

La fase di progetto

- Fase in cui si decidono le modalità di passaggio da "*che cosa*" deve essere realizzato nel sistema software a "*come*" la realizzazione deve aver luogo
- La fase di progetto **prende in input** il *documento di specifica* (analisi dei requisiti) **e produce** un *documento di progetto* che guida la successiva fase di codifica
- La fase di progetto può essere suddivisa in due sotto-fasi:
 - **progetto architettonale** (o **preliminare**), in cui il sistema software complessivo viene suddiviso in più sottosistemi (*decomposizione modulare*)
 - **progetto dettagliato**, in cui ogni sottosistema identificato viene progettato in dettaglio, scegliendo algoritmi e strutture dati specifiche

Principi di progettazione

- *stepwise refinement* (usato anche in fase di *analisi* dei requisiti)
- **astrazione** (usato anche in fase di *analisi* dei requisiti)
- **decomposizione modulare**
- **modularità**
- *information hiding*
- **riusabilità**

Stepwise refinement

- Il procedere per **raffinamenti successivi (stepwise refinement)** è una strategia di progettazione top-down proposta da **Wirth** (1971) nell'ambito della programmazione strutturata
- Il raffinamento è un **processo di elaborazione** che parte dalla specifica di una funzione (o di dati) in cui non si descrive il funzionamento interno della funzione o la struttura interna dei dati
- Il raffinamento elabora la specifica **aggiungendo ad ogni passo un livello di dettaglio maggiore**

Stepwise refinement (2)

- L'importanza del raffinamento deriva dalla **legge di Miller (1956)**: "*at any one time a human being can concentrate on at most 7 ± 2 chunks (quanta of information)*"
- Il *raffinamento* è un concetto **complementare** al concetto di *astrazione*

Astrazione

- L'astrazione consiste nel concentrarsi sugli **aspetti essenziali** di una entità e nell'**ignorare i dettagli secondari**
- Il concetto di livello di astrazione è stato introdotto da **Dijkstra** (1968) nell'ambito dei sistemi operativi, al fine di descriverne l'architettura a strati
- Nell'ambito del processo software, **ogni passo rappresenta un raffinamento del livello di astrazione della soluzione**. Ciò significa concentrarsi su cosa è e cosa fa una entità del sistema software prima di decidere come debba essere realizzata

Astrazione (2)

- I principali tipi di astrazione sono:
 - **astrazione procedurale** (es. funzioni C)
 - **astrazione dei dati** (es. *data encapsulation*)
- Con il termine ***data encapsulation*** ci si riferisce ad una struttura dati insieme alle azioni eseguite su di essa (es. *stack*, struttura dati con le azioni che realizzano il meccanismo LIFO)
- L'uso di *data encapsulation* in fase di progetto permette di ottenere vantaggi sia in fase di codifica che in fase di manutenzione

Tipi di dati astratti

- Un tipo di dato astratto (*abstract data type*) identifica un tipo di dato insieme alle azioni eseguite sulle istanze del tipo di dato
- Un tipo di dato astratto combina astrazione procedurale e astrazione dei dati
- L'uso dei tipi di dati astratti permette di migliorare la qualità del software in relazione agli attributi di *riusabilità* e *manutenibilità*
- Una *classe C++* è un esempio di tipo di dato astratto che inoltre supporta il meccanismo di ereditarietà

Esempio di *abstract* *data* *type* (classe C++)

```
class JobQueue
{
    // attributes
public:
    int queueLength;                                // length of job queue
    int queue[25];                                   // queue can contain up to 25 jobs

    // methods
public:
    void initializeJobQueue ()
    /*
     * empty job queue has length 0
     */
    {
        queueLength = 0;
    }

    void addJobToQueue (int jobNumber)
    /*
     * add job to end of job queue
     */
    {
        queue[queueLength] = jobNumber;
        queueLength = queueLength + 1;
    }

    void removeJobFromQueue (int& jobNumber)
    /*
     * set jobNumber equal to the number of the job stored at the head of the queue,
     * remove the job at the head of the job queue, and move up the remaining jobs
     */
    {
        jobNumber = queue[0];
        queueLength = queueLength - 1;
        for (int k = 0; k < queueLength; k++)
            queue[k] = queue[k + 1];
    }
} // class JobQueue
```

Modularità (1)

- Prodotti software fatti di un unico, monolitico, blocco di codice sono difficili a:
 - manutenere
 - correggere
 - capire
 - riusare
- La soluzione consiste nel suddividere il prodotto software in segmenti più piccoli detti **moduli**

Modularità (2)

- Definizione (IEEE Std 610.12 - Standard Glossary of Software Engineering Technology):
the extent to which software is composed of discrete components such that a change to one component has minimal impact on the other components

