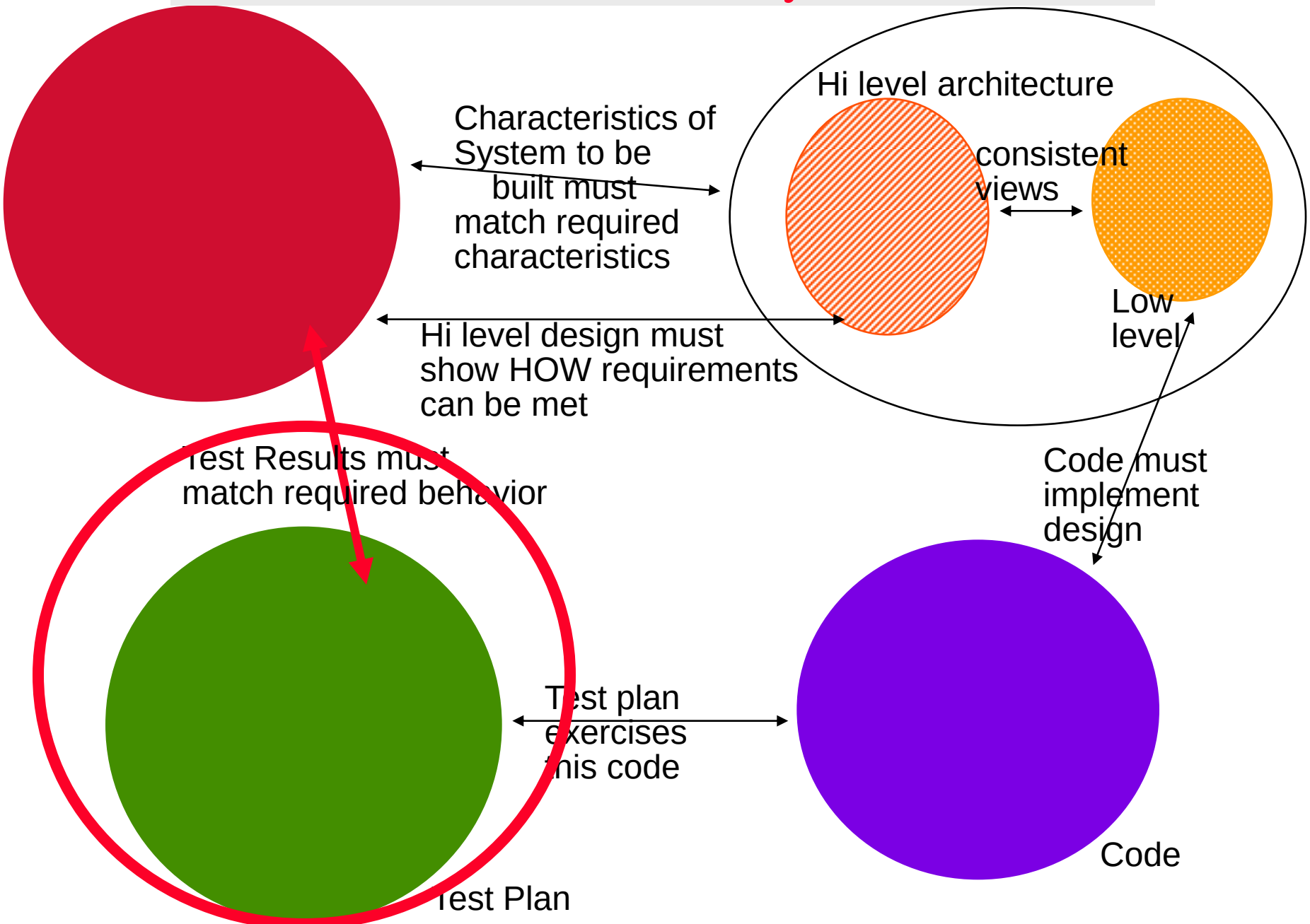
Architecture

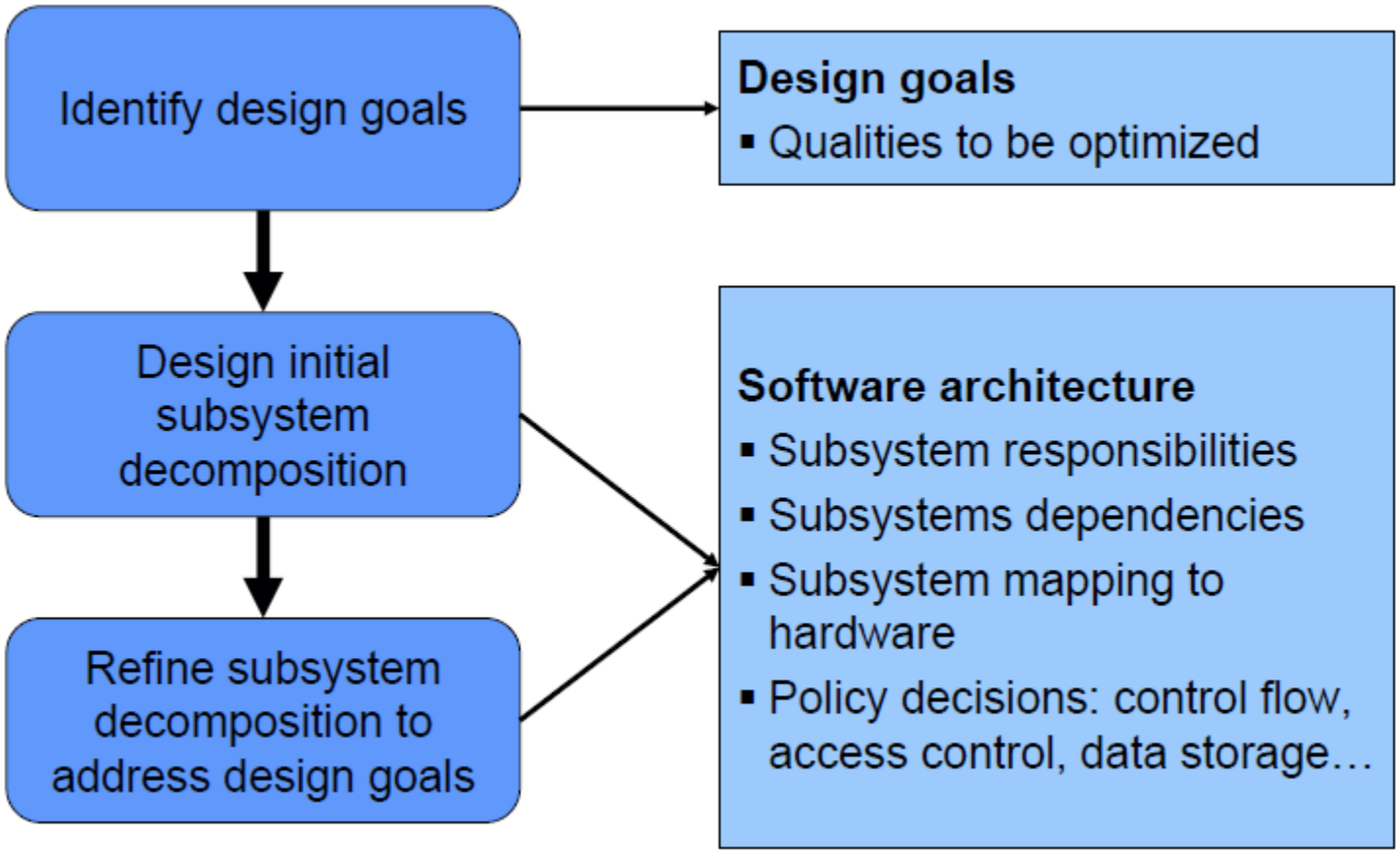
Reda Bendraou
reda.bendraou{{@}}Lip6.fr
http://pagesperso-systeme.lip6.fr/Reda.Bendraou/

Some slides were adapted from L. Osterweil, B. Meyer, and P. Müller material

Focus on "How do you know"



Software Architecture & Design: Goals



From B. Meyer and P. Muller

Why decompose a system?

