Mubarak

Architecture Design vs Implementation Design

Quality
Excellent
Good
Average
Poor

Quality

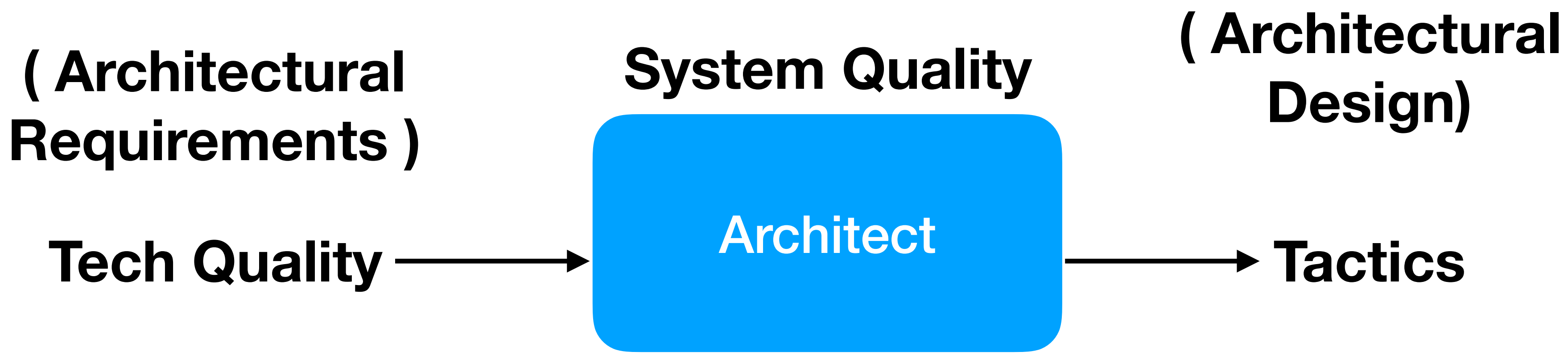
1. Cost
2. Time

Tech Quality

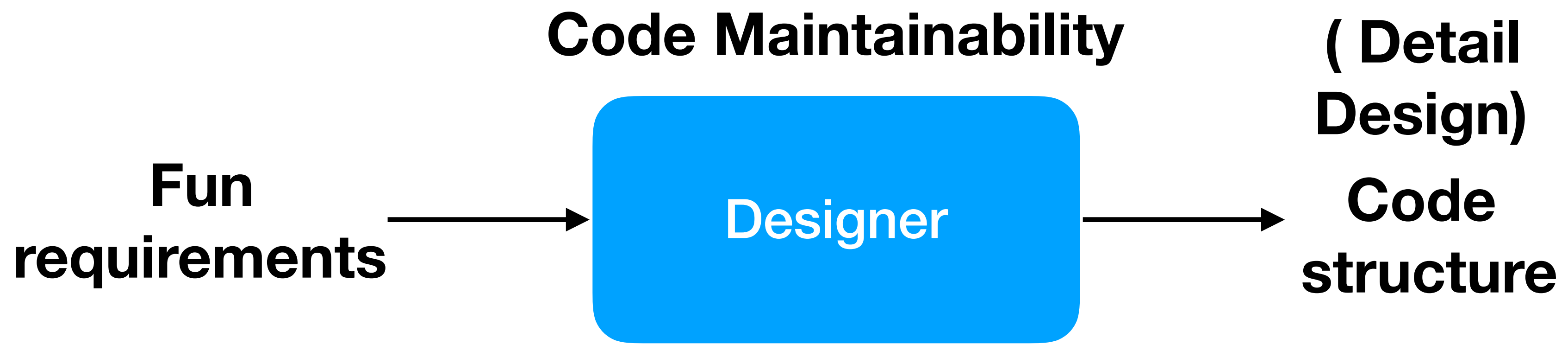
1.Performance (cpu,memory,I/O, …)
2.Maintainablity
3.Scalability (volume- cpu, memory,I/O,…)
4.Security (Trustability)
5.Usability
6.Reliability (Trustability)
7.Availability
8.Robustnes (Rugud)
9. Portability
10.Interoperability

Tactics

1. Reduce memory foot print
2. Extensibile, readability,log, Testability
3. Authentication,Audit
4. ACID - Transaction
5. Input validation
6. Parallel
7.Caching
8. Lazy loading
9.



- Arch Principles
- Arch Patterns
- Ref Arch
- …



- Design Principles
- OO Design Patterns
- Fun design Patterns
- Anti Patterns
- Idioms
- …

Java / py/ C++/ JS/

	Procedural	OO	Functional
Performance	n/a	n/a	
Security	n/a	n/a	n/a
Testability	1	2	
Manage code Complexity	1	3	
Learning Curve	3	1	
Time to develop	3	1	