Decomposizione modulare

- Un **modulo** è un elemento software che:
 - contiene istruzioni, logica di elaborazione e strutture dati
 - può essere compilato separatamente e memorizzato all'interno di una libreria software
 - può essere incluso in un programma
 - può essere usato invocando segmenti di modulo identificati da un nome e da una lista di parametri
 - può usare altri moduli
- La suddivisione in moduli di un sistema software (*decomposizione modulare*) produce come risultato l'identificazione di una **architettura dei moduli (structure chart)**

Decomposizione modulare (2)

- L'architettura dei moduli di un sistema software descrive la struttura dei moduli, il modo in cui tali moduli **interagiscono** e la **struttura dei dati manipolati**
- La decomposizione modulare si basa sul principio del "*divide et impera*"
- Detti **p1** e **p2** due *problem*, **C** la *complessità* ed **E** l'*effort* si ha che:

$$C(p_1) > C(p_2) \Rightarrow E(p_1) > E(p_2)$$

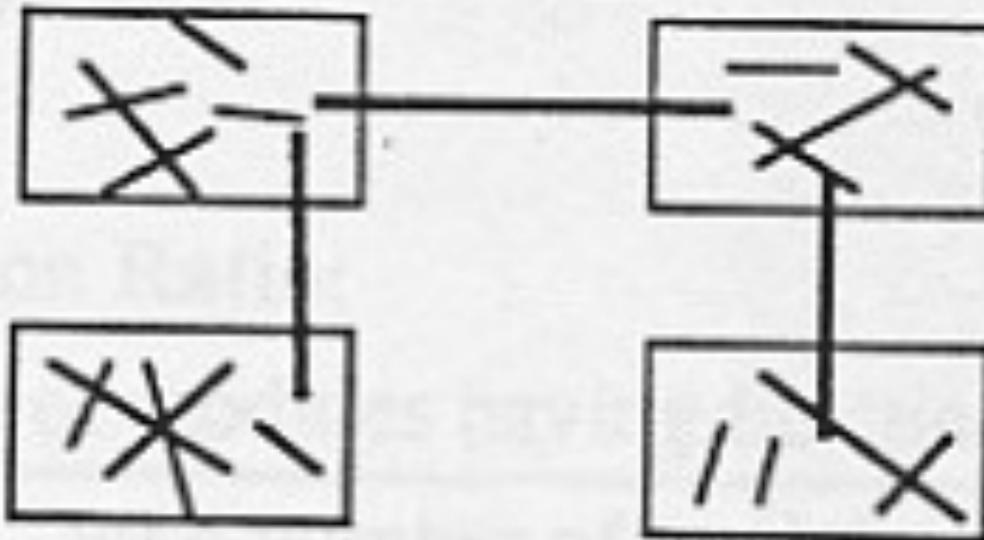
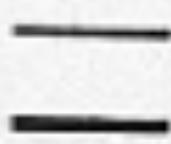
$$C(p_1+p_2) > C(p_1) + C(p_2)$$
$$\Downarrow$$