Management
- Partition effort
- Clear assignment of requirements to modules

Modification
- Decouple parts so that changes to one don’t affect others

Understanding
- Allow understanding system one chunk at a time

What is the Nature of Design?

Addresses the question: HOW?"

• **Goal**: Indicate how to develop a solution system that will satisfy requirements"

• Complements:

– Requirements: "WHAT"

– System Test Plan: HOW WOULD I KNOW IT IF I SAW IT "

How are Designs Represented?"

Familiar approaches
– Use of hierarchy to conquer size/complexity
– Use of multiple views to capture different aspects
– Use of pictures and diagrams to appeal to nontechnical stakeholders

Connected to requirements elements they respond to

Connected to code elements that implement them

Architecture Vs. Specification

Architecture

– High level system design
– Concerned with components and the *interactions* among components
– Not with the algorithms or data structures

Specification (Low Level Design)

– Emphasis on data structures and algorithms
– Focus on implementation issues
  » Stepwise refinement
  » Evolvability
  » Use of abstraction

Typical Architecture Issues

Component interaction models
– What are the components' interfaces?
– Who can use them? And how?

How much flexibility is achievable? How modifiable?
– Is plug and play possible?

Where is network access used? How?
– Message passing, broadcasting, etc?

Late-binding issues
– Non-determinism
– Use of proxies

New issues in characterizing system objects

Interaction protocols
- Tightly coupled objects
  - » Direct or Remote procedure calls
- Loosely coupled
  - » Event based notification, observers

Degree of separation
- Locally
- Internet scale
- “in the cloud”

Modes of communicating with each other
- message passing
- broadcast
- multi-cast

Architecture description (specification or design)

A high level design that defines the components, connectors, constraints and the interrelationships among these entities

- Usually compositional

Suggests the value of elaborate semantics and annotations of the nodes and edges

Many notations can be used.

- **UML has a bunch of diagrams to represent Architecture's aspects**
- **Component, Class, Package, Deployment diagrams**

Components, Connectors, Constraints: Central Software Architecture Entities

Components--computational units
– Subsystems
– Classes
– Objects

Connectors--interaction model
– Which components are connected to which?
– How are they connected?
– Are connectors just components with restricted semantics?

