Design

Software Design

- Design activity begins with a set of requirements, and maybe an architecture
- Design done before the system is implemented
- Design focuses on module view – i.e. what modules should be in the system
- Module view may have easy or complex relationship with the C&C view
- Design of a system is a blue print for implementation
- Often has two levels – high level (modules are defined), and detailed design (logic specified)

Design…

- Design is a creative activity
- Goal: to create a plan to satisfy requirements
- Perhaps the most critical activity during system development
- Design determines the major characteristics of a system
- Has great impact on testing and maintenance
- Design document forms reference for later phases
- Design methodology – systematic approach for creating a design

Design Concepts

- Design is correct, if it will satisfy all the requirements and is consistent with architecture
- Of the correct designs, we want *best design*
- We focus on modularity as the main criteria (besides correctness)

Modularity

- Modular system – in which modules can be built separately and changes in one have minimum impact on others
- Modularity supports independence of models
- Modularity enhances design clarity, eases implementation
- Reduces cost of testing, debugging and maintenance
- Cannot simply chop a program into modules to get modularly
- Need some criteria for decomposition – coupling and cohesion are such criteria

Coupling

- Independent modules: if one can function completely without the presence of other
- Independence between modules is desirable
  - Modules can be modified separately
  - Can be implemented and tested separately
  - Programming cost decreases
- In a system all modules cannot be independent
- Modules must cooperate with each other
- More connections between modules
  - More dependent they are
  - More knowledge about one module is required to understand the other module.
- Coupling captures the notion of dependence

Coupling…

- Coupling between modules is the strength of interconnections between modules
- In general, the more we must know about module A in order to understand module B the more closely connected is A to B
- "Highly coupled" modules are joined by strong interconnection
- "Loosely coupled" modules have weak interconnections

Coupling…

- Goal: modules as loosely coupled as possible
- Where possible, have independent modules
- Coupling is decided during high level design
- Cannot be reduced during implementation
- Coupling is inter-module concept
- Major factors influencing coupling
  - Type of connection between modules
  - Complexity of the interface
  - Type of information flow between modules

Coupling – Type of connection

- Complexity and obscurity of interfaces increase coupling
- Minimize the number of interfaces per module
- Minimize the complexity of each interface
- Coupling is minimized if
  - Only defined entry of a module is used by others
  - Information is passed exclusively through parameters
- Coupling increases if
  - Indirect and obscure interface are used
  - Internals of a module are directly used
  - Shared variables employed for communication

Coupling – interface complexity

- Coupling increases with complexity of interfaces eg. number and complexity of parms
- Interfaces are needed to support required communication
- Often more than needed is used eg. passing entire record when only a field is needed
- Keep the interface of a module as simple as possible

Coupling – Type of Info flow

- Coupling depends on type of information flow
- Two kinds of information: data or control.
- Transfer of control information
  - Action of module depends on the information
  - Makes modules more difficult to understand
- Transfer of data information
  - Module can be treated as input-output function
- Lowest coupling: interfaces with only data communication
- Highest: hybrid interfaces

Coupling - Summary

Coupling	Interface complexity	Type of connections	Type of commu-nication
Low	Simple obvious	to module by name	data
High	complicated obscure	to internal elements	Hybrid

Coupling in OO Systems

- In OO systems, basic modules are classes, which are richer than fns
- OO Systems have three types of coupling
  - Interaction coupling
  - Component coupling
  - Inheritance coupling

Coupling in OO - Interaction

- Interaction coupling occurs due to methods of a class invoking methods of other classes
  - Like calling of functions
  - Worst form if methods directly access internal parts of other methods
  - Still bad if methods directly manipulate variables of other classes
  - Passing info through tmp vars is also bad

Coupling in OO …

- Least interaction coupling if methods communicate directly with parameters
  - With least number of parameters
  - With least amount of info being passed
  - With only data being passed
- I.e. methods should pass the least amount of data, with least no of parms

Coupling in OO - Component

- Component coupling – when a class A has variables of another class C
  - A has instance vars of C
  - A has some parms of type C
  - A has a method with a local var of type C
- When A is coupled with C, it is coupled with all subclasses of C as well
- Component coupling will generally imply the presence of interaction coupling also

Coupling in OO - Inheritance

- Inheritance coupling – two classes are coupled if one is a subclass of other
- Worst form – when subclass modifies a signature of a method or deletes a method
- Coupling is bad even when same signature but a changed implementation
- Least, when subclass only adds instance vars and methods but does not modify any

Cohesion

- Coupling characterized the inter-module bond
- Reduced by minimizing relationship between elts of different modules
- Another method of achieving this is by maximizing relationship between elts of same module
- Cohesion considers this relationship
- Interested in determining how closely the elements of a module are related to each other
- In practice both are used

Cohesion…

- Cohesion of a module represents how tightly bound are the elements of the module
- Gives a handle about whether the different elements of a module belong together
- High cohesion is the goal
- Cohesion and coupling are interrelated
- Greater cohesion of modules, lower coupling between module
- Correlation is not perfect.