OO => Manage Code Complexity

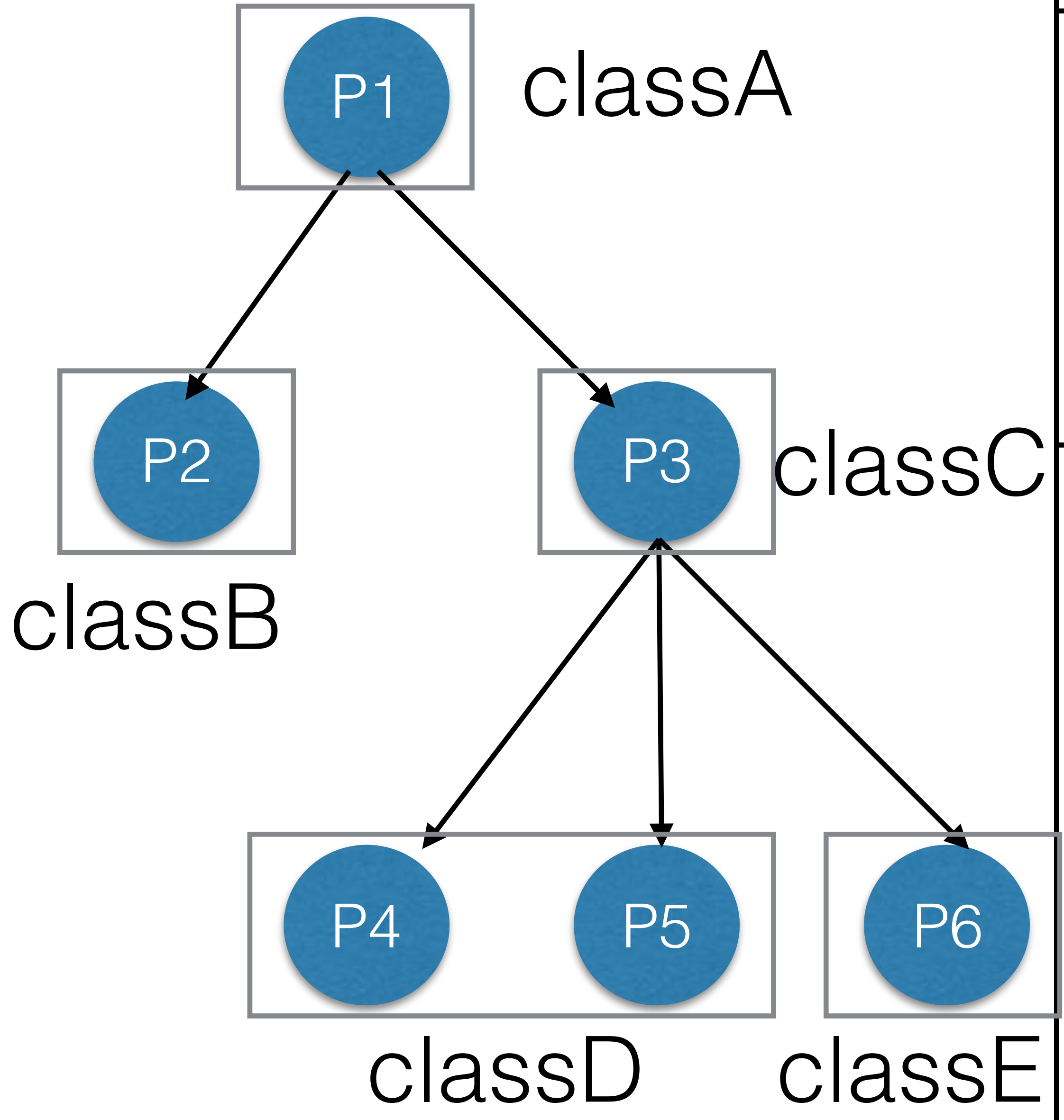
```
Interface  Bird
{
  fly();
  buildNest();
  layEggs();
  sing();
}
```

```
Interface  Bird
{
  eat()
}
```

```
fun(Bird bird)
{
   //logic
}
```