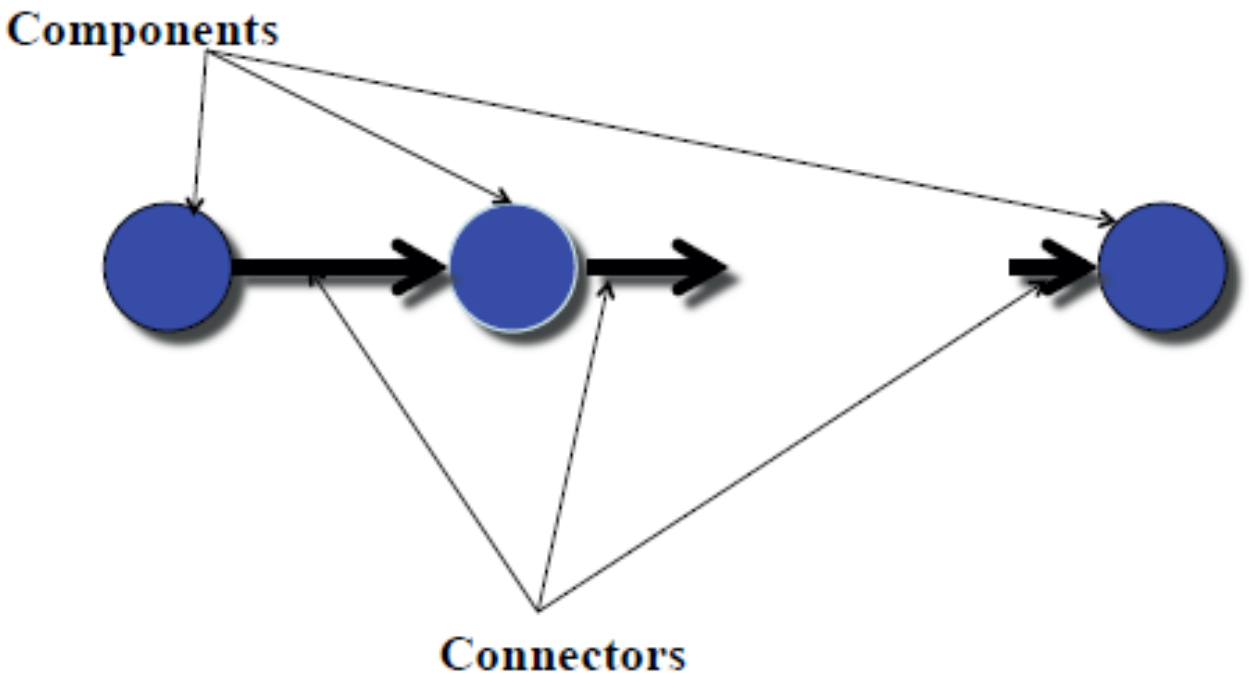
Constraints
– Guides and limits to the ways components and connectors can be configured

Architectural Styles

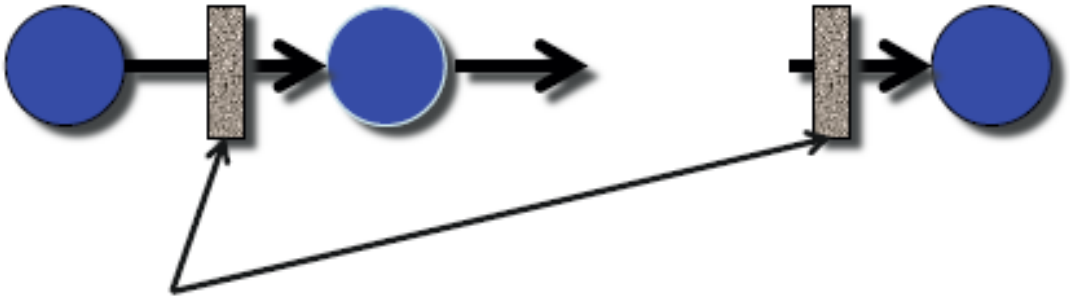
Sets of constraints that are widely used because they offer understood capabilities and features

- Data flow systems
  - Batch sequential, pipe-and-filter
- Call-and-return system
  - Main program and subroutine
- Independent components
  - Interacting processes, event system
- Data-centered systems (repositories)
  - Databases, blackboards
- Hierarchical systems
  - Layers
  - Interpreters, rule-based systems
- Client-server systems
- Peer-to-peer systems

Pipeline Architecture:Each component has one input connector, one output connector



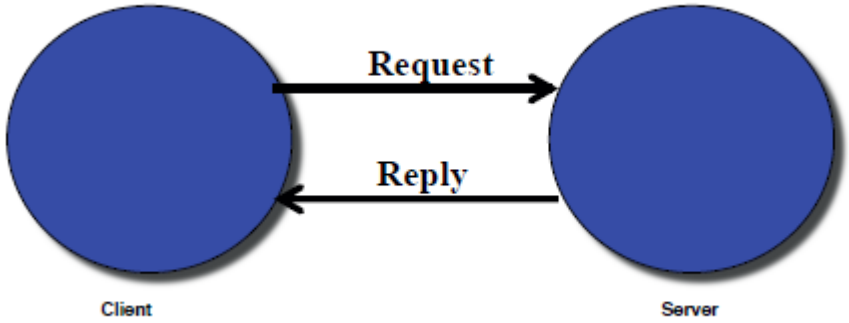
Pipe and Filter: Pipeline architecture where some connectors have a “filter”



