Microsoft

Overview of Core Design Patterns

Crystal Tenn
Crystal.Tenn@microsoft.com

The Software Development Pyramid

Completeness

The design must cover as many important situations as is practical.

Consistency

The design should be consistent.

Simplicity

The design must be simple, both in implementation and interface. The primary job of an Architect is to take the complex and make it simple and easy to use/maintain. Simplicity is the single most important design criteria outside of correctness.

Correctness

The design must be correct in all observable aspects. **The "correctness" is verified by testing against defined requirements.**

Object Oriented Design Philosophy

Writing “SOLID” Code

Principles articulated circa 2000 by “Uncle Bob” (Robert C. Martin)

Michael Feathers is credited with the acronym

Each Principle Helps with OOD

S – Single Responsibility Principle

O – Open/Closed Principle

L – Liskov Substitution Principle

I – Interface Segregation Principle

D – Dependency Inversion Principle

DRY and YAGNI

Don’t Repeat Yourself. Duplicity in code is a sure way to fail.

You Ain’t Gonna Need It. Don’t borrow problems from the future. Solve today’s issues only.

DRY

- Stands for Don’t Repeat Yourself
- DRY aims to reducing repetition of information of all
- Kinds especially in multi-tiered architectures.
- “Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.”
- AKA “The Single Source of Truth” philosophy

DRY

1. Inadvertent Duplication
   - Developers ***don't realize*** that they are duplicating
   - functionality.
2. “Cutting-Corners” Duplication
   - Developers are ***pressured*** and duplication, it saves time.
3. Inter-developer Duplication
   - Multiple people on a team (or on different teams) duplicate a piece of information.

- Beware: Duplication can be subtle…
  - ***Identical code*** in different locations
  - ***Identical functionality*** in different locations
  - Either can lead to a ***maintenance nightmare!***

DRY

1. Inadvertent Duplication
   - Developers ***don't realize*** that they are duplicating
   - functionality.
2. “Cutting-Corners” Duplication
   - Developers are ***pressured*** and duplication, it saves time.
3. Inter-developer Duplication
   - Multiple people on a team (or on different teams) duplicate a piece of information.

- Beware: Duplication can be subtle…
  - ***Identical code*** in different locations
  - ***Identical functionality*** in different locations
  - Either can lead to a ***maintenance nightmare!***

YAGNI

- "You aren't gonna need it" is a principle that states a programmer should not add functionality until deemed necessary.
- XP co-founder Ron Jeffries has written: "Always implement things when you actually need them, never when you just foresee that you need them.“
- It may not be obvious why this is an issue until…

YAGNI

- The time spent is taken from adding, testing or improving the currently necessary functionality.
- The new features must be debugged, documented, and supported.
  - Microsoft must translate everything into eight languages for initial release
- New features impose constraints on what can be done in the future
- Until it's actually needed, it is difficult to fully define what a new feature should do and test it.
- This leads to code bloat and adds complexity that's not required.
- Plus, the feature may not be known to programmers who could make use of it.
- Adding the new feature may suggest other new features. If these new features are implemented as well, this could result in a snowball effect towards feature creep.