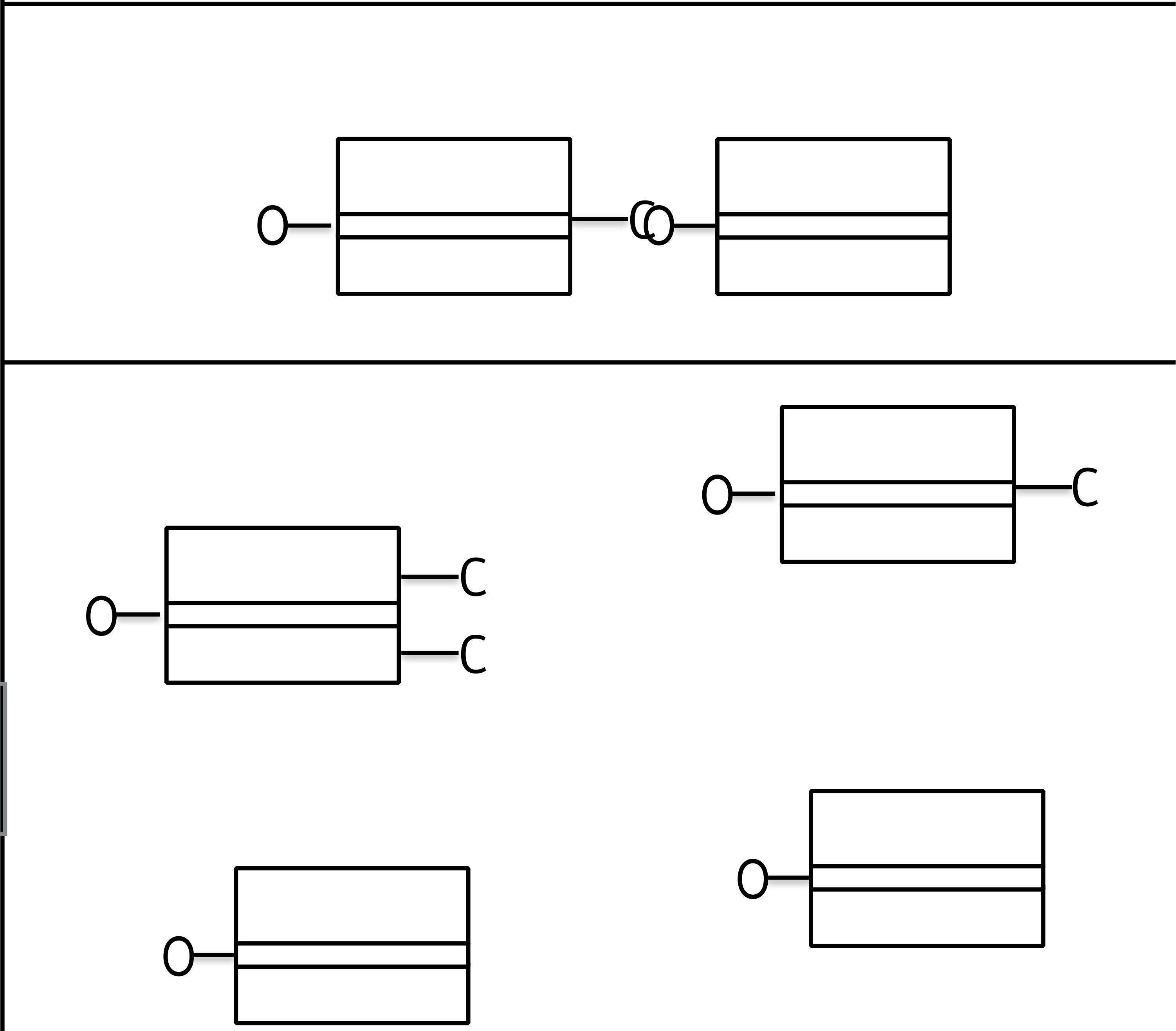
Procedural Prog

(tree)



OO Prog

(Lego)



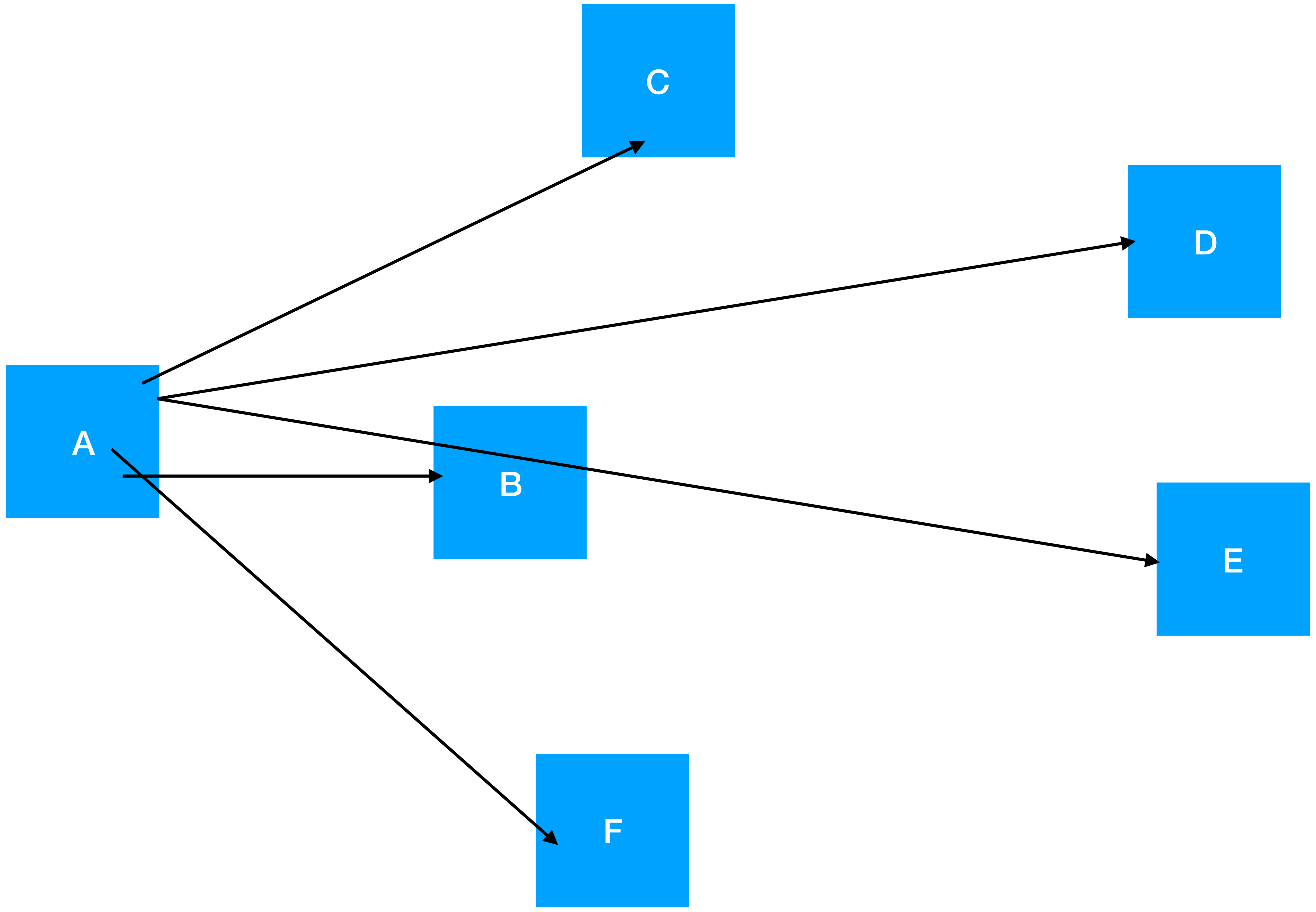
If ==> polymorphism (interface)
“Things which don’t change together should not be kept together”
“Bank”