Levels of Cohesion

- There are many levels of cohesion.
  - Coincidental
  - Logical
  - Temporal
  - Communicational
  - Sequential
  - Functional
- Coincidental is lowest, functional is highest
- Scale is not linear
- Functional is considered very strong

Determining Cohesion

- Describe the purpose of a module in a sentence
- Perform the following tests

1. If the sentence has to be a compound sentence, contains more than one verbs, the module is probably performing more than one function. Probably has sequential or communicational cohesion.
2. If the sentence contains words relating to time, like "first", "next", "after", "start" etc., the module probably has sequential or temporal cohesion.

3. If the predicate of the sentence does not contain a single specific object following the verb, the module is probably logically cohesive. Eg "edit all data", while "edit source data" may have functional cohesion.

4. Words like "initialize", "clean-up" often imply temporal cohesion.

- Functionally cohesive module can always be described by a simple statement

Cohesion in OO Systems

- In OO, different types of cohesion is possible as classes are the modules
  - Method cohesion
  - Class cohesion
  - Inheritance cohesion
- Method cohesion – why diff code elts are together in a method
  - Like cohesion in functional modules; highest form is if each method implements a clearly defined function with all elts contributing to implementing this function

Cohesion in OO…

- Class cohesion – why diff attributes and methods are together in a class
  - A class should represent a single concept with all elts contributing towards it
  - Whenever multiple concepts encapsulated, cohesion is not as high
  - A symptom of multiple concepts – diff gps of methods accessing diff subsets of attributes

Cohesion in OO…

- Inheritance cohesion – focuses on why classes are together in a hierarchy
  - Two reasons for subclassing – generalization-specialization and reuse
  - Cohesion is higher if the hierarchy is for providing generalization-specialization

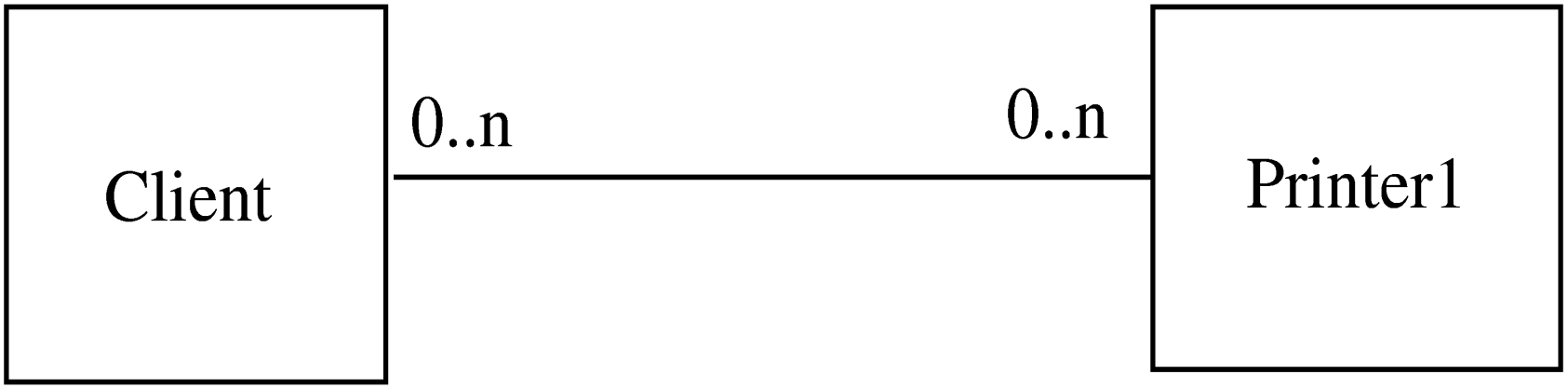
Open-closed Principle

- Besides cohesion and coupling, open closed principle also helps in achieving modularity
- Principle: A module should be open for extension but closed for modification
  - Behavior can be extended to accommodate new requirements, but existing code is not modified
  - I.e. allows addition of code, but not modification of existing code
  - Minimizes risk of having existing functionality stop working due to changes – a very important consideration while changing code
  - Good for programmers as they like writing new code

Open-closed Principle…

- In OO this principle is satisfied by using inheritance and polymorphism
- Inheritance allows creating a new class to extend behavior without changing the original class
- This can be used to support the open-closed principle
- Consider example of a client object which interacts with a printer object for printing

