Progettazione

Alberto Gianoli

Univ. Ferrara - Corso di Ingegneria del Software

Dove?

- Pressman, cap. 11,13
-

Cos'è

- La progettazione è il processo che porta alla definizione ingegneristica di ciò che deve essere realizzato
- Si compone di due fasi:
  - **diversificazione**: il progettista acquisisce il materiale grezzo del progetto (componenti, possibilità di realizzazione, conoscenze) per individuare le possibilità realizzative
  - **convergenza**: il progettista sceglie e combina gli elementi disponibili per arrivare ad un prodotto finale

I 3 requisiti progettuali

- Il progetto deve soddisfare tutti i requisiti espliciti contenuti nel modello concettuale e tutti i requisiti impliciti voluti dal cliente
- Il progetto deve essere una guida leggibile e comprensibile per chi si occuperà delle fasi di codifica, collaudo e manutenzione
- Il progetto deve dare un quadro completo e coerente del software, considerando i domini dei dati, funzionale e comportamentale dal punto di vista dell'implementazione

Indicazioni generali

- L'architettura di progetto deve
  - essere creata con modelli di progettazione riconoscibili
  - essere costituita da componenti ben progettate
  - poter essere implementata in modo evolutivo
- Il progetto deve essere modulare e contenere una rappresentazione distinta di dati, architettura, interfacce e componenti
  - le strutture dei dati devono essere tratte da modelli di dati riconoscibili e devono essere appropriate per i dati da implementare
  - le componenti devono avere caratteristiche funzionali indipendenti
  - le interfacce devono tendere a ridurre la complessità delle comunicazioni tra moduli e verso l'esterno
- Il metodo di progetto deve essere ripetibile e pilotato dai requisiti

Tools per la progettazione

- I tools usati di solito sono catalogabili come
  - meccanismi per tradurre il modello concettuale in progetto
  - notazioni per rappresentare i componenti funzionali e le loro interfacce
  - regole euristiche per il raffinamento e la suddivisione dei moduli
  - metodi per la valutazione della qualità

Regole empiriche

- Non procedete col paraocchi
  - siate aperti all'utilizzo di soluzioni alternative
- Il progetto deve sempre essere riconducibile al modello concettuale
  - poiché un singolo elemento del progetto è relativo a più requisiti, è necessario poter risalire al modo in cui i requisiti sono soddisfatti nel progetto
- Non re-inventate la ruota
  - dove è possibile, riutilizzate schemi o strutture già sviluppati in altri progetti
- Il progetto finale deve apparire uniforme ed integrato
  - se si lavora in team definire da subito regole di formato e di stile
- Il progetto deve poter accogliere modifiche

Regole empiriche

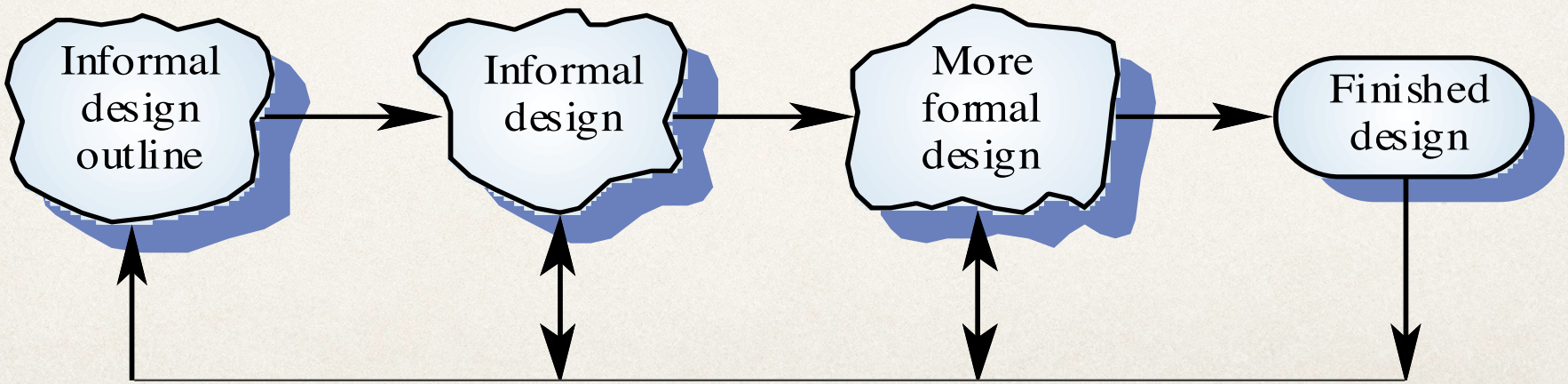
- Il software deve reagire in maniera controllata alle situazioni di errore
  - se è ben fatto, dovrebbe poter reagire a condizioni non standard e se necessario arrestarsi in maniera regolata
- Progetto ≠ stesura del codice
  - il livello di astrazione è più alto nel progetto
- La qualità del progetto e come mantenerla vanno decisi all'inizio dello sviluppo
  - sia per la qualità esterna (direttamente percepibile dall'utente) sia per quella interna (percepibile da sviluppatori e manutentori)
- Al termine del progetto va sempre prevista una revisione formale che lo riesamini

Fasi della progettazione

- Comprensione del problema
  - guardare al problema da angolature differenti
- Identificare una o più soluzioni
  - valutare le soluzioni possibili e scegliere la più appropriata rispetto all'esperienza del progettista e alle risorse disponibili
- Descrivere astrazioni delle soluzioni
  - usare notazioni grafiche, formali o altro per descrivere le componenti del progetto
- Ripetere lo step per ogni astrazione identificata, finché la progettazione non è espressa in termini primitivi