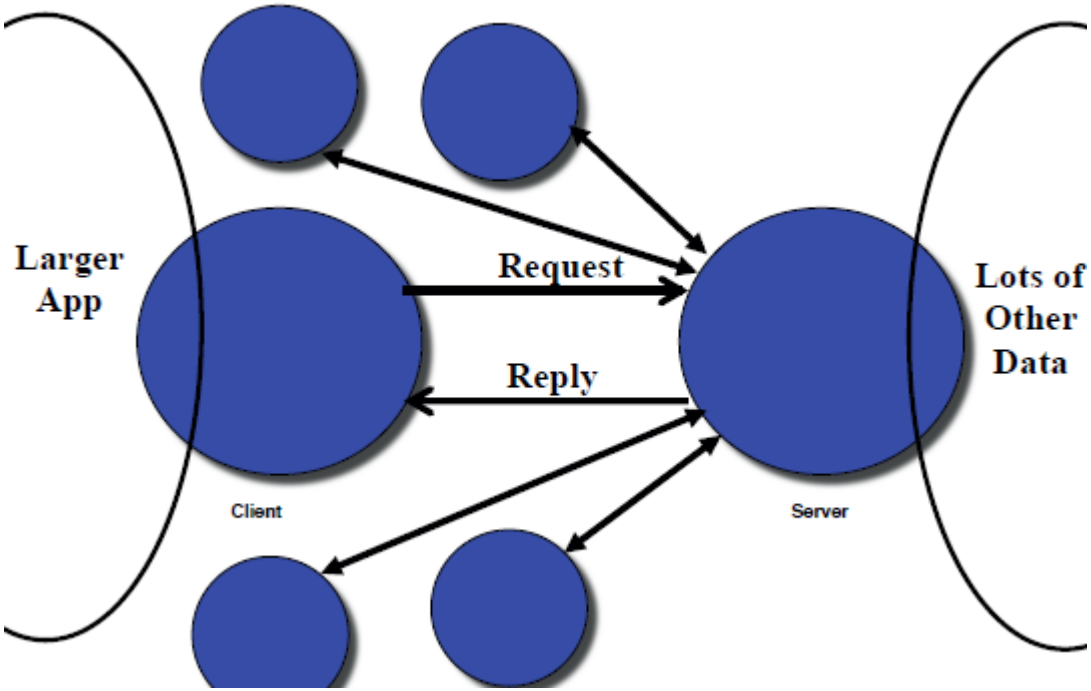
Filters:

Components that have particular properties
(they “filter” the data moving thru the connectors)

Client/Server



Client/Server : multiple clients



Need to specify details

- What will a request look like?
- What will a reply look like?
- How will multiple simultaneous requests be served?
- Any constraints on requests, replies?
  - **E.g. speed**

Different Substyles

• How to specify different ways for client/server to perform

• REST Architecture
– Server is “stateless”
– No memory of details of client
– A key property that www infrastructure is built upon

Service Oriented Architecture

We will address this in a specific lecture

- Applications composed from components

- Components are accessed via the Web
  – Specified generically (as a “service”)"
  – Located by web searches (using proxies)
  – Accessed via the web

- How to compose such services?
  – What composition constructs

- How to be sure they provide correct services?

- How to maintain privacy and security?

Service Oriented Architecture

Variants
SaaS
PaaS
IaaS
….