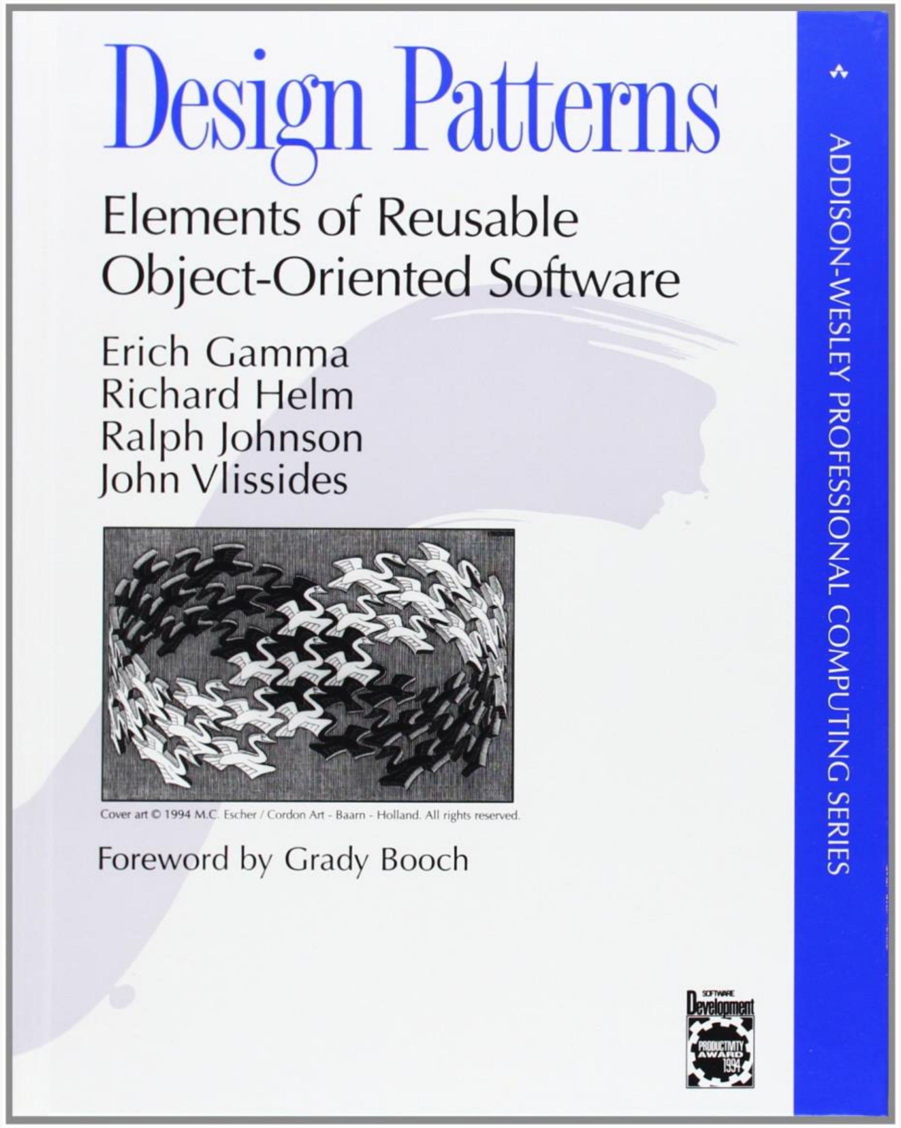
KISS

- Keep it simplestupid.
- “Simplicity is the ultimate sophistication” – Leonardo da Vinci
- Kelly Johnson (credited with the acronym) gave his team of design engineers a handful of tools, with the challenge that the jet aircraft they were designing must be repairable by an average mechanic, in the field, under combat conditions with only these tools.
- ‘Stupid' refers to the relationship between the way things break and the sophistication available to fix them

Let’s talk about the Gang of Four patterns

GoF patterns

- There are 23 of them (there are a lot of them and they are difficult to remember).
- They fit into 3 categories
- The 4 authors of the book "***Design Patterns: Elements of Reusable Object-Oriented Software***" are often referred to as the 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

Rubrics Used for Three Core Categories

Creational

Patterns **create** objects for you, rather than have you instantiate them directly. This gives you flexibility in deciding which objects need to be created in any given case and also enabled an interface-based-design.

Structural

Patterns concern class and object **composition**. The use inheritance to compose interfaces and define ways to compose objects to obtain new functionality.

Behavioral

Patterns allow objects to enable **complex** and **adaptable** responses, often through communication between objects.

THE 23 GANG OF FOUR DESIGN PATTERNS

C	Abstract Factory	S	Facade	S	Proxy
S	Adapter	C	Factory Method	B	Observer
S	Bridge	S	Flyweight	C	Singleton
C	Builder	B	Interpreter	B	State
B	Chain of Responsibility	B	Iterator	B	Strategy
B	Command	B	Mediator	B	Template Method
S	Composite	B	Memento	B	Visitor
S	Decorator	C	Prototype		