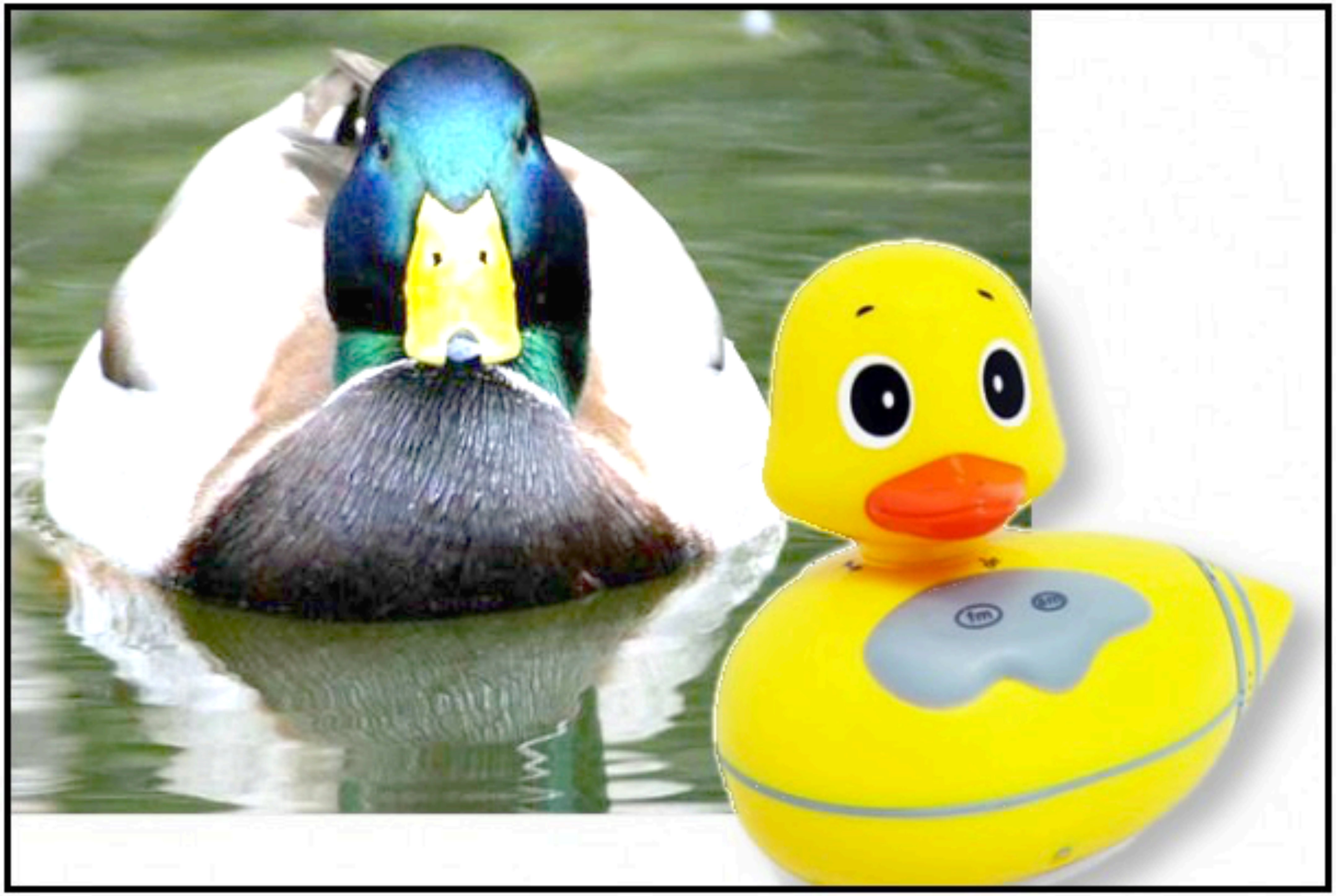
getmub@gmail.com

Design Check list

+ LSP
+ SRP (*)
    # things which don’t change together
    #class size
        $ Avg: 5 interface methods
        $ Max: 12
+ Low Coupling (*)
+ Exceptions
+ DRY (*)

- Flag
- bool/null/int for error handling
- Static Methods
- Swiss Knife/ God Class (Util,Controller, Helper, Provider, Handler,Activity, Manager, Processor, Module, …)