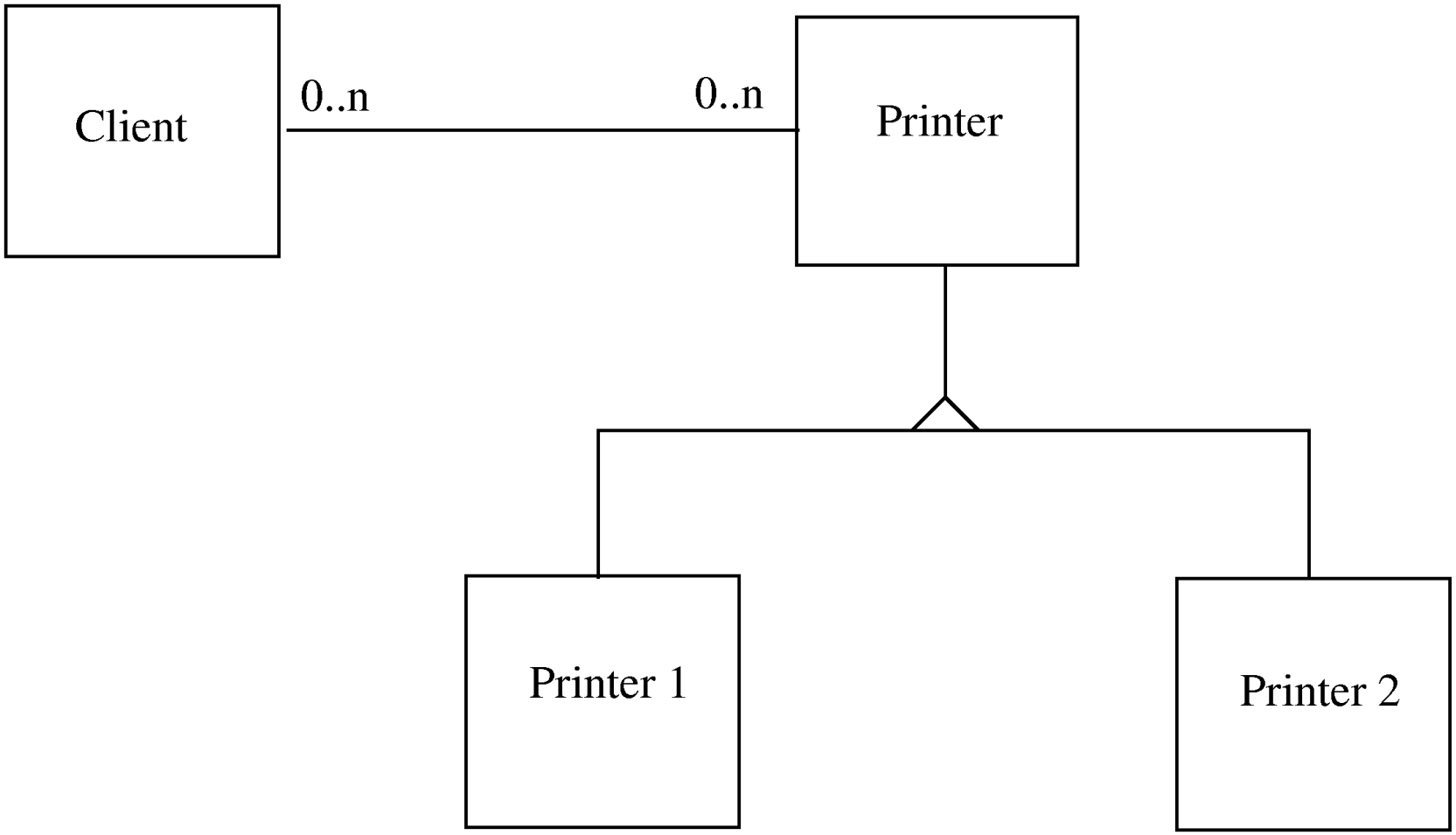
Example



Example..

- Client directly calls methods on Printer1
- If another printer is to be allowed
  - A new class Printer2 will be created
  - But the client will have to be changed if it wants to use Printer 2
- Alternative approach
  - Have Printer1 a subclass of a general Printer
  - For modification, add another subclass Printer 2
  - Client does not need to be changed

Example…



Liskov's Substitution Principle

- Principle: Program using object o1 of base class C should remain unchanged if o1 is replaced by an object of a subclass of C
- If hierarchies follow this principle, the open-closed principle gets supported

Summary

- Goal of designing is to find the best possible correct design
- Modularity is the criteria for deciding quality of the design
- Modularity enhanced by low coupling, high cohesion, and following open-closed principle