GoF patterns

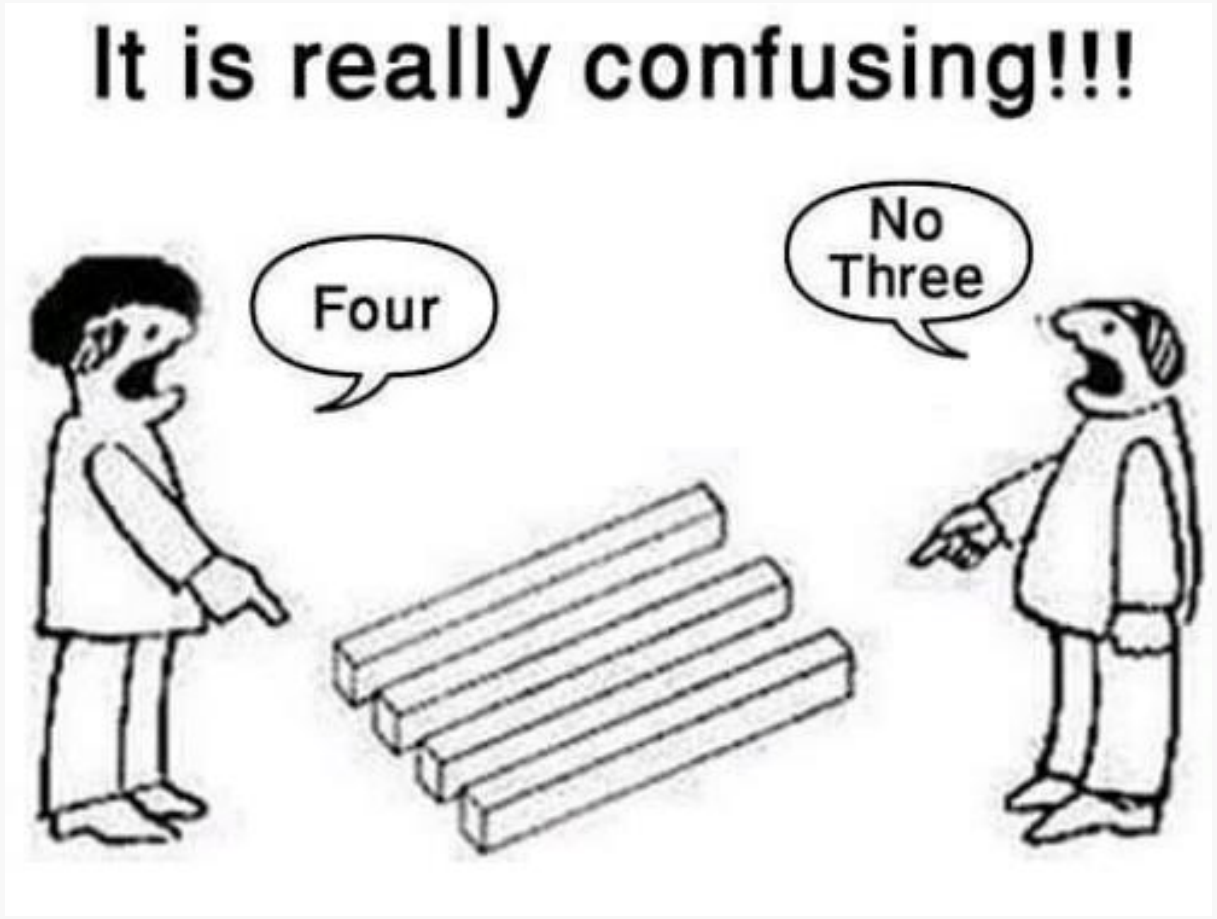
- Most commonly used: Abstract Factory, Factory, Façade, Iterator, Observer,