Astrazione

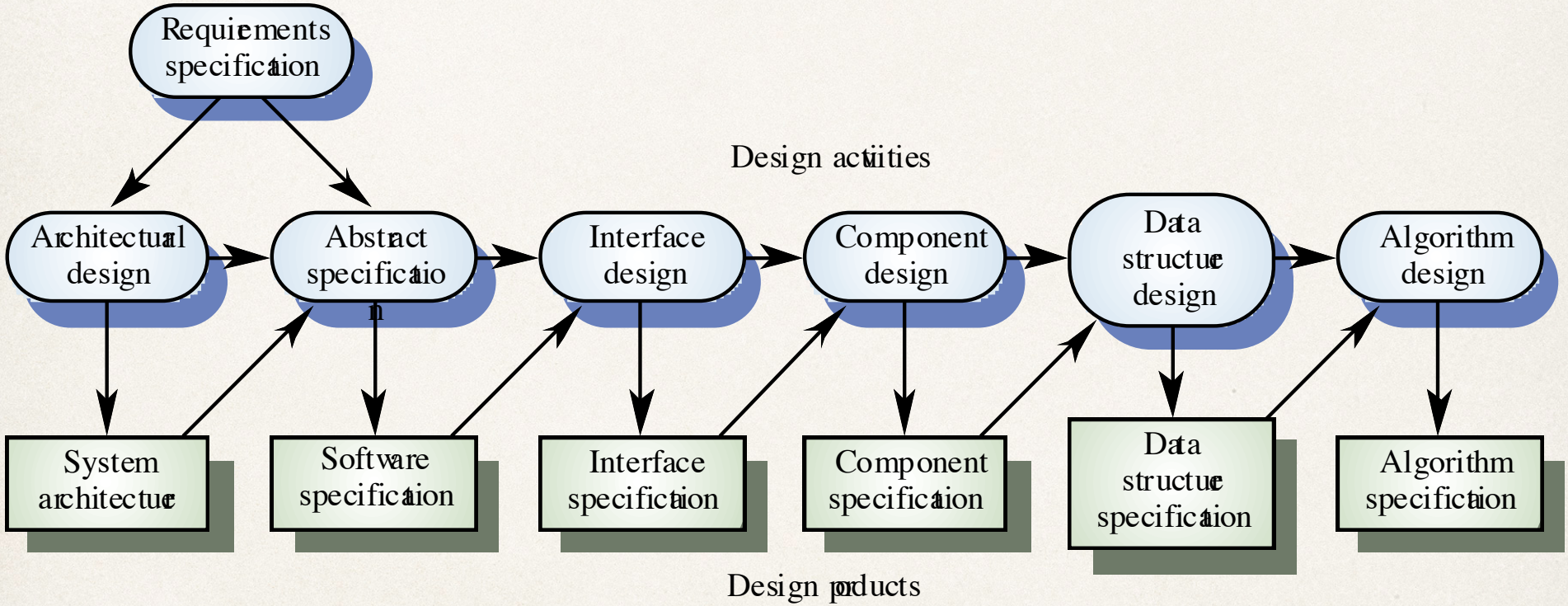
- L'astrazione è l'atto di dare una descrizione del sistema ad un certo livello, trascurando i dettagli inerenti i livelli sottostanti
  - a livelli di astrazione elevati si utilizza un linguaggio vicino al contesto del problema che il sistema dovrà risolvere (p.e. la specifica)
  - a livelli più bassi di astrazione il linguaggio si formalizza sempre di più fino ad arrivare, al livello più basso, al codice sorgente



Raffinamento

- Per raffinare utilizziamo tecniche di scomposizione per passare da astrazioni funzionali ad alto livello alle linee di codice
- Raffinamento ed astrazione possono essere considerate attività complementari
  - mediante l'astrazione il progettista specifica procedure e dati eliminando i dettagli di basso livello
  - mediante raffinamento i dettagli emergono via via

Fasi della progettazione



Fasi del design

- Architectural Design
  - identificare e documentare i sottosistemi e le loro relazioni
- Abstract Specification
  - specifichiamo i servizi forniti da ciascun sottosistema e i vincoli a cui deve sottostare
- Interface Design
  - descriviamo l'interfaccia dei sottosistemi verso altri sottosistemi; la specifica delle interfacce deve essere non ambigua: deve consentire la definizione dei sottosistemi ignorando come sono fatti all'interno