$$E(p_1+p_2) > E(p_1) + E(p_2)$$

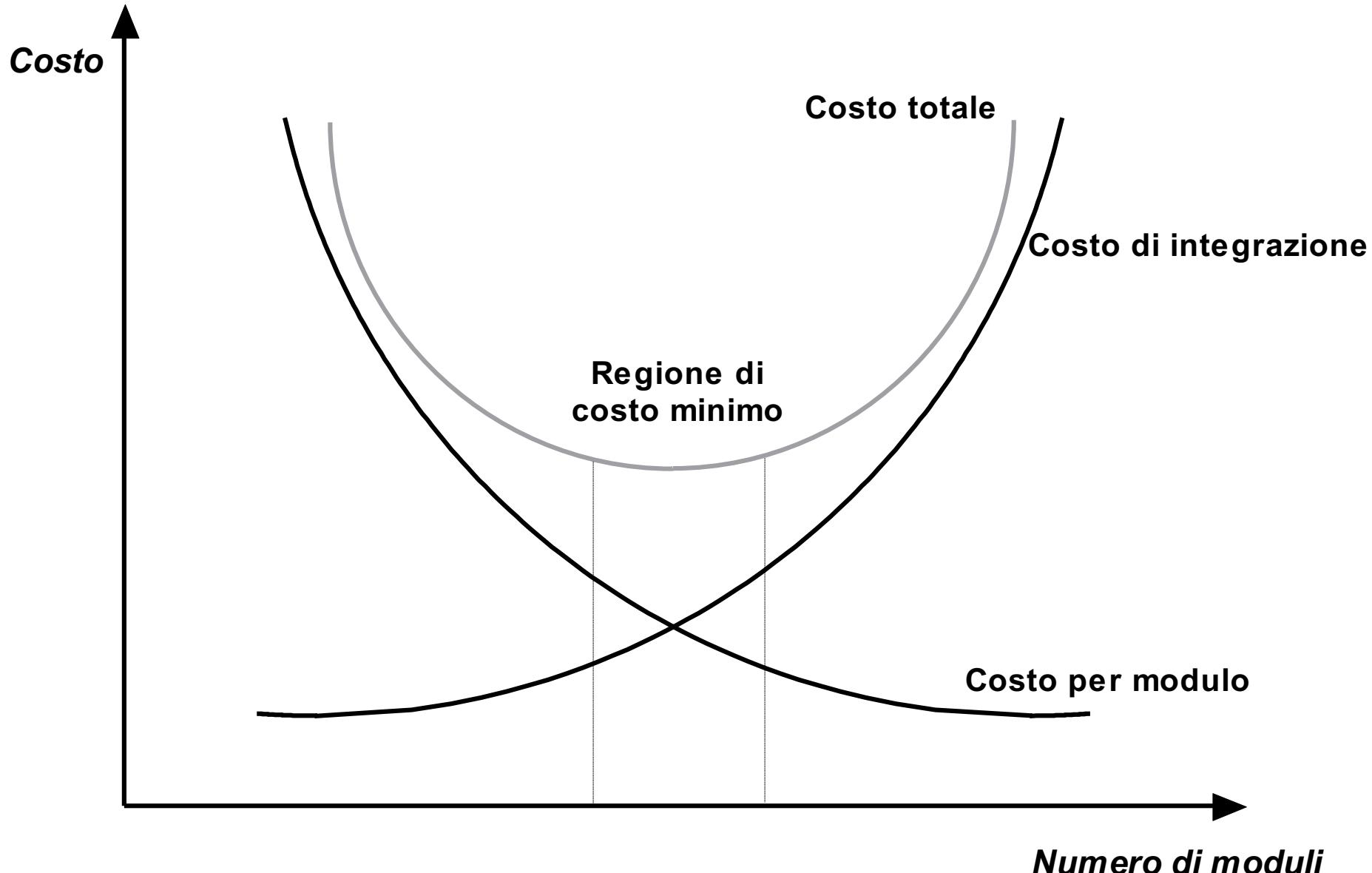
Decomposizione modulare (3)

- Un buona divisione di un prodotto software in moduli è quella che permette di ottenere:
 - *Massima coesione (**cohesion**) interna ai moduli*
 - *Minimo grado di accoppiamento (**coupling**) tra i moduli*
 - Infatti, massima coesione e minimo coupling permettono di incrementare:
 - Comprensibilità
 - Manutenibilità
 - Estensibilità
 - Riusabilità
- del prodotto software