Cloud Architecture

• SOA approach, but
– Don’t know/don’t care where or how services are provided via the Web

• Service may be different each time the system runs

• Similar problems, but now more worrisome
– Correctness
– Security
– Privacy

Web Based Applications

Presented in more details on class (on the white board)

MVC

Presented in more details on class (on the white board)

Some important Concerns about Architecture

Modularity

Cohesion Vs Coupling

Cohesion: **interdependence** of elements of **one module**

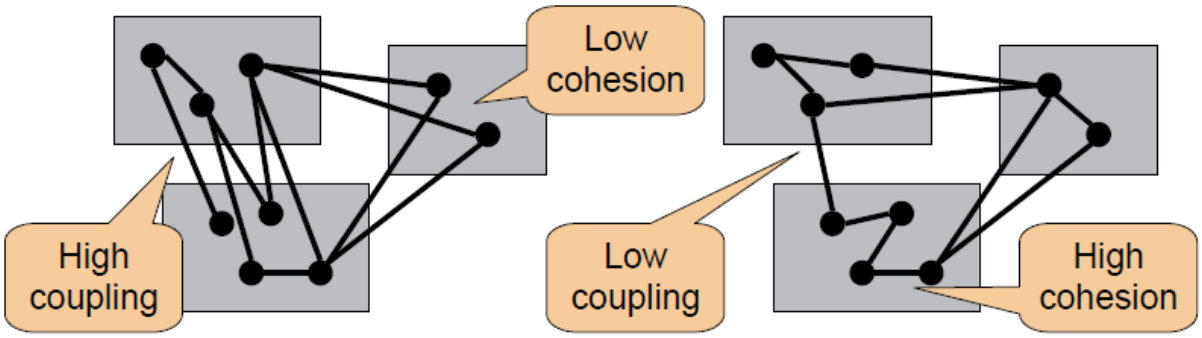
Each subsystem has a clear responsibility

Coupling: **interdependence between** different **modules**

Small interfaces between subsystems

Goal: **high cohesion** and **low coupling**

Modularity: increase cohesion, decrease coupling



From B. Meyer and P. Muller

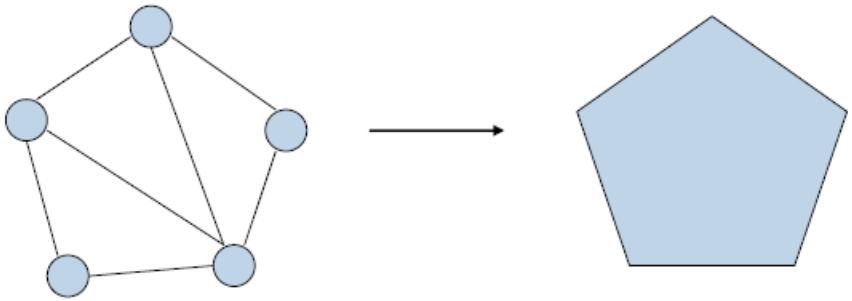
Cohesion Vs Coupling in OO design

- **Cohesion**
- Features work on same data
- Implement one ADT

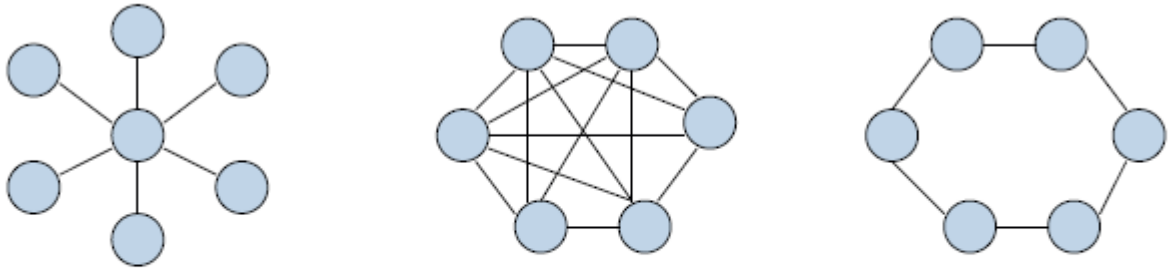
- **Low coupling**
- Small interfaces
- Information hiding
- No global data
- Interactions are within subsystem rather than across subsystem boundaries

Composability

Build software elements so that they may be freely combined with others to produce new software.



