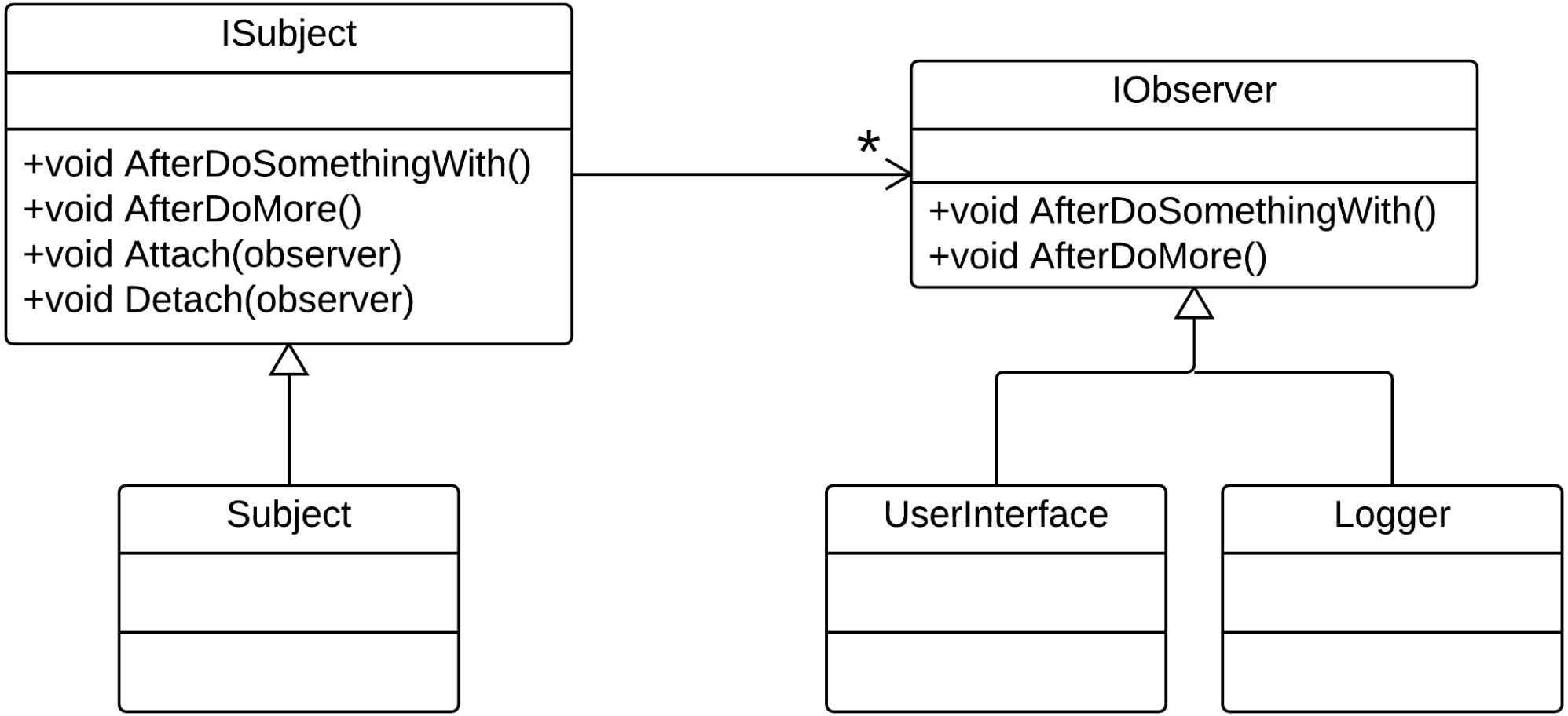
Example: Observer Design Pattern



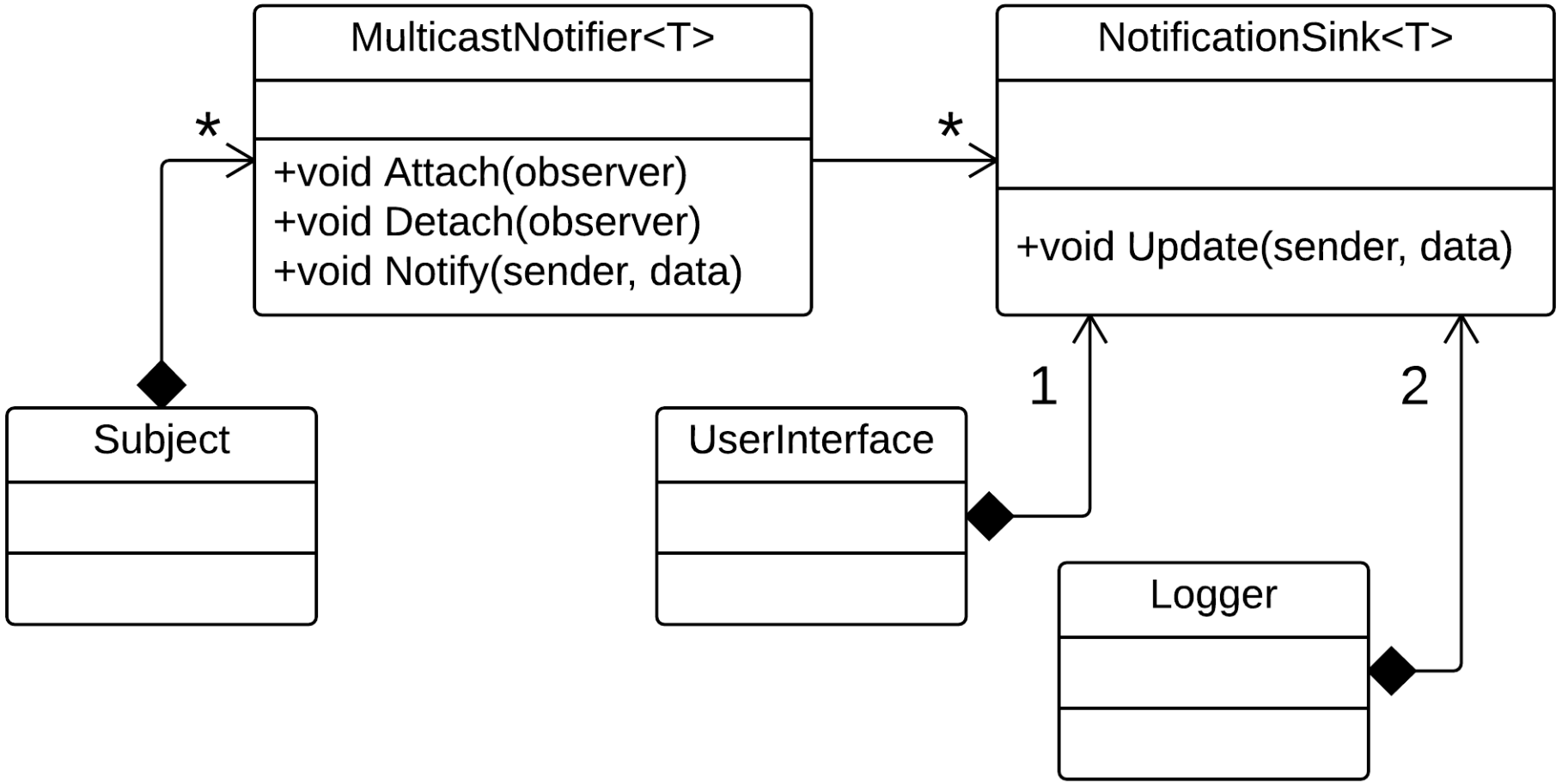
Observer Design Pattern Out of the Box

- Observer design pattern is supported by the C# language
  - Keywords delegate and event had to be added to the language to support it
- Implementation is not the same as principal design from the literature
  - Literature presents generic solutions that can be applied in many projects
  - Actual implementation is specific to language, framework, project…