Cohesion/Coupling rispetto a Modularità



Modularità e costo del software



Coesione

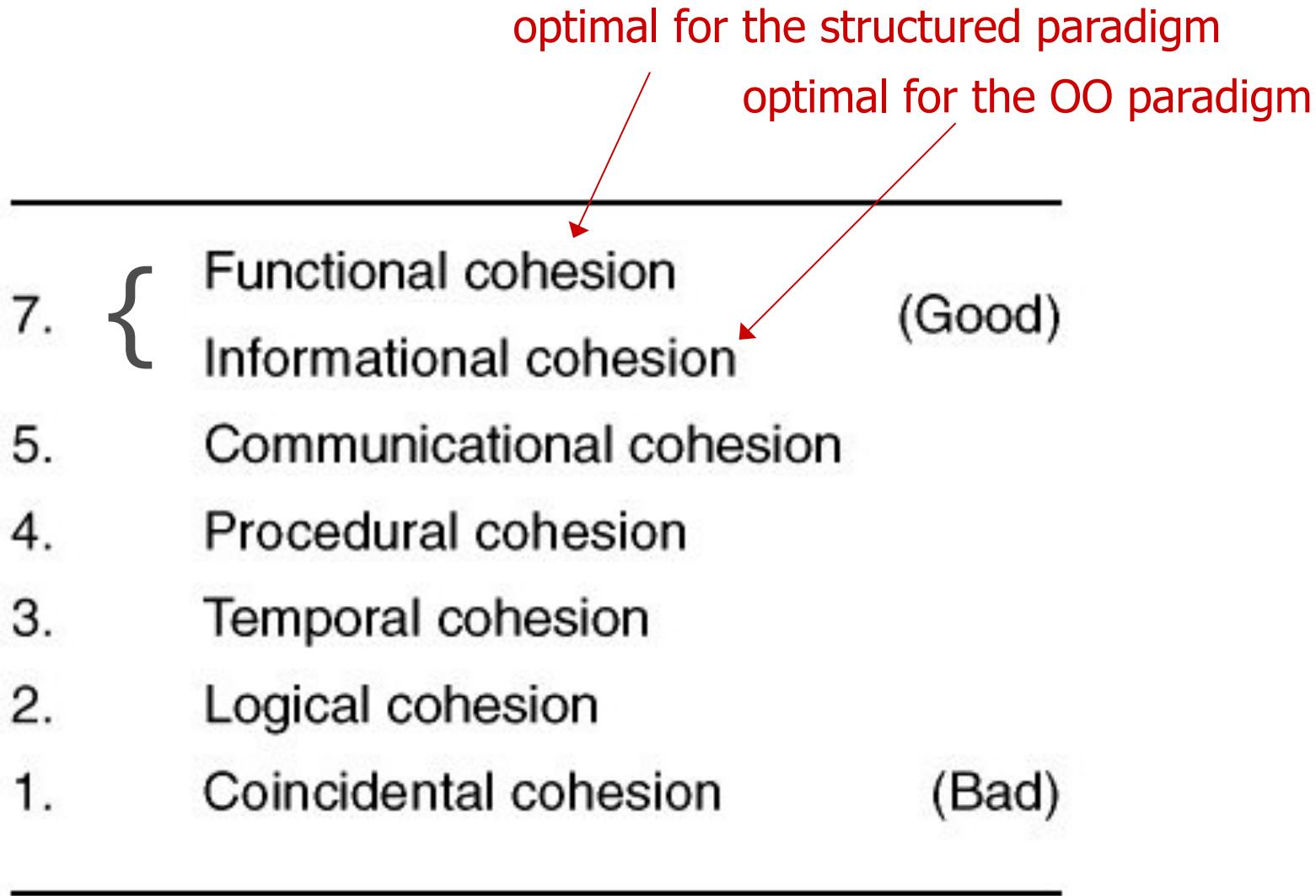
- Per eseguire una *funzione* sono necessarie varie *azioni*.
- Le azioni possono essere concentrate in un singolo modulo oppure sparse tra tanti.
- Coesione di un modulo = Misura in cui il modulo espleta *internamente* tutte le azioni necessarie a espletare una data funzione (cioè senza interagire con le azioni interne ad altri moduli).
- Coesione misura dunque il grado di *interazione interna* al modulo tra le azioni di una funzione.

Livelli di Coesione

(1 è il peggiore, 7 il migliore)

1. **Coincidental** (nessuna relazione tra gli elementi del modulo).
2. **Logical** (elementi correlati, di cui uno viene selezionato dal modulo chiamante)
3. **Temporal** (relazione di ordine temporale tra gli elementi).
4. **Procedural** (elementi correlati in base ad una sequenza predefinita di passi da eseguire).
5. **Communicational** (elementi correlati in base ad una sequenza predefinita di passi che vengono eseguiti sulla stessa struttura dati).
6. **Informational** (ogni elemento ha una porzione di codice indipendente e un proprio punto di ingresso ed uscita; tutti gli elementi agiscono sulla stessa struttura dati).
7. **Functional** (tutti gli elementi sono correlati dal fatto di svolgere una singola funzione)

Livelli di Coesione (2)



Coincidental Cohesion: example

- Module functions:
 - print next line
 - invert characters of the second string parameter
 - add 7 to the fifth parameter
 - perform *int-double* conversion to the fourth parameter

Logical Cohesion: example

module performing all input and output		
1.	Code for all input and output	
2.	Code for input only	
3.	Code for output only	
4.	Code for disk and tape I/O	
5.	Code for disk I/O	
6.	Code for tape I/O	
7.	Code for disk input	
8.	Code for disk output	
9.	Code for tape input	
10.	Code for tape output	
:	:	:
37.	Code for keyboard input	

Temporal Cohesion: example

- Module functions:
 - Open *old_master_file*
 - Open *new_master_file*
 - Open *transaction_file*
 - Open *print_file*
 - Initialize *sales_region_table*
 - Read first *transaction_file* records
 - Read first *old_master_file* record