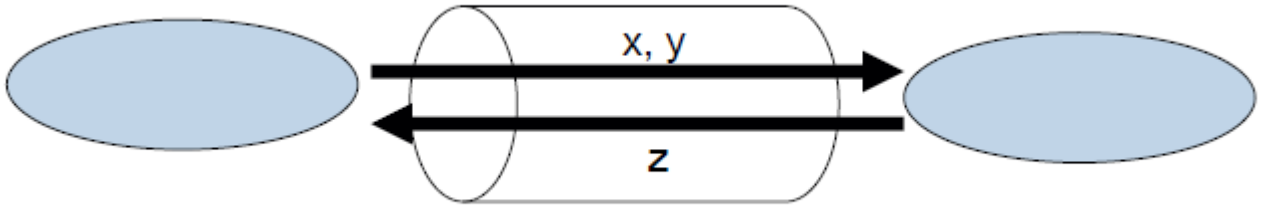
Every module communicates with as few others as possible.



(A) (B) (C)

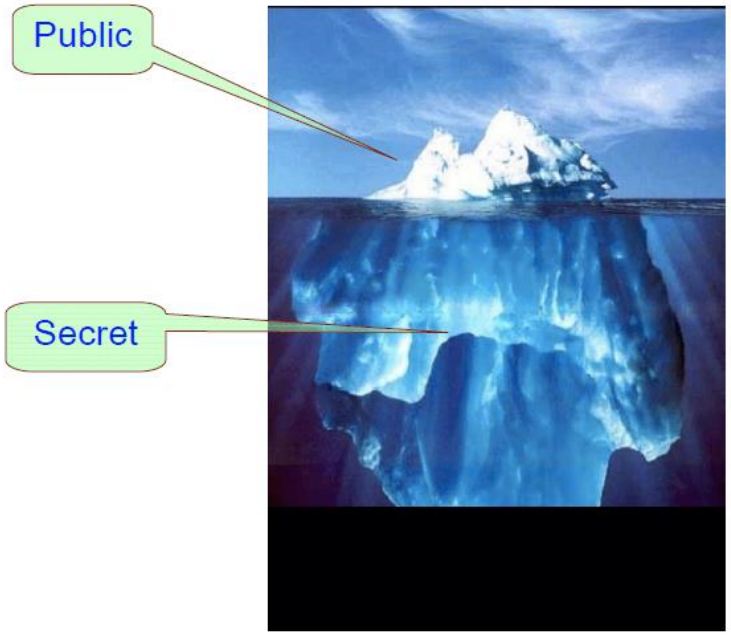
If two modules communicate, they exchange as little information as possible.

If two modules communicate, they exchange as little information as possible.



Information Hiding Principle

The designer of every module must select a subset of the module's properties as the official information about the module, to be made available to authors of client modules.



Design Patterns

Use them when possible

GoF book is a good reference but many other patterns exist in literature

Patterns can be in all phases of developmment

- Requirement patterns
- Architectural patterns
- Design patterns
- etc.

The five secrets of good architecture

- Simplicity of design
- Consistency of design
- Ease of learning of the APIs
- Support for change
- Support for reuse

Simplicity: always remember !!!

There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies.

Tony Hoare

C.A.R. Hoare
The Emperor's Old Clothes
1980 Turing Award lecture
http://tinyurl.com/3yk3v2

Conclusion

Design goals definition
- Describes and prioritizes the qualities that are important for the system

Subsystem decomposition
- Decomposes the overall system into manageable parts by using the principles of cohesion and coherence

Architectural style
- A pattern of a typical subsystem decomposition

Software architecture
- An instance of an architectural style