C
D
A
B
E
F

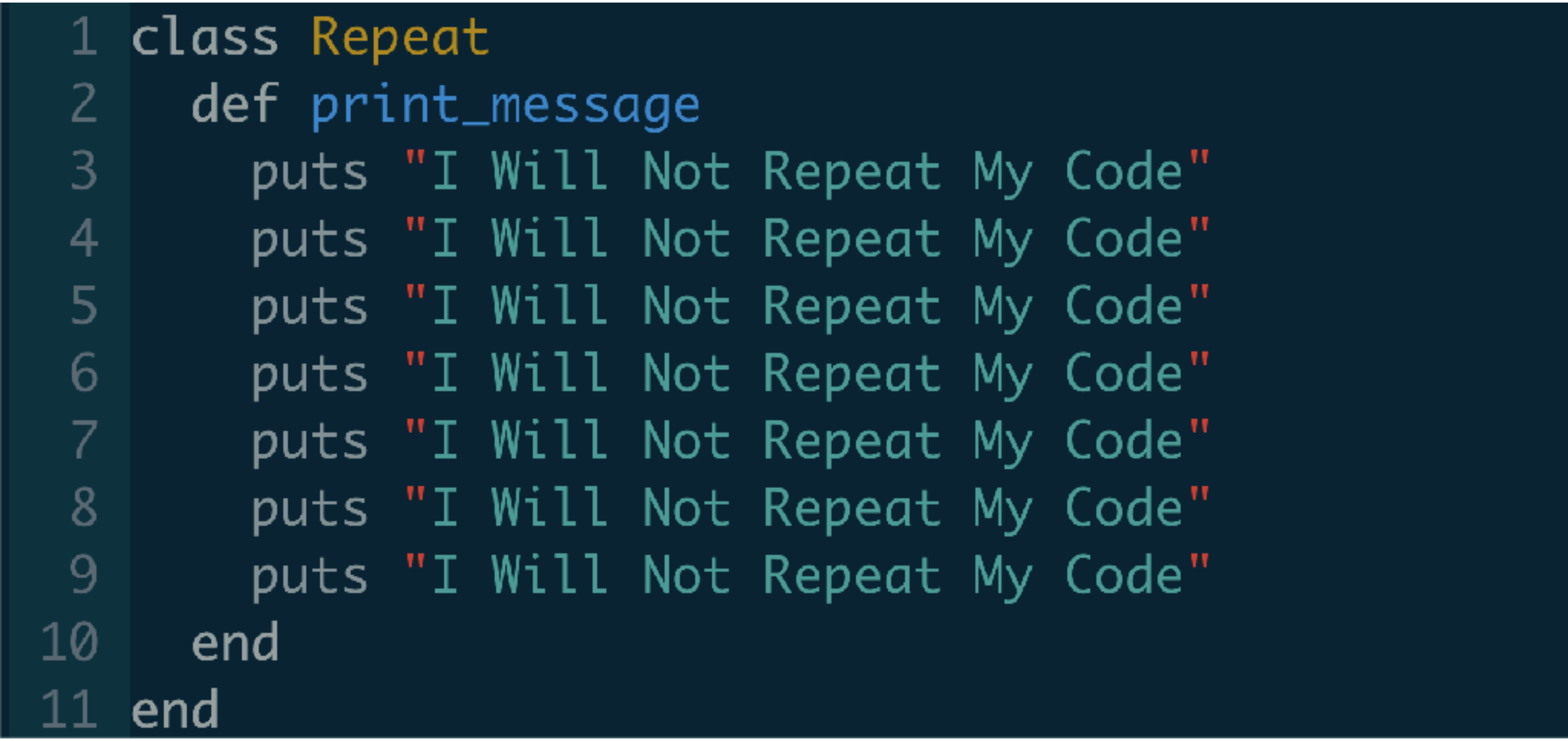


LISKOV SUBSTITUTION PRINCIPLE

If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

SINGLE RESPONSIBILITY PRINCIPLE

Every object should have a single responsibility, and all its services should be narrowly aligned with that responsibility.



```ruby
class Repeat
  def print_message
    puts "I Will Not Repeat My Code"
    puts "I Will Not Repeat My Code"
    puts "I Will Not Repeat My Code"
    puts "I Will Not Repeat My Code"
    puts "I Will Not Repeat My Code"
    puts "I Will Not Repeat My Code"
    puts "I Will Not Repeat My Code"
  end
end
```