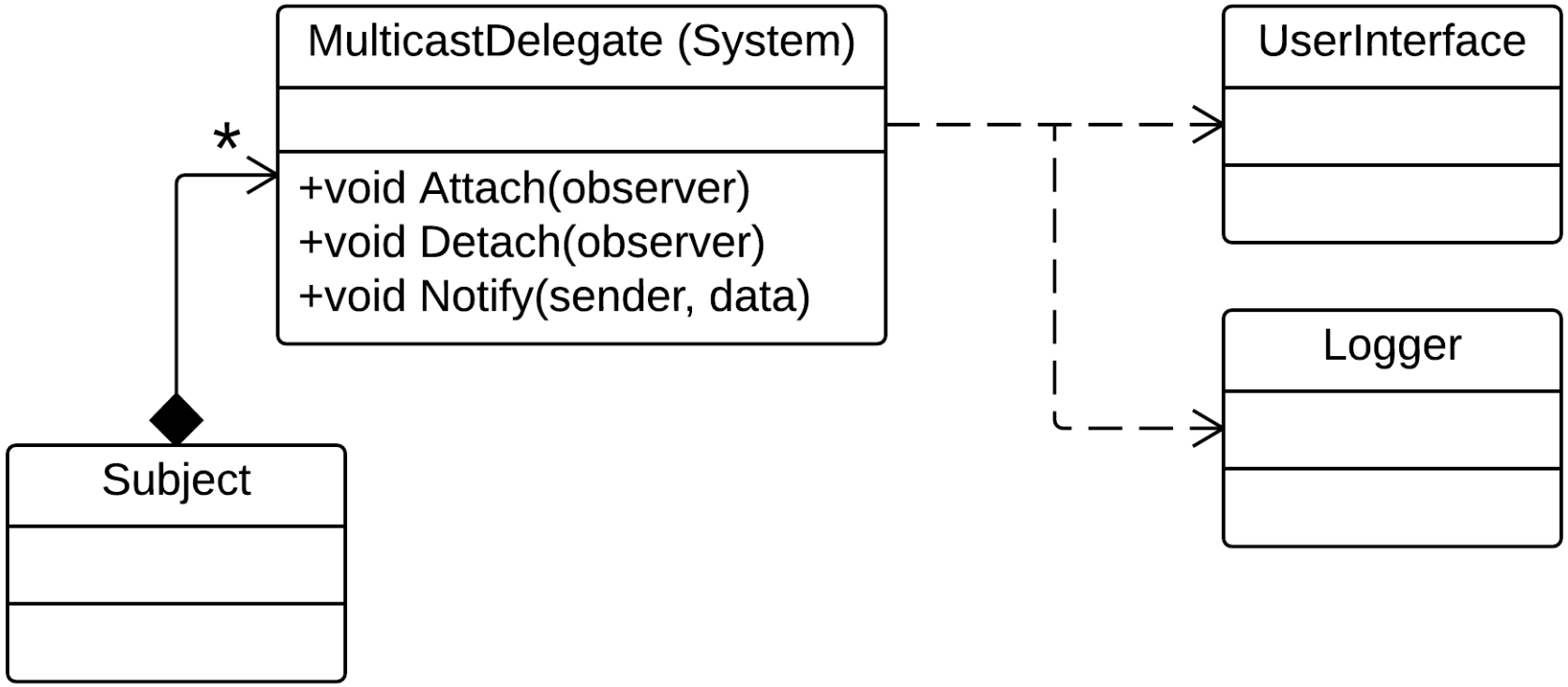
Observer Principal Design



Observer Custom Design



Observer Implementation in C#: Events



Other Design Patterns Out of the Box

- Several design patterns were implemented in the .NET Framework
  - Observer using delegate and event keywords
  - Factory Method – TryParse methods
  - Builder – connection string builder classes
- Design patterns can rarely be implemented as ready-made solutions
  - Attempts to make pattern libraries have failed
- Code generation improves chances of success
  - Entity Framework is applied through generated code
  - Runtime Callable Wrapper is generated from the COM interface

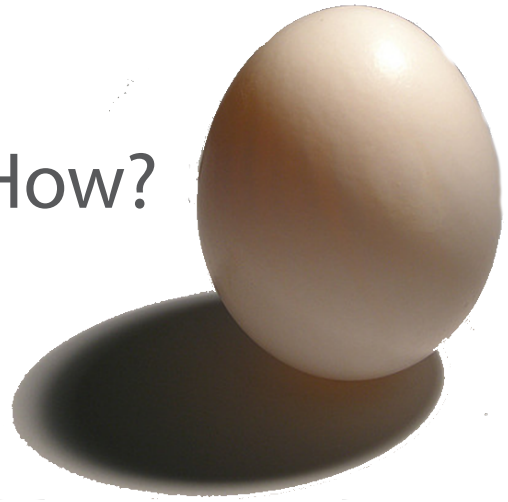
Design Patterns in the .NET Framework

- IEnumerable
  - Later it was used as base of LINQ library
  - Required extension methods, dynamic types, Func and Action, etc.
- IComparable and IComparer
- ICloneable
  - Deprecated because it doesn't communicate its intention well
- IEquatable
  - Important when implementing the Value Object design pattern
  - Methods not mentioned in the interface are required: GetHashCode, ==, !=
  - C# language is unable to communicate the whole pattern through the interface

