Measures of Module Strength

The relative independence of a module is considered to be the strength of a module.

- Fairly independent modules are easier to implement and easier to maintain.

- **Two qualitative measures:**
  - coupling
  - cohesion

Coupling

Coupling is a **measure of interconnection between the modules of a software product.**

- There must be at least two modules involved.
- Measures the complexity of module interfaces:
  - how many parameters are passed?
  - how many parameters are composite?
  - how many parameters are shared values?
  - ....
**Coupling**

- No direct coupling
- Data coupling
- Stamp coupling
- Control coupling
- External coupling
- Content coupling
- Hybrid coupling

**Types of Coupling**

**Data coupling**
- data passed through arguments
- generally, one argument list is used
- relatively, low-level coupling
  - example: `print (a, b)`

**Stamp coupling**
- data structures passed as arguments
- generally, passed via module interface
- coupling measure in this type is higher than in data coupling
Types of Coupling

Control coupling
- a control variable is passed to coordinate or synchronize executions between modules
- similar to data coupling but limited to one or two control variables
- relatively low-level coupling

External coupling
- relatively high-level coupling
- example: I/O connections – involve specific formats and protocols

Common coupling
- sharing data that is stored in a common place, called “global data area”
- both data and data structures may be shared
- high-level coupling because the effect of failure of one module is propagated to all the other modules that use the same data
- example: files in disks/secondary storage
Types of Coupling (cont’d)

Content coupling
- One module can access the data and control information (possibly the code) of another module.
- Practically highest level of coupling
- Example: two procedures in an assembly language program

Hybrid coupling
- One module modifies the code of another.
- This is rarely intended, but sometimes used to create viruses.

Impact of coupling on design

A good design process should aim at reducing coupling:
- Reduction of coupling ➔ reduction of dependence of one module on another ➔ increase the independence of module ➔ increase the ability to change or maintain the modules
Cohesion

Cohesion measures how a single module is related to a particular functionality in the system.
- only one module is involved
- ideally, a highly cohesive module should do only one task/activity/function
- example:
  - a sorting module that contains only one sorting function and this function sorts integers only.
  - a sorting module that contains several sorting functions that implement various sorting techniques but all sort integers only.
  - a sorting module that contains several sorting functions that implement various sorting techniques but sort integers and floats.
Types of Cohesion

Coincidental cohesion
- functions that are not at all related to each other but are placed in a single module (happen to be a coincidence)
  • example: a function that performs sorting and a printer driver, both in the same module
- functions that are somewhat related but do not have much in common also fall in this category
  • example: a function that computes an average of a sequence and a function that sorts a sequence, both being placed in the same module
- low-level cohesion

Types of Cohesion (cont’d)

Logical cohesion
- functions that are logically related to each other, all placed in the same module
  • example: a set of functions that output a given data in various formats (bar chart, graph, pie-chart, …)
- moderate level of cohesion

Temporal cohesion
- functions that are related by time, all placed in the same module
  • example: the alarm system, automatic telephone dialing unit of a security system both placed in the same module; these two must be activated at the same time
- moderate level of cohesion
Types of Cohesion (cont’d)

Procedural cohesion
- functions that are related by a sequence of procedure calls, all placed in the same module
- example: an input function receiving a data, a function that processes the data, and a function that outputs the result of that computation, all placed in the same module
- moderate level of cohesion

Types of Cohesion (cont’d)

Communication cohesion
- all functions in a module concentrate on one area of a data structure or use the same data structure
- example: a shared file and a set of functions acting on that shared file
- low to moderate level of cohesion, mostly lower
Types of Cohesion (cont’d)

Data cohesion
- functions that share data are grouped together
  - example: a pipe or a filter
  - good level of cohesion

Functional cohesion
- functions that are related as part of achieving the same functional requirement
  - high level of cohesion

Cohesion and OO

- ADT - abstract data type
  - data and its associated operations should be grouped together -- a class

- Inheritance reduces cohesion
  - Understanding a class requires knowledge of its superclasses.
Impact of cohesion on Design

A good design process should try to maximize cohesion of each module
- maximizing cohesion $\Rightarrow$ maximizing the use of the module towards particular functionality $\Rightarrow$ appropriate modularization of the design

Evaluating coupling and cohesion

- **Coupling can be evaluated using metrics tools.**
  - metrics will be discussed later in the course

- **Cohesion is generally evaluated manually by experts / software engineers.**
  - walk through the design documents and iterate the design until cohesion is improved to a satisfactory level
Three Reasons for OO

• **Encapsulation & Information Hiding**
  - data related by real-world usage
  - data easy to access inside the class, difficult to access outside the class (public & private)

• **Abstraction**
  - data and functionality grouped together
  - use descriptive method names (e.g., get & set)

• **Inheritance and Polymorphism**
  - reuse data and functions from superclasses