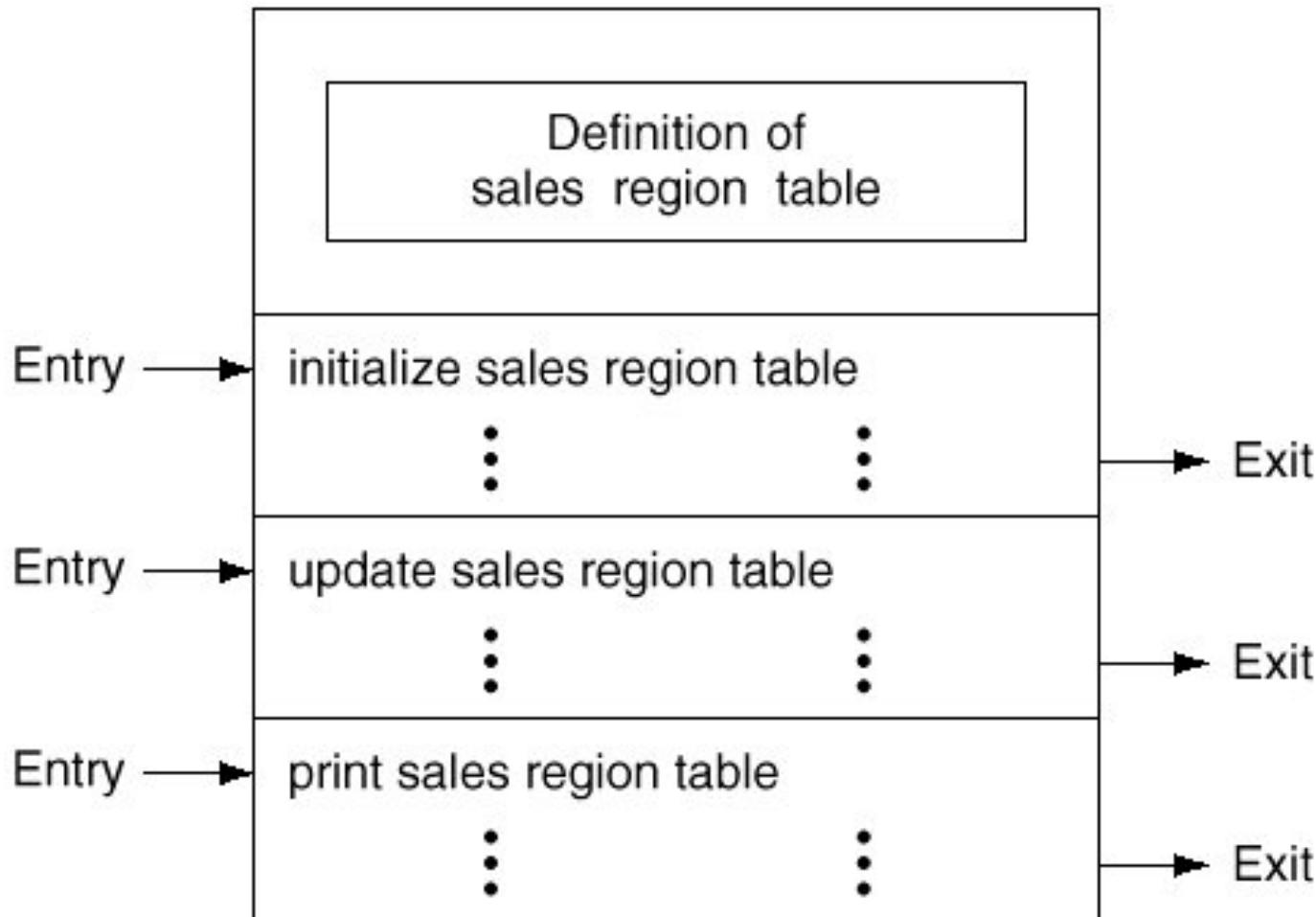
Procedural Cohesion: example

- Module functions:
 - Read *part_number* from database
 - Use *part_number* to update
repair_record on *maintenance_file*