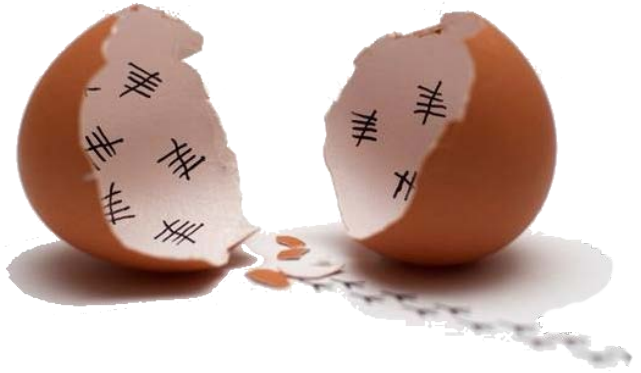
Questions About Design Patterns

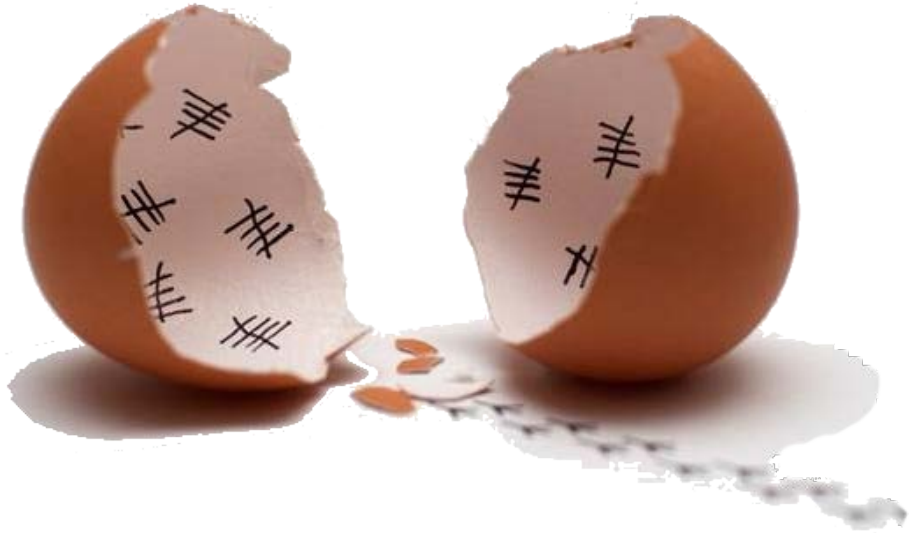
Why?

How?



When?





When?

- Apply the design pattern
  when the design is ready to accept it

Man not only reacts passively to incoming information, but creates intentions, forms plans and programs of his actions, inspects their performance, and regulates his behavior so that it conforms to these plans and programs; finally, he verifies his conscious activity, comparing the effects of his actions with the original intentions and correcting any mistakes he has made.

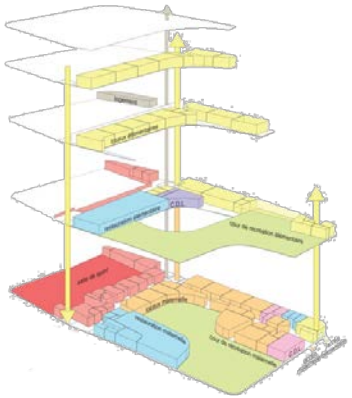
Alexander Luria, The Working Brain, Basic Books, New York 1973, pp. 79-80

Intention – Plan – Regulation – Verification

Requirements gathering

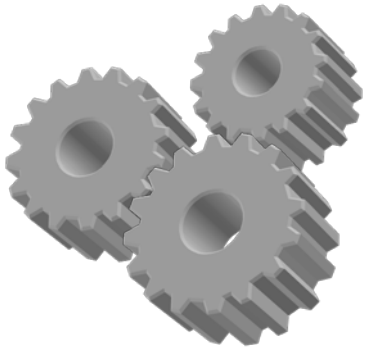
Intentions

Analysis



Plan

Implementation



Regulation

Acceptance testing

Verification

Let **not** your intentions sound like:
I want to implement this design pattern.

Instead, just think: **I want to solve this problem.**

Multiple Iterations

- The first attempt is to produce the tailored design
  - Design patterns do not fit into this phase
- Then refactor to reach a better design
  - This is where design patterns fit well

Summary

- Principal scheme is not the same as implementation of a design pattern
- Fundamental questions about design patterns
  - How and why – technical questions
  - When – teaches us how to be patient and not to apply the design pattern too early

Following Modules

- Cascading Factories to Eliminate Dependencies
- Real World Composition Pitfalls
- Compositing the Control Role
- Object Composition Using Chain of Responsibility
- Visitor Design Pattern and Encapsulation
- Calling Protocols and the Visitor
- Using Mixin to Move Responsibilities Out