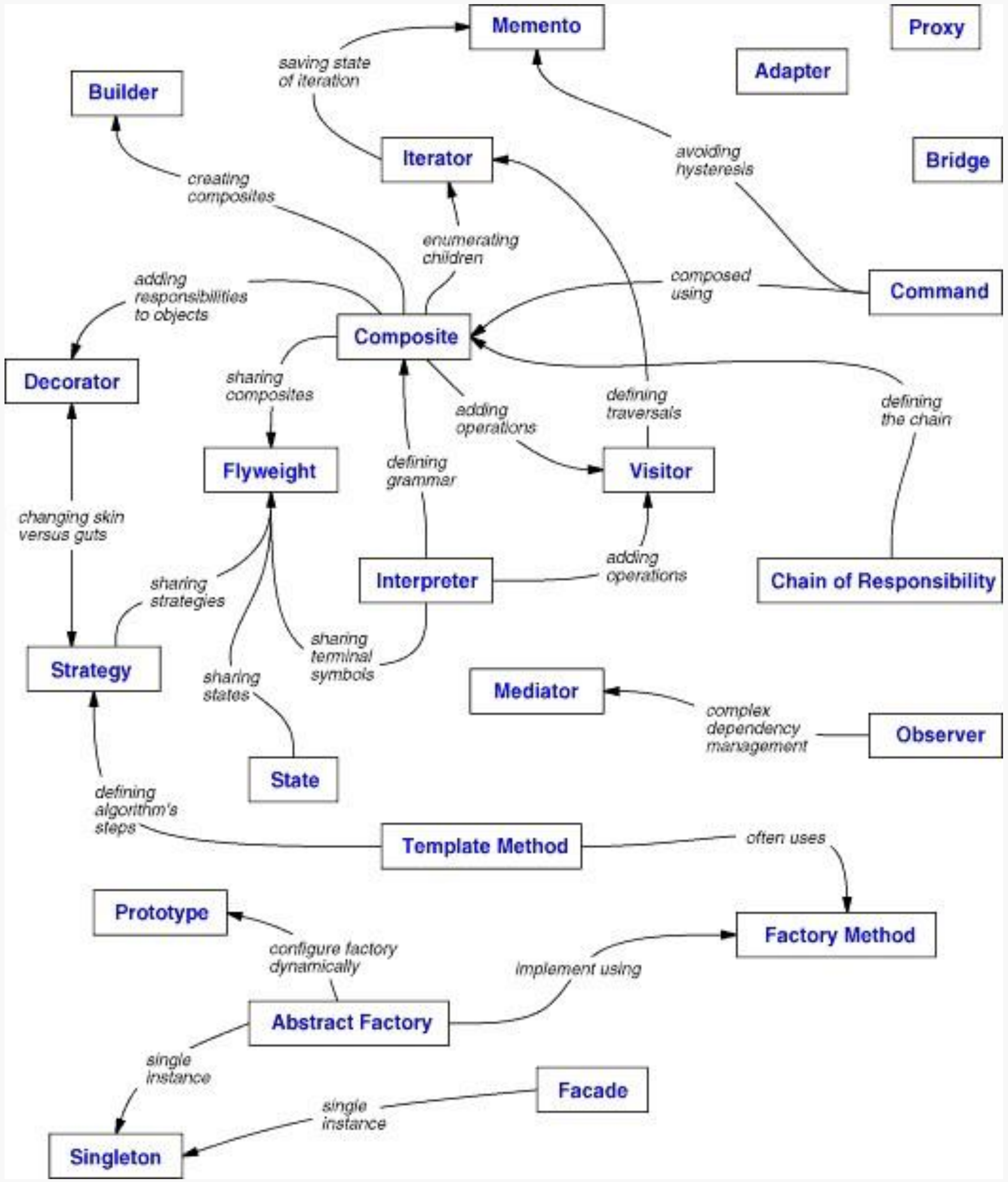
- Sometimes used: Singleton, Adapter, Composite, Proxy, Command, Strategy

**hint, remember all of these above!!

- Least commonly used: Flyweight, Interpreter, Memento, Visitor.

**hint.. Small chance of seeing these.




Memento
Proxy
Adapter
Builder
saving state of iteration
Iterator
avoiding hysteresis
Bridge
creating composites
enumerating children
adding responsibilities to objects
composed using
Command
Composite
Decorator
sharing composites
adding operations
defining traversals
defining the chain
Flyweight
defining grammar
Visitor
changing skin versus guts
sharing strategies
Interpreter
adding operations
Chain of Responsibility
sharing terminal symbols
Strategy
sharing states
Mediator
complex dependency management
Observer
State
defining algorithm's steps
Template Method
often uses
Prototype
configure factory dynamically
implement using
Factory Method
Abstract Factory
single instance
Facade
single instance
Singleton

Lab

- 01 Lab – Core Design Patterns.docx
- 01 Core Design Patterns [begin]

Microsoft

Thank you!