Software Engineering v/s Tuning

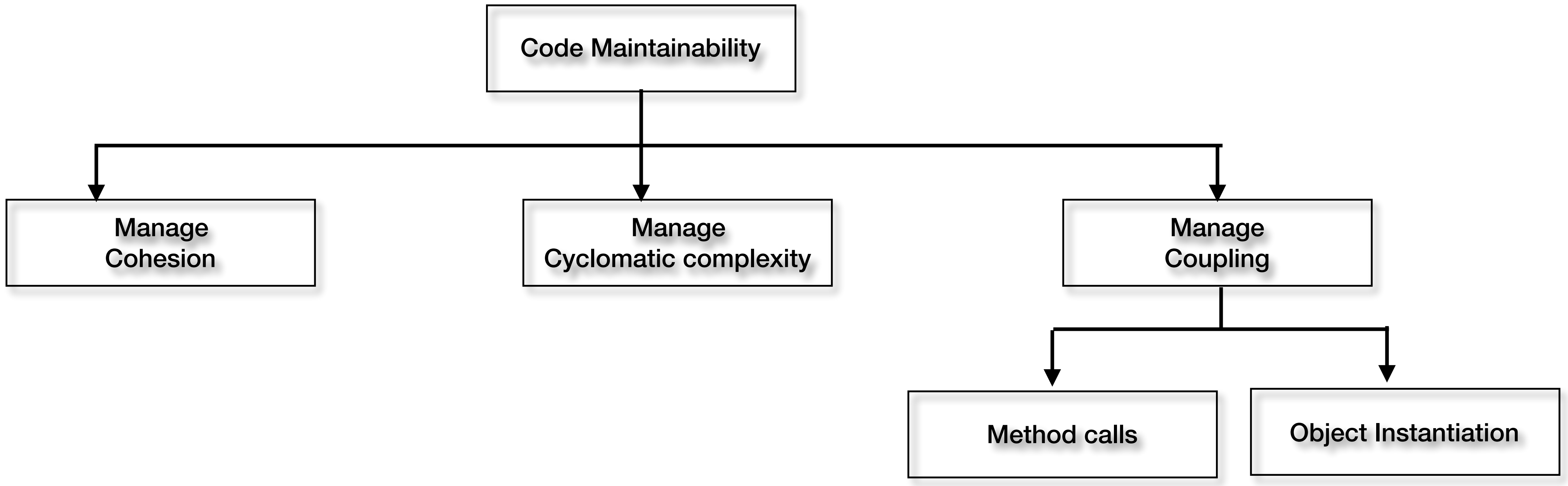
Quality

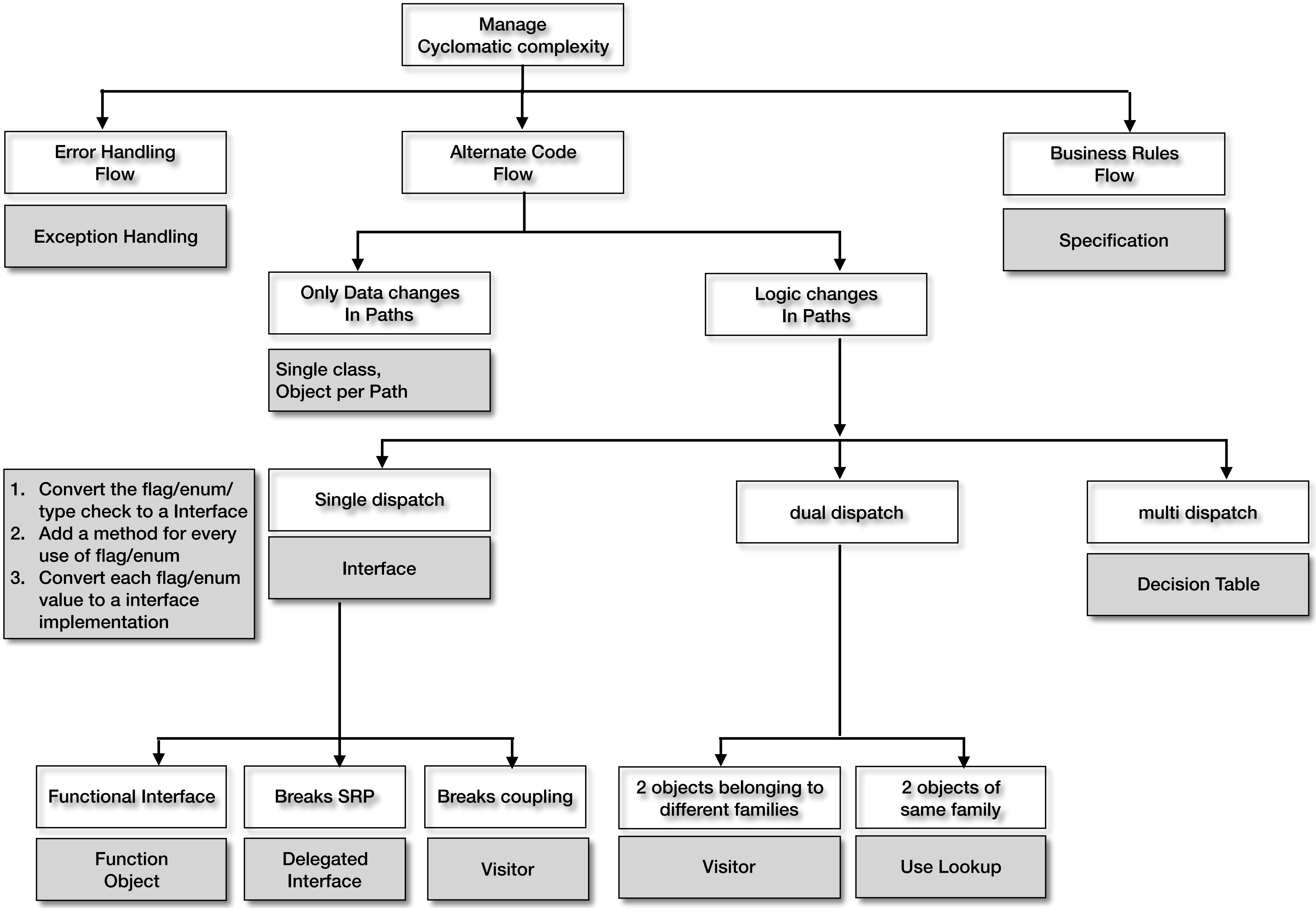
# Performance Engineering

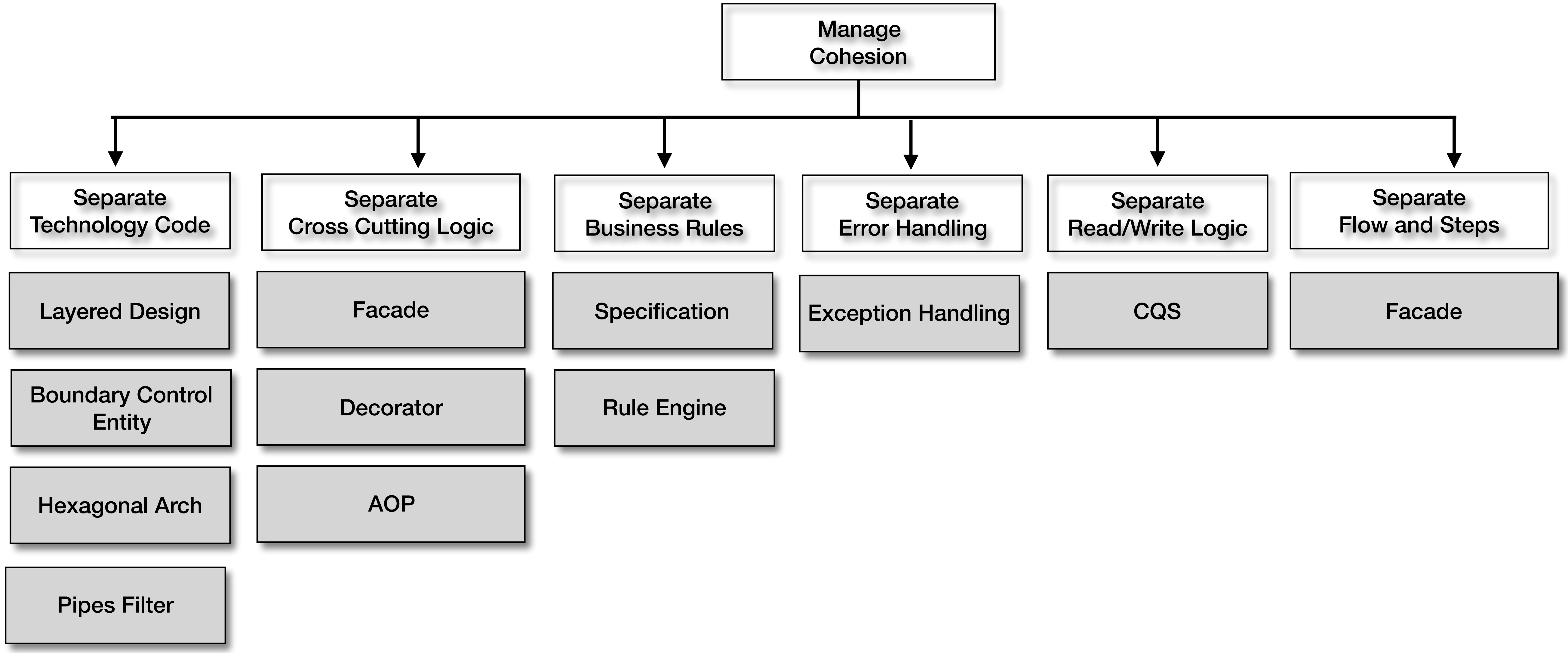
# Performance Tuning

# Threat Modeling

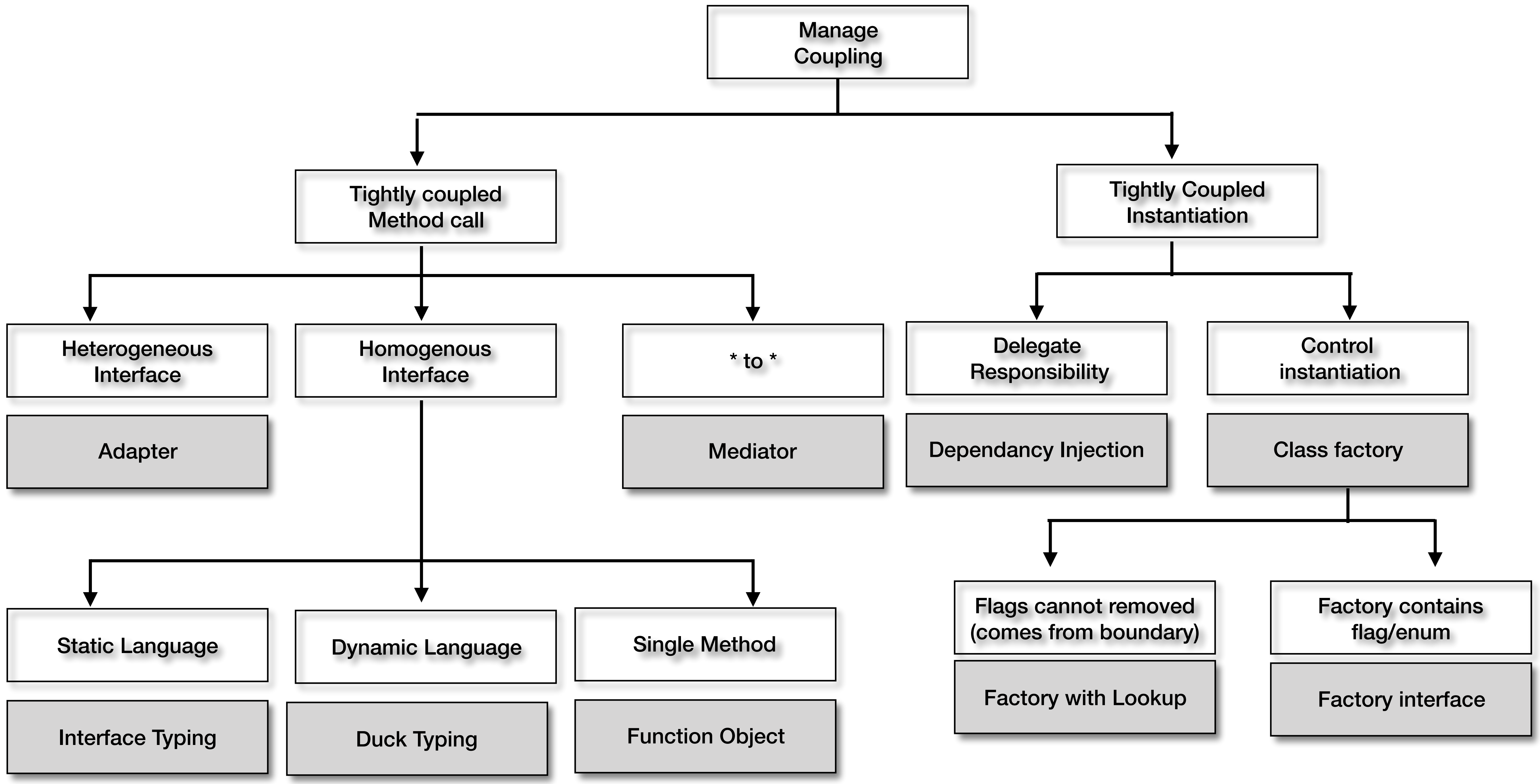
Ethical hacking

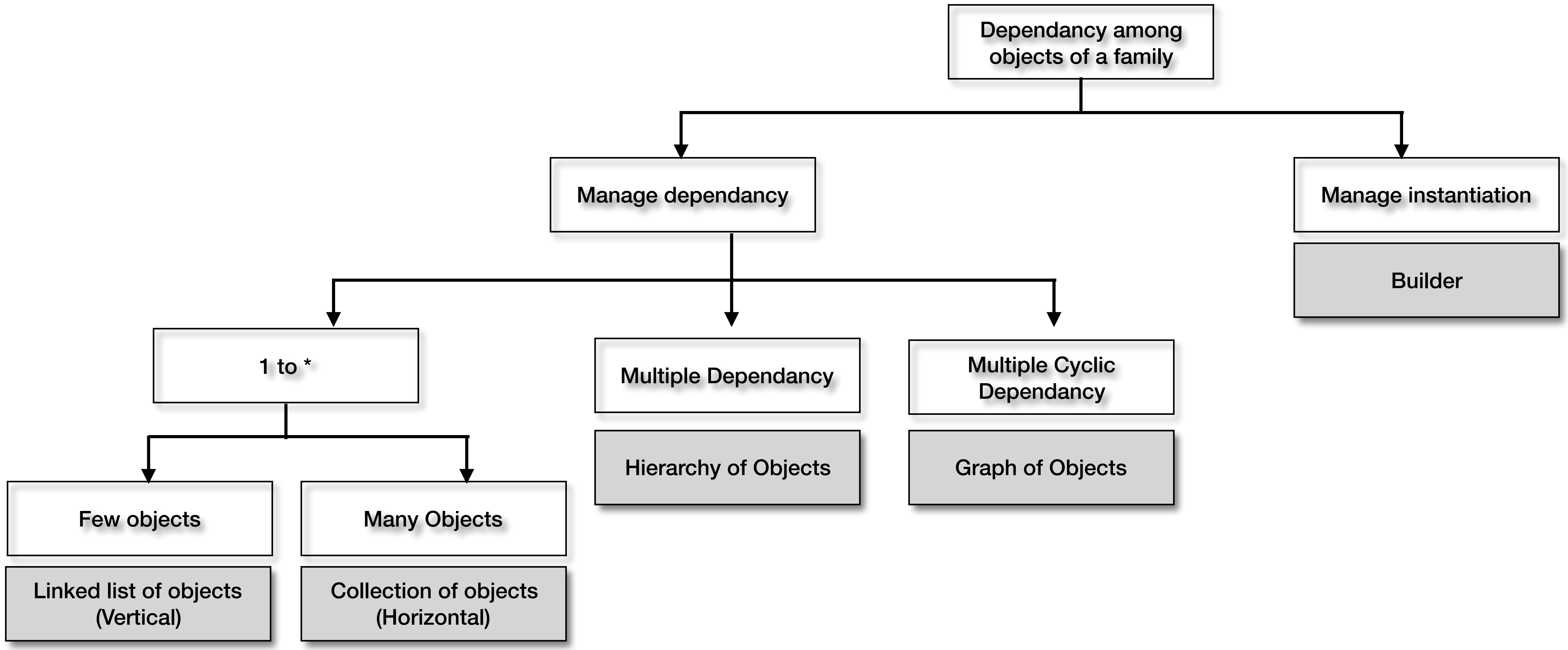

Code Maintainability
Manage Cohesion
Manage Cyclomatic complexity
Manage Coupling
Method calls
Object Instantiation


Manage Cyclomatic complexity
Error Handling Flow
Exception Handling
Alternate Code Flow
Business Rules Flow
Specification
Only Data changes In Paths
Single class, Object per Path
Logic changes In Paths
1. Convert the flag/enum/ type check to a Interface