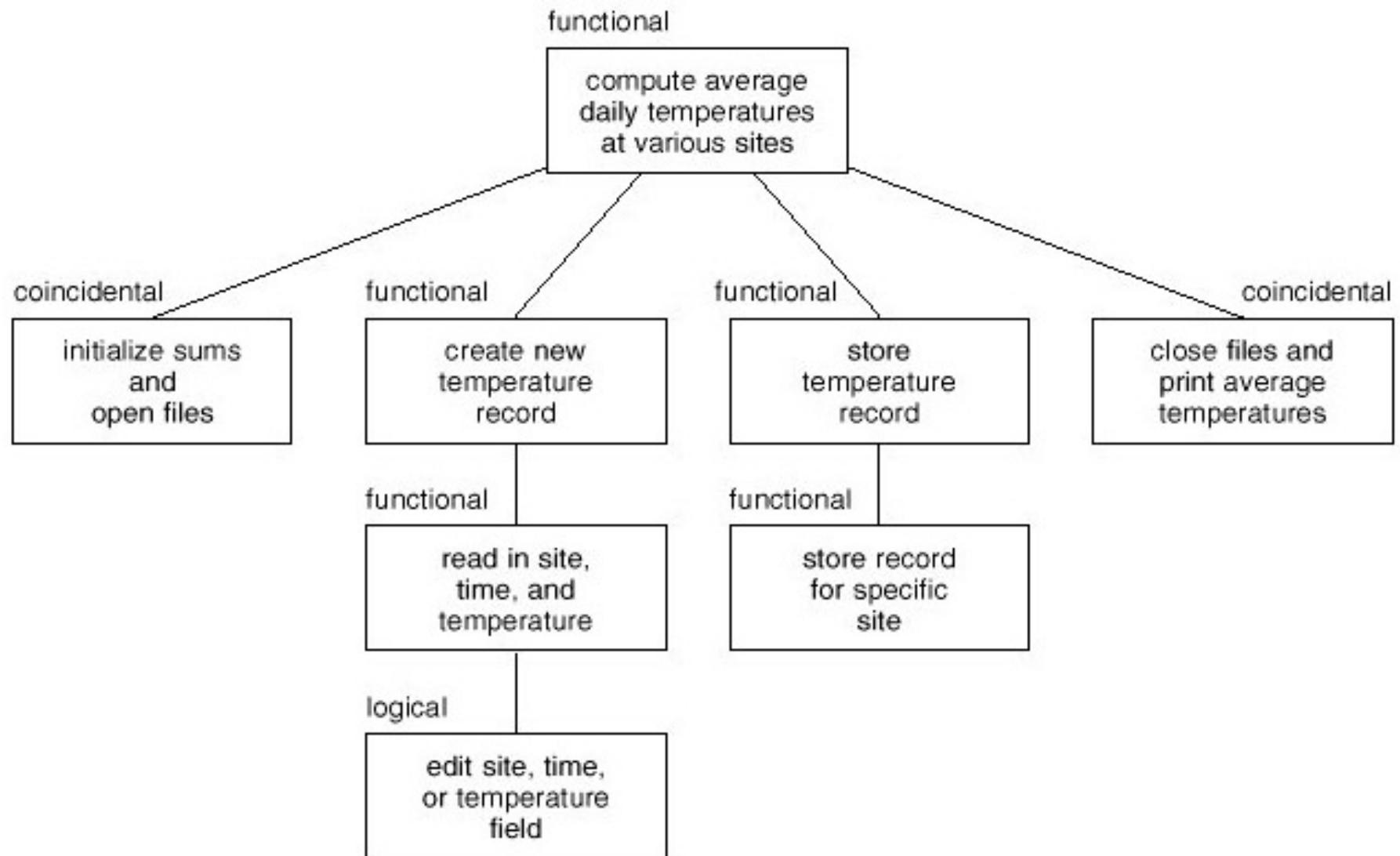
Communicational Cohesion: example

- Ex. 1: module functions
 - Update *record_a* in database
 - Write *record_a* to the *trajectory_file*
- Ex. 2: module functions
 - Calculate *new_trajectory*
 - Send *new_trajectory* to the printer

Informational Cohesion: example



Example Structure Chart & Modules Cohesion



Coupling

- Misura il grado di *accoppiamento* tra moduli
- Livelli di *coupling* (1 è il peggiore, 5 il migliore):
 1. **Content** (un modulo fa diretto riferimento al contenuto di un altro modulo).
 2. **Common** (due moduli che accedono alla stessa struttura dati)
 3. **Control** (un modulo controlla esplicitamente l'esecuzione di un altro modulo).
 4. **Stamp** (due moduli che si passano come argomento una struttura dati, della quale si usano solo alcuni elementi).
 5. **Data** (due moduli che si passano argomenti omogenei, ovvero argomenti semplici o strutture dati delle quali si usano tutti gli elementi).

Factors affecting Coupling

Strength of coupling depends on:

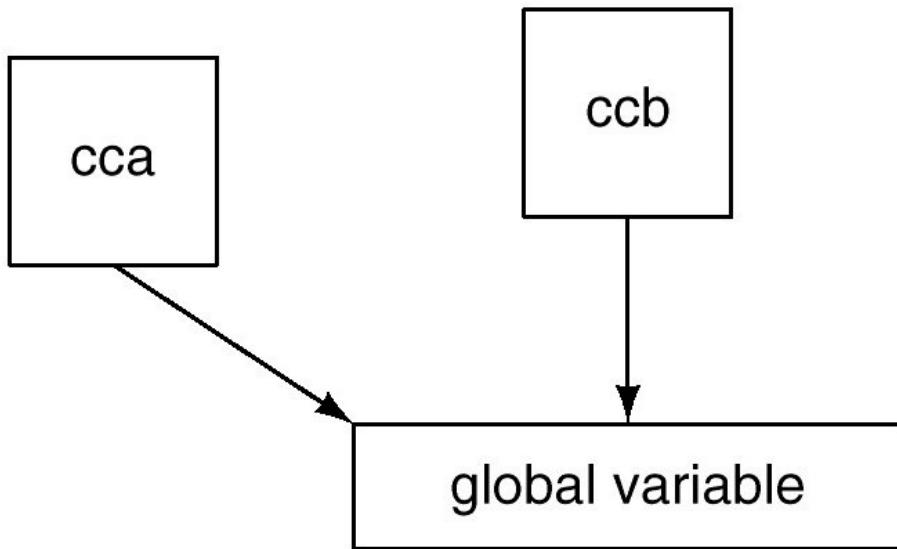
- the number of references of one module by another
- the amount of data passed/shared between modules
- the complexity of the interface between modules
- the amount of control exercised by one module over another

Content Coupling: example

$p \rightarrow q$

- Ex. 1: module p modifies a statement of module q
- Ex. 2: p refers to local data of module q in terms of some numerical displacement within q
- Ex. 3: p branches to a local label of q

Common Coupling: example



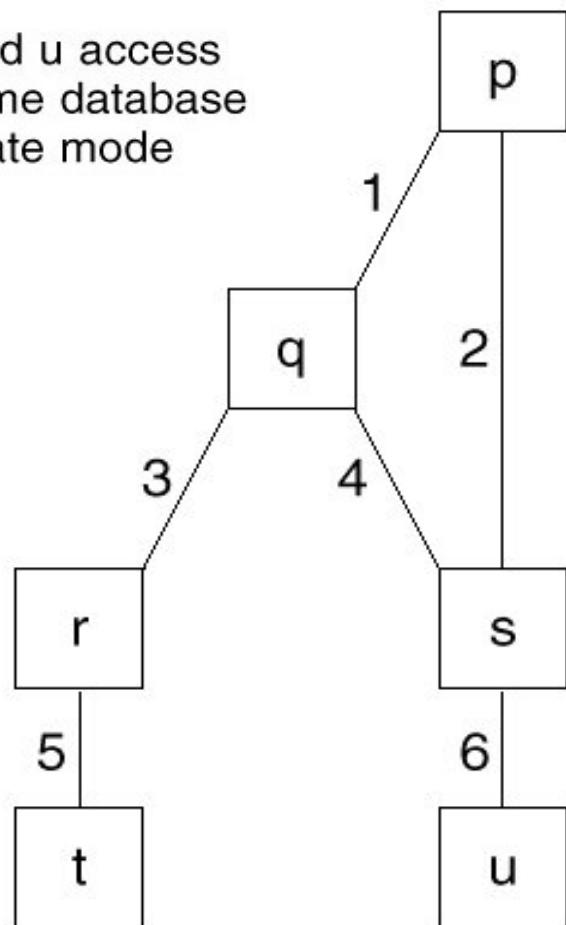
```
while (global variable == 0)
{
    if (argument xyz > 25)
        module 3 ();
    else
        module 4 ();
}
```

Control Coupling: example

- Module p calls module q
 - and asks q to perform an action,
 - q passes back a flag (e.g. “task not completed”)
 - and also asks p to perform an action (e.g. “print an error message”)

Example Structure Chart & Modules Coupling

p, t, and u access
the same database
in update mode



parameters

number	In	Out
1	aircraft type	status flag
2	—	list of aircraft parts
3	function code	—
4	—	list of aircraft parts
5	part number	part manufacturer
6	part number	part name

coupling

	q	r	s	t	u
p	Data	—	Data or Stamp	Common	Common
q		Control	Data or Stamp	—	—
r			—	Data	—
s					Data

Information hiding

- I concetti di astrazione procedurale e astrazione dei dati sono derivati da un concetto più generale detto *information hiding*, introdotto da **Parnas** (1971)
- La tecnica di *information hiding* consiste nel definire e progettare i moduli in modo che i dettagli implementativi (procedura e dati) non siano accessibili ad altri moduli che non abbiano necessità di conoscere tali dettagli
- I vantaggi della tecnica di *information hiding* si riscontrano quando è necessario apportare modifiche (fasi di *testing* e *manutenzione*)