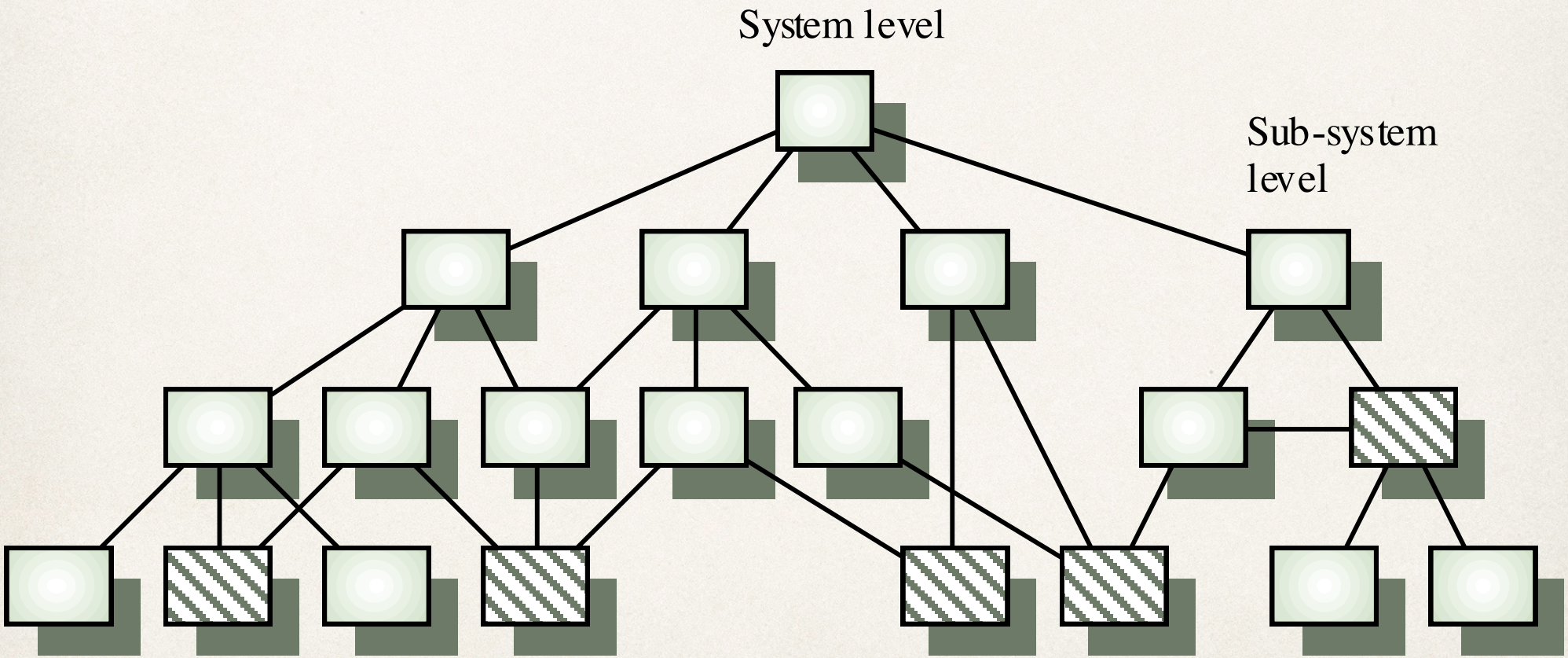
Fasi del design

- Component Design
  - allocare i servizi ai diversi componenti e definire le interfacce di questi componenti
- Data Structure Design
  - definire come sono fatte le strutture dati
- Algorithm Design
  - specificare gli algoritmi utilizzati

Progettazione top-down

- Il problema viene partizionato ricorsivamente in sottosistemi fino a che si identificano dei sottoproblemi trattabili
- In teoria, si inizia con il componente radice della gerarchia e si procede verso il basso livello dopo livello
- In pratica per sistemi di grosse dimensioni la progettazione non è mai completamente top-down: alcuni rami vengono sviluppati prima, e i progettisti riutilizzano l'esperienza e le componenti

Schema top-down



Strategie di decomposizione

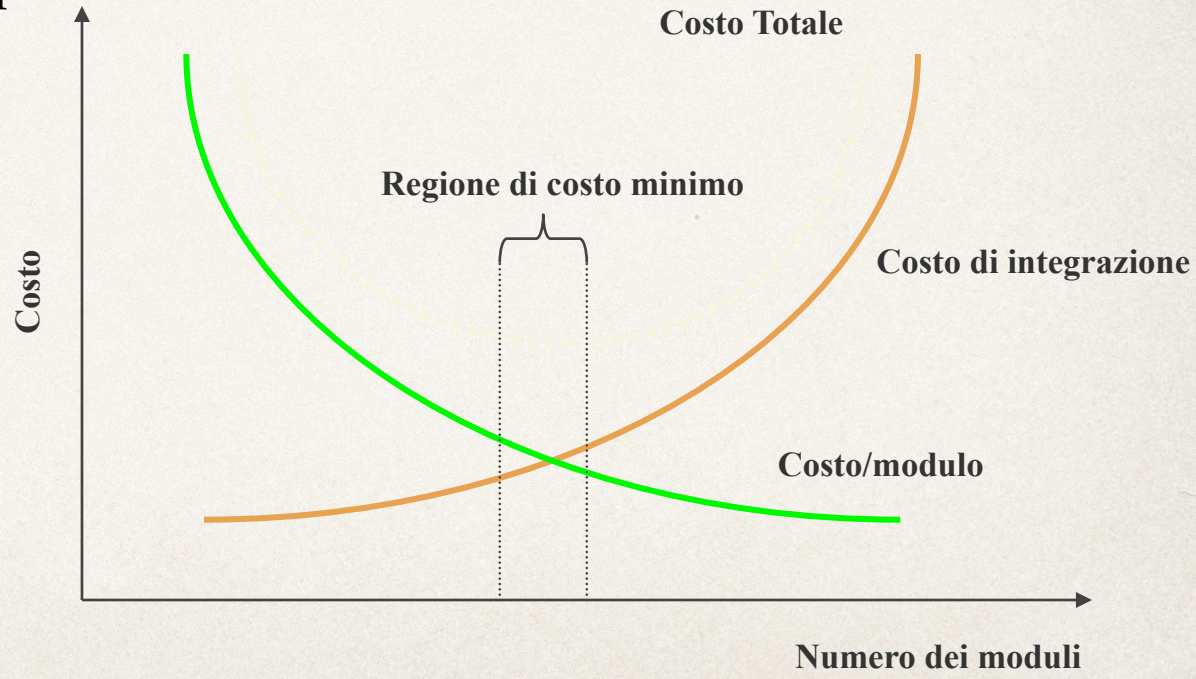
- Top-down
  - decomposizione del problema
- Bottom-up
  - composizione di soluzioni
- Sandwich
  - soluzione naturale

Modularità e integrazione

- Un sistema composto da un unico blocco monolitico di software è sempre difficile da comprendere, implementare e mantenere
- L'unico modo per permettere di gestire intellettualmente il programma è suddividerlo in moduli con funzionalità definite e limitate e con interfacce ben definite
- Warning: una eccessiva modularità richiede sforzi per l'integrazione

Costi di sviluppo e integrazione

- C(p)=complessità percepita per la soluzione del problema p
- E(p)= impegno impiegato nella risoluzione del problema p
- empiricamente si ha che:
  - $C(p_1) > C(p_2)$ implica $E(p_1) > E(p_2)$
  - $C(p_1+p_2) > C(p_1) + C(p_2)$
  - $E(p_1+p_2) > E(p_1) + E(p_2)$



Come fare in pratica?

- Come definire un modulo opportuno di date dimensioni?
  - cioè come individuare la dimensione ottimale dei moduli che minimizza la somma dei costi di sviluppo e integrazione?
- La risposta la troviamo nei metodi con cui si sviluppa un sistema basato su moduli
- Meyer (1988) ha definito cinque criteri per valutare un metodo di progettazione software in base alla loro capacità di produrre efficientemente dei sistemi modulari

I criteri di Meyer

- Scomponibilità
  - un metodo che permette la scomposizione del problema in sottoproblemi riduce la complessità
- Componibilità
  - un metodo che permette l'assemblamento di componenti preesistenti migliora la produttività
- Comprensibilità
  - un modulo le cui interfacce con altri moduli siano minimi è di più facile costruzione e modificabilità
- Continuità
  - modifiche ai requisiti di sistema che comportano solo modifiche a singoli moduli sono di facile controllo
- Protezione
  - se effetti anomali in un modulo non si propagano la mantenibilità migliora

L'architettura del software

- Intendiamo la "struttura complessiva del software e il modo in cui tale struttura sorregge l'integrità concettuale di un sistema"
  - cioè descrive la struttura gerarchica dei moduli di un programma, come interagiscono e la struttura dei dati che manipolano
- Uno degli obiettivi della progettazione è la definizione di una architettura appropriata per il sistema che si sviluppa

L'architettura del software

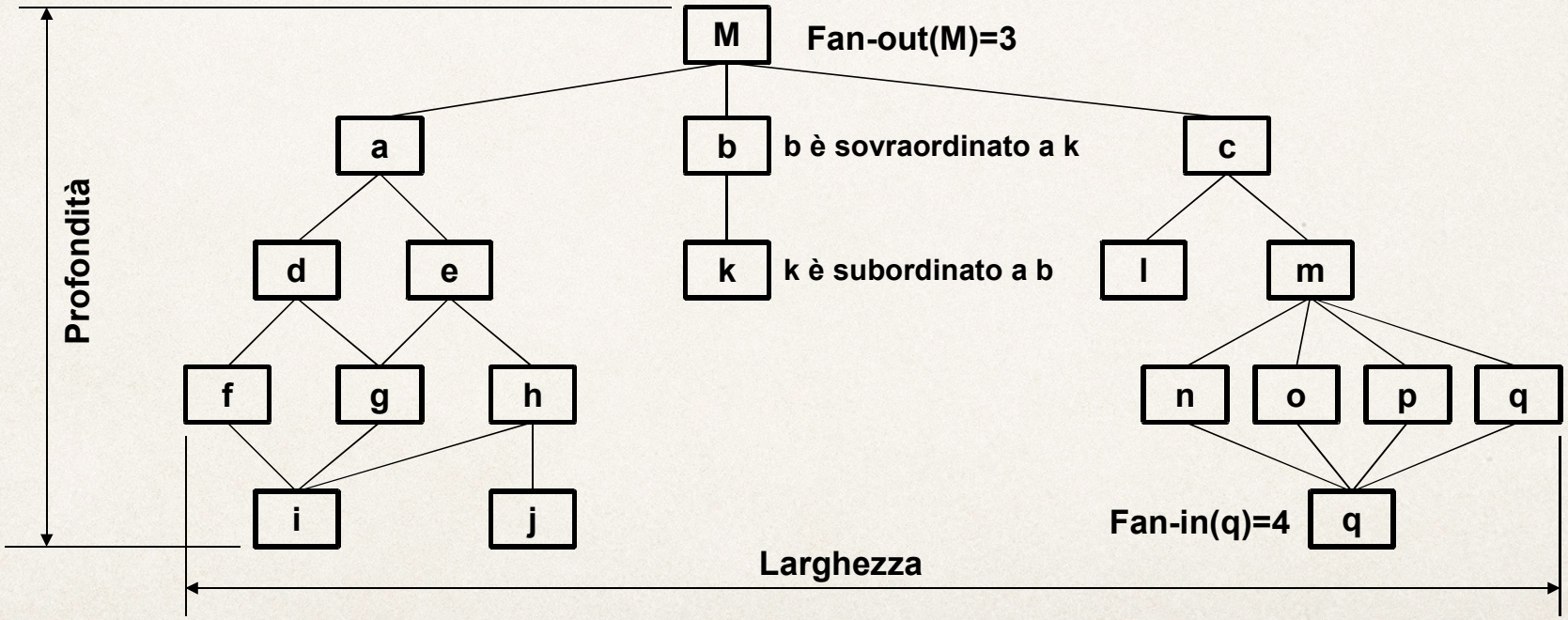
- *Proprietà strutturali*
  - l'architettura deve descrivere le componenti del sistema e come sono assemblate (interazioni tra componenti)
- *Proprietà extrafunzionali*
  - l'architettura deve esplicitare in che modo vengono soddisfatti i requisiti di prestazioni, affidabilità, sicurezza, modificabilità, ...
- *Affinità*
  - il progetto dell'architettura deve permettere di usare strutture e schemi simili per progetti simili

L'architettura del software

- Specificate le proprietà precedenti, l'architettura può essere presentata usando uno o più modelli
  - Modelli strutturali: mostrano l'architettura come una collezione organizzata di componenti
  - Modelli schematici: usati per individuare schemi progettuali ricorrenti in applicazioni dello stesso tipo
  - Modelli dinamici: mostrano gli aspetti comportamentali dell'architettura, indicano in che modo il sistema muta a seguito di eventi esterni
  - Modelli di processo: descrivono il processo aziendale o tecnico che il sistema deve supportare
  - Modelli funzionali: descrivono la gerarchia funzionale di un sistema (quali funzioni usano quali altre)

La gerarchia di controllo

- Descrive l'organizzazione gerarchica dei moduli di un programma
  - Esempio classico: la struttura ad albero *call and return*

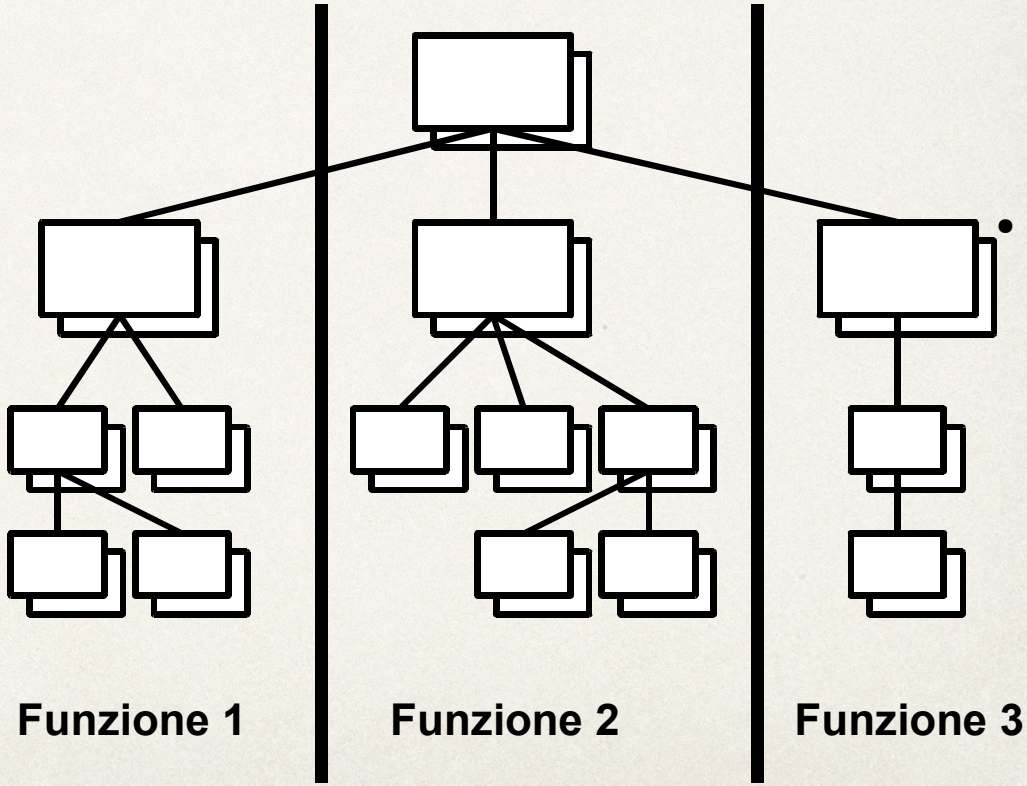


Ripartizione strutturale

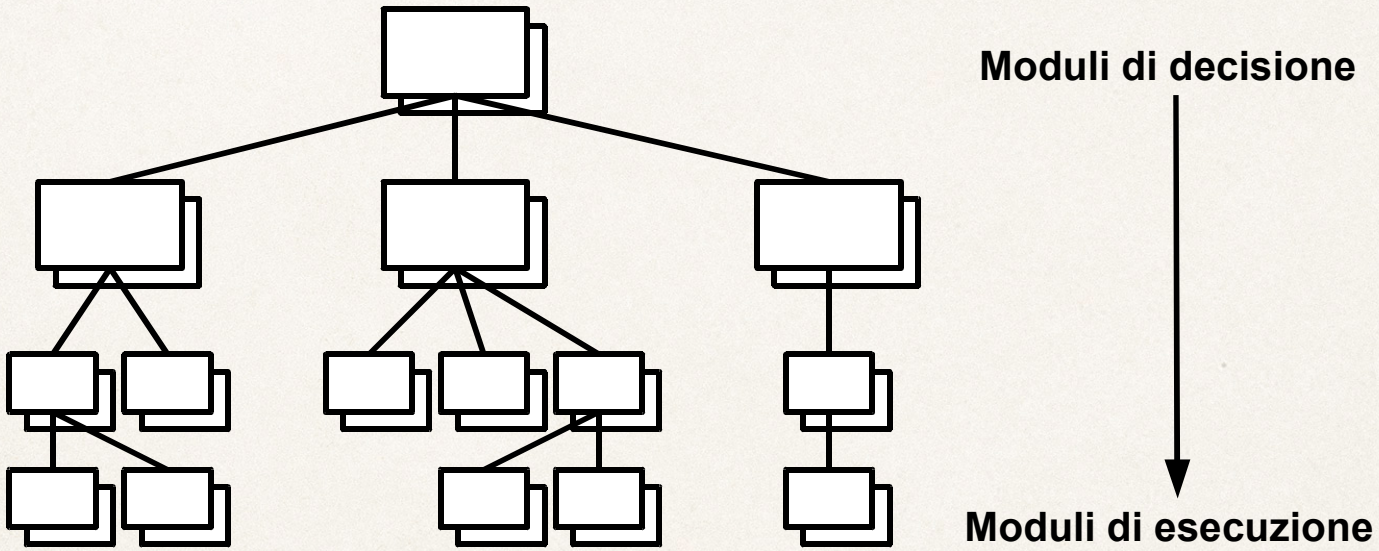
- Dato un sistema a organizzazione gerarchica (struttura ad albero), la struttura del programma può essere partizionata in senso orizzontale o verticale

Ripartizione orizzontale

- La versione più semplice definisce tre partizioni: input, trasformazioni, output
- Vantaggi
  - più facile da collaudare
  - manutenzione più semplice
  - la propagazione di effetti collaterali è ridotta
  - il software risulta più facile da estendere



Ripartizione verticale



La struttura dei dati

- La struttura dei dati definisce l'organizzazione, i metodi di accesso, il grado di associatività e le alternative di elaborazione per le informazioni
- Organizzazione e complessità dipendono dall'inventiva del progettista e dalla natura del problema
- Esistono diverse strutture di base che possono essere combinate
  - elemento scalare: entità elementare (bit, intero, reale, stringa)
  - vettore sequenziale: gruppo contiguo di elementi scalari omogenei
  - spazio n-dimensionale: vettore a 2 o più dimensioni
  - lista: gruppo di elementi connessi

La procedura software

- La gerarchia di controllo riassume le relazioni gerarchiche tra i moduli ma non descrive la logica interna dei moduli
- La procedura software si concentra sui dettagli dell'elaborazione specificando la sequenza degli eventi, i punti di decisione e i punti di chiamata ai moduli subordinati
- Un flow-chart è una rappresentazione grafica della procedura software

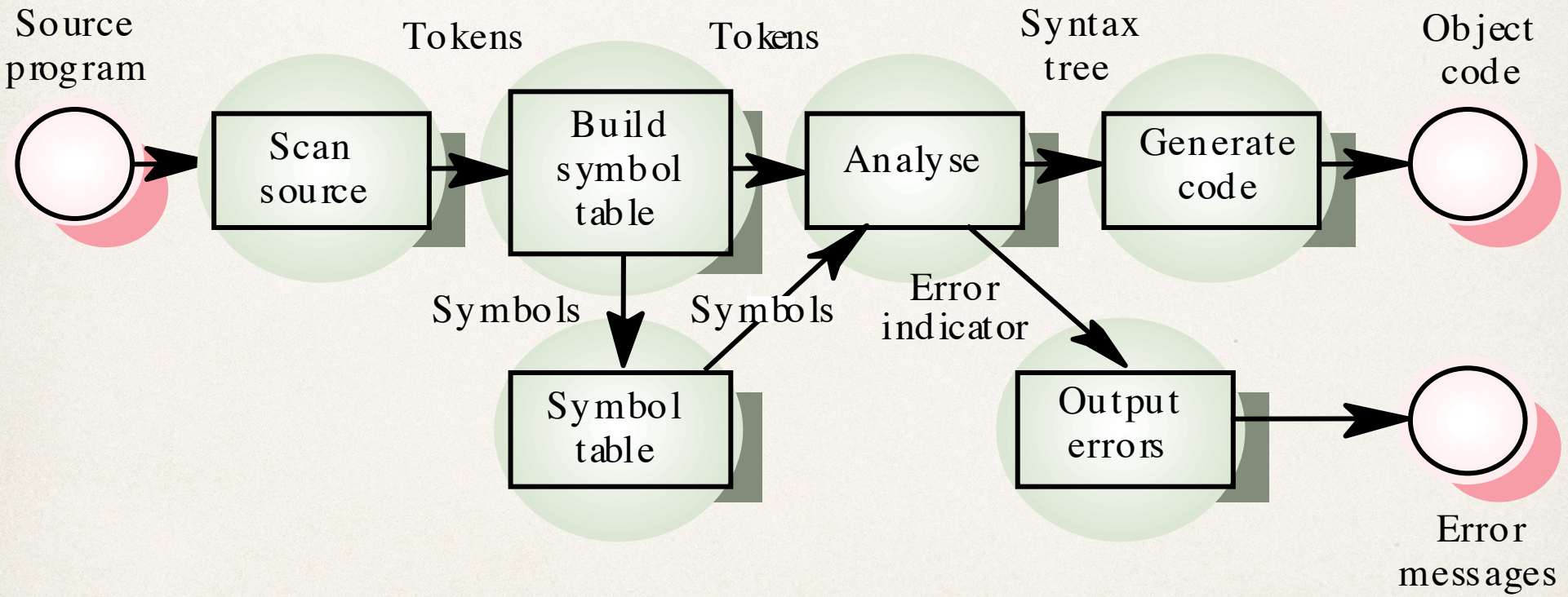
Information hiding

- Il principio dell'information hiding richiede che ciascun modulo sia definito in modo che le sue procedure e le informazioni locali su cui agisce non siano accessibili ad altri moduli
- l'interazione con gli altri moduli deve avvenire solo tramite la sua interfaccia
- Vantaggi: facilità di modifica di un modulo, perché non ci si deve preoccupare di effetti collaterali delle modifiche
- La definizione di moduli tramite la tecnica dell'information hiding può essere d'aiuto nell'identificare il punto di minimo costo per la modularità del sistema

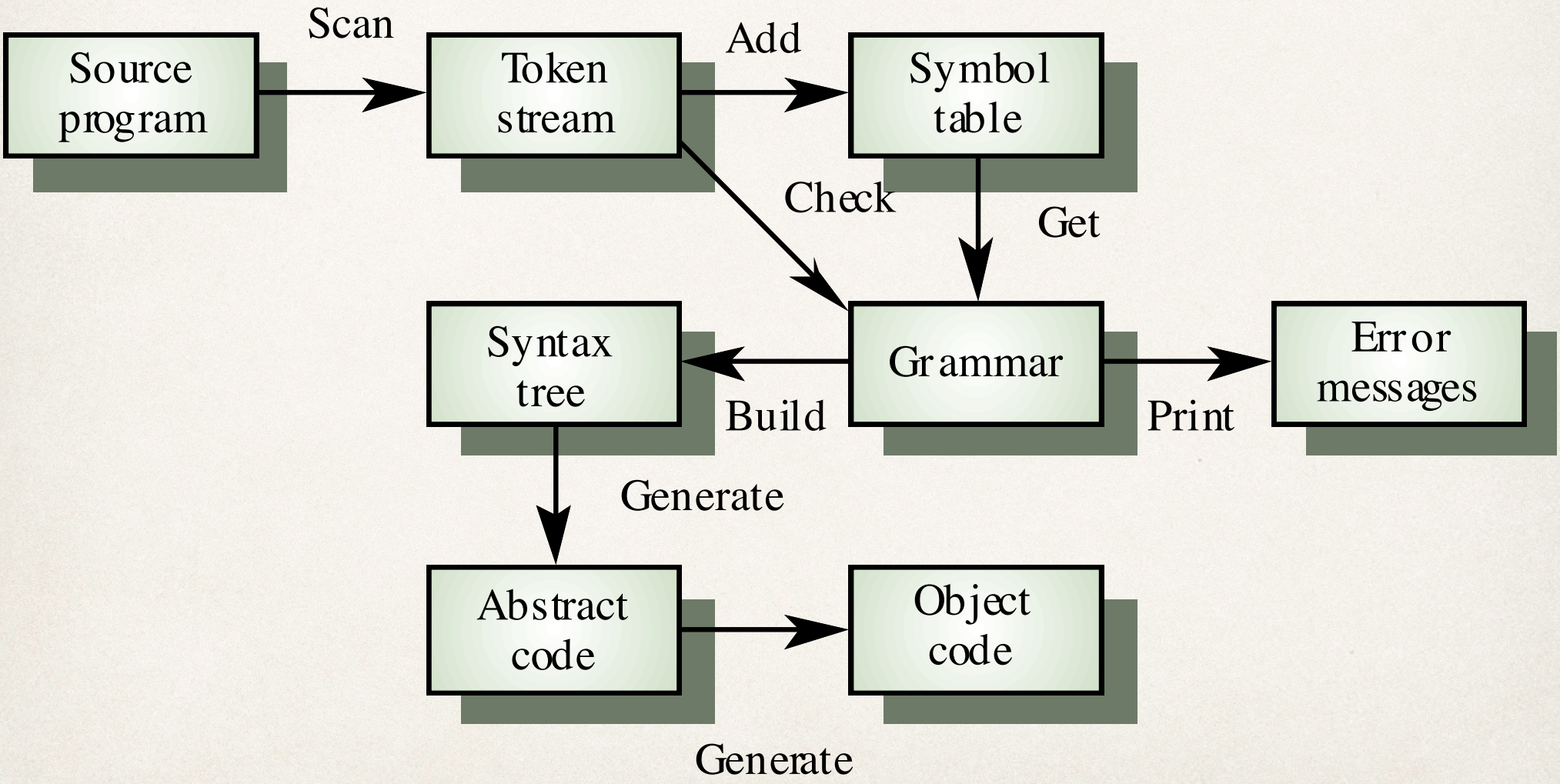
Strategie di progettazione

- Progettazione funzionale
- lo stato del sistema è centralizzato e condiviso tra funzioni che operano su quello stato
- Progettazione object-oriented
- il sistema è visto come un insieme di oggetti che interagiscono. Il sistema è de-centralizzato e ogni oggetto ha un proprio stato. Gli oggetti possono essere istanze di una classe e comunicano scambiando attraverso i propri metodi

Esempio: visione funzionale di un compilatore



Esempio: visione object-oriented di un compilatore



Progettazione mista

- C'è una complementarietà tra approccio funzionale e approccio object-oriented
- Di volta in volta un buon ingegnere del software dovrebbe scegliere l'approccio più appropriato per il sottosistema che sta progettando

Qualità della progettazione

- La qualità di un progetto è difficile da stabilire. Dipende da specifiche priorità di tipo organizzativo
  - un "buon" progetto potrebbe essere il più efficiente, il meno costoso, il più mantenibile, il più affidabile, ...
- Noi sottolineeremo gli attributi legati alla mantenibilità del progetto: coesione, accoppiamento, comprensibilità, adattabilità
  - un progetto mantenibile può essere adattato modificando funzionalità esistenti o aggiungendone di nuove; il progetto dovrebbe rimanere comprensibile; i cambiamenti dovrebbero avere effetto locale
- Le stesse caratteristiche di qualità si applicano sia alla progettazione funzionale che a quella orientata ad oggetti

Indipendenza modulare

- Per ottenere una modularità effettiva, i moduli devono essere indipendenti, cioè devono occuparsi di una funzione ben determinata nelle specifiche dei requisiti ed interagire con gli altri moduli solo tramite interfacce semplici
- L'indipendenza di un modulo può essere misurata in termini della sua coesione e dal suo accoppiamento con altri moduli

Coesione

- Un modulo è coeso quando esegue un numero di compiti limitato e coerente: nel caso ideale implementa una singola entità logica o una singola funzione
- Ogni modulo avrà un grado più o meno alto di coesione: occorre tenere alta la coesione media ed eliminare i moduli a bassa coesione
- La coesione è un attributo importante in quanto, qualora si dovesse effettuare un cambiamento al sistema, permette di mantenere il cambiamento locale ad una singola componente
- Si possono individuare livelli diversi di coesione

Coesione

- Proprietà interna al singolo componente
  - funzionalità vicine devono stare nello stesso componente
  - vicinanza per tipologia, algoritmi, dati in ingresso e in uscita
- Vantaggi di un alto grado di coesione
  - vantaggi rispetto al riuso e alla manutenibilità
  - riduce l'interazione fra componenti
  - migliore comprensione dell'architettura del sistema

Tipologie e livelli di coesione

- Coesione incidentale (debole)
  - le diverse parti di un componente sono semplicemente raggruppate insieme, ma non sono affatto correlate
- Associazione logica (debole)
  - vengono raggruppate le componenti che svolgono azioni simili (p.e. tutte le routine matematiche)
- Coesione temporale (debole)
  - vengono raggruppate le componenti che sono attivate nello stesso istante di tempo
- Coesione procedurale (debole)
  - vengono raggruppati tutti gli elementi di una componente che costituiscono una singola sequenza di controllo, cioè che vengono attivati in sequenza uno dopo l'altro

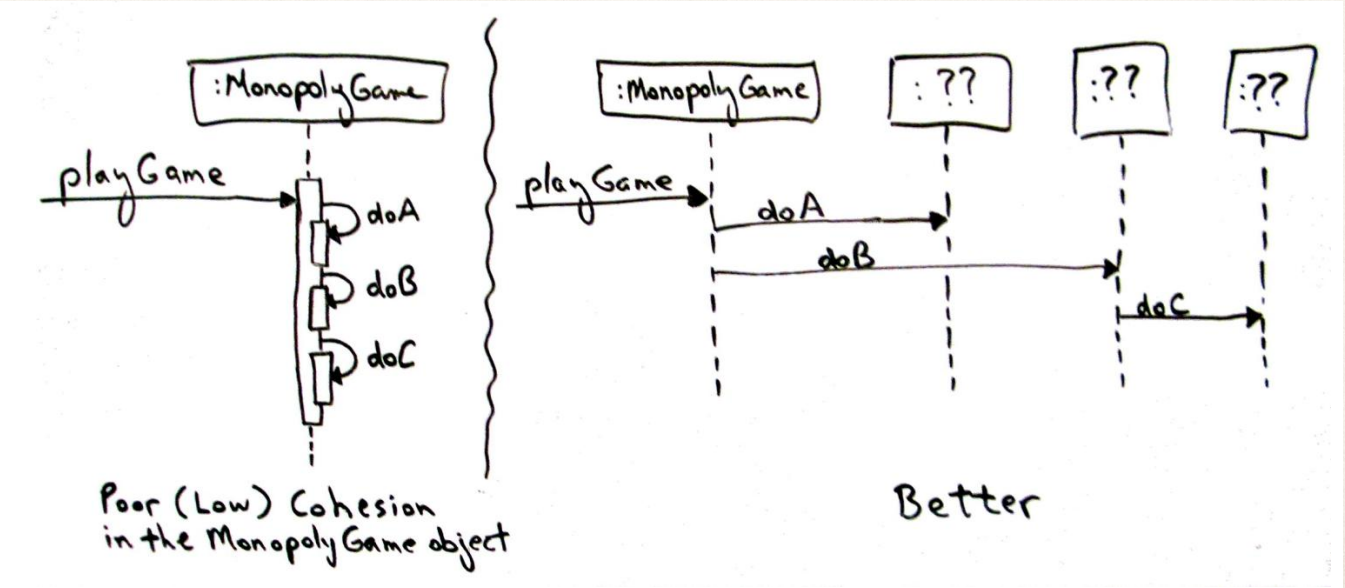
Tipologie e livelli di coesione

- Coesione di comunicazione (media)
  - tutti gli elementi di una componente operano su di uno stesso input o producono lo stesso output
- Coesione sequenziale (media)
  - l'output di una parte della componente è l'input di un'altra parte
- Coesione funzionale (forte)
  - ogni parte di una componente è necessaria solo per l'esecuzione di una singola funzione di quella componente
- Coesione d'oggetto (forte)
  - ogni operazione fornisce delle funzionalità per osservare o modificare gli attributi di un oggetto

Coesione

Quindi…

- oggetto Big, 100 metodi molto diversi per tipo di responsabilità (es: accesso ai db, log, calcoli matematici) è poco coeso ovvero poco "focalizzato" dal punto di vista funzionale
- oggetto small, 10 metodi con un solo tipo di responsabilità (es: accesso ai db) è molto coeso



Coesione come attributo di progetto in progettazione OO

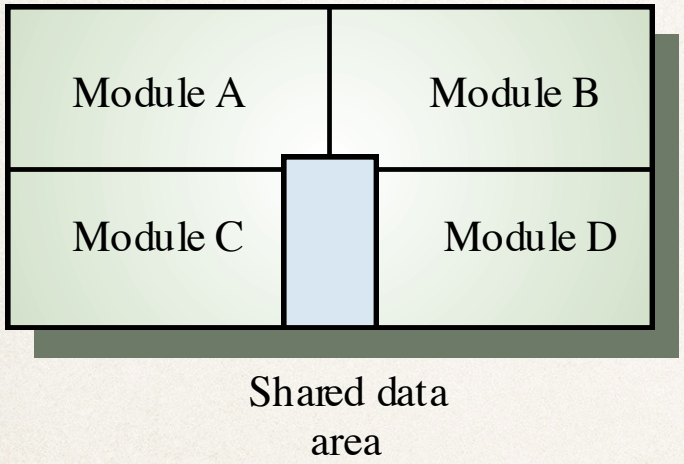
- Se si ereditano attributi da una superclasse si diminuisce la coesione
  - per comprendere una classe bisogna esaminare sia tutte le sue superclassi che le componenti della classe

Coupling (accoppiamento)