2. Add a method for every use of flag/enum
3. Convert each flag/enum value to a interface implementation
Single dispatch
Interface
dual dispatch
multi dispatch
Decision Table
Functional Interface
Function Object
Breaks SRP
Delegated Interface
Breaks coupling
Visitor
2 objects belonging to different families
Visitor
2 objects of same family
Use Lookup



Manage Cohesion
Separate Technology Code
Separate Cross Cutting Logic
Separate Business Rules
Separate Error Handling
Separate Read/Write Logic
Separate Flow and Steps
Layered Design
Boundary Control Entity
Hexagonal Arch
Pipes Filter
Facade
Decorator
AOP
Specification
Rule Engine
Exception Handling
CQS
Facade


Manage Coupling
Tightly coupled Method call
Tightly Coupled Instantiation
Heterogeneous Interface
Homogenous Interface
* to *
Delegate Responsibility
Control instantiation
Adapter
Mediator
Dependancy Injection
Class factory
Static Language
Dynamic Language
Single Method
Flags cannot removed (comes from boundary)
Factory contains flag/enum
Interface Typing
Duck Typing
Function Object
Factory with Lookup
Factory interface


Dependancy among objects of a family
Manage dependancy
Manage instantiation
Builder
1 to *
Multiple Dependancy
Multiple Cyclic Dependancy
Hierarchy of Objects
Graph of Objects
Few objects
Many Objects
Linked list of objects (Vertical)
Collection of objects (Horizontal)