Esempio di *information hiding*

Esempio di tipo
di dato astratto
(classe C++)
realizzato con
*information
hiding*

```
class JobQueue
{
    // attributes
private:
    int    queueLength;      // length of job queue
    int    queue[25];        // queue can contain up to 25 jobs

    // methods
public:
    void initializeJobQueue()
    {
        ....
    }

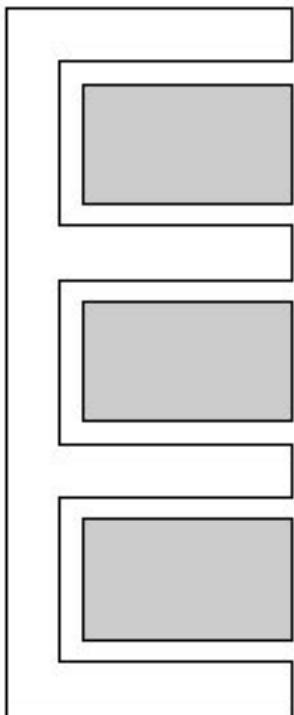
    void addJobToQueue (int jobNumber)
    {
        ....
    }

    void removeJobFromQueue (int& jobNumber)
    {
        ....
    }
}
```

Riusabilità

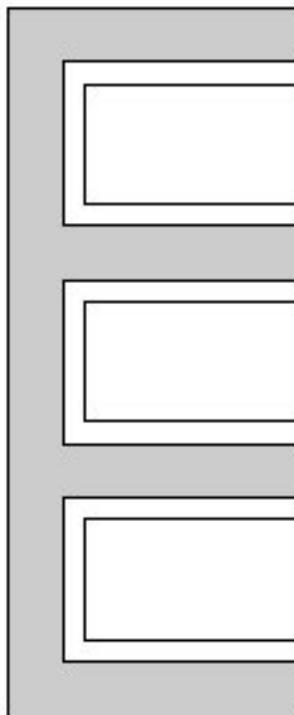
- La **riusabilità** fa riferimento all'utilizzo di componenti sviluppati per un prodotto all'interno di un prodotto differente
- Per **componente riusabile** si intende non solo un modulo o un frammento di codice, ma anche progetti, parti di documenti, insiemi di test data o stime di costi e durata
- Vantaggi:
 - netta diminuzione di costi e tempi di produzione del software
 - incremento dell'affidabilità dovuto all'uso di componenti già convalidati
- La riusabilità nella fase di progetto si applica a:
 - (a) *moduli software*
 - (b) *application framework*, che incorpora la logica di controllo di un progetto
 - (c) *design pattern*, che identifica una soluzione di progetto ricorrente in applicazioni dello stesso tipo
 - (d) *architetture software* comprendenti (a), (b) e (c)

Riusabilità (2)



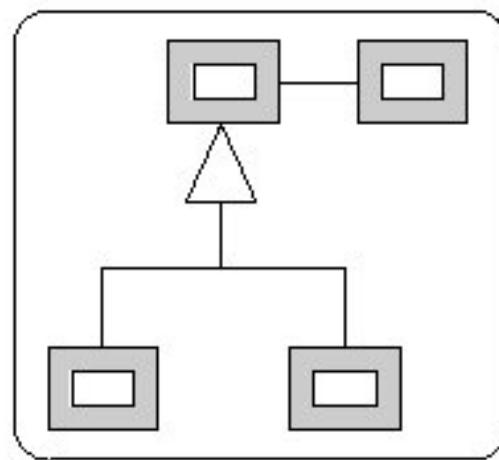
(a)

*moduli
software*



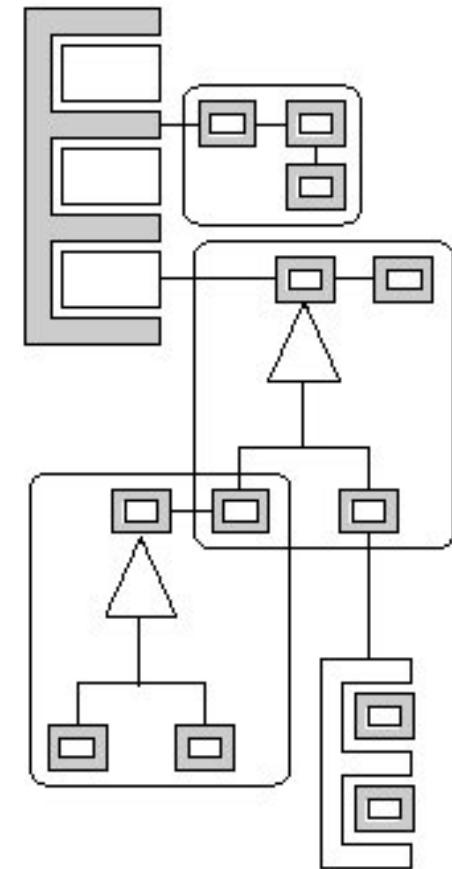
(b)

*application
framework*



(c)

*design
pattern*



(d)

*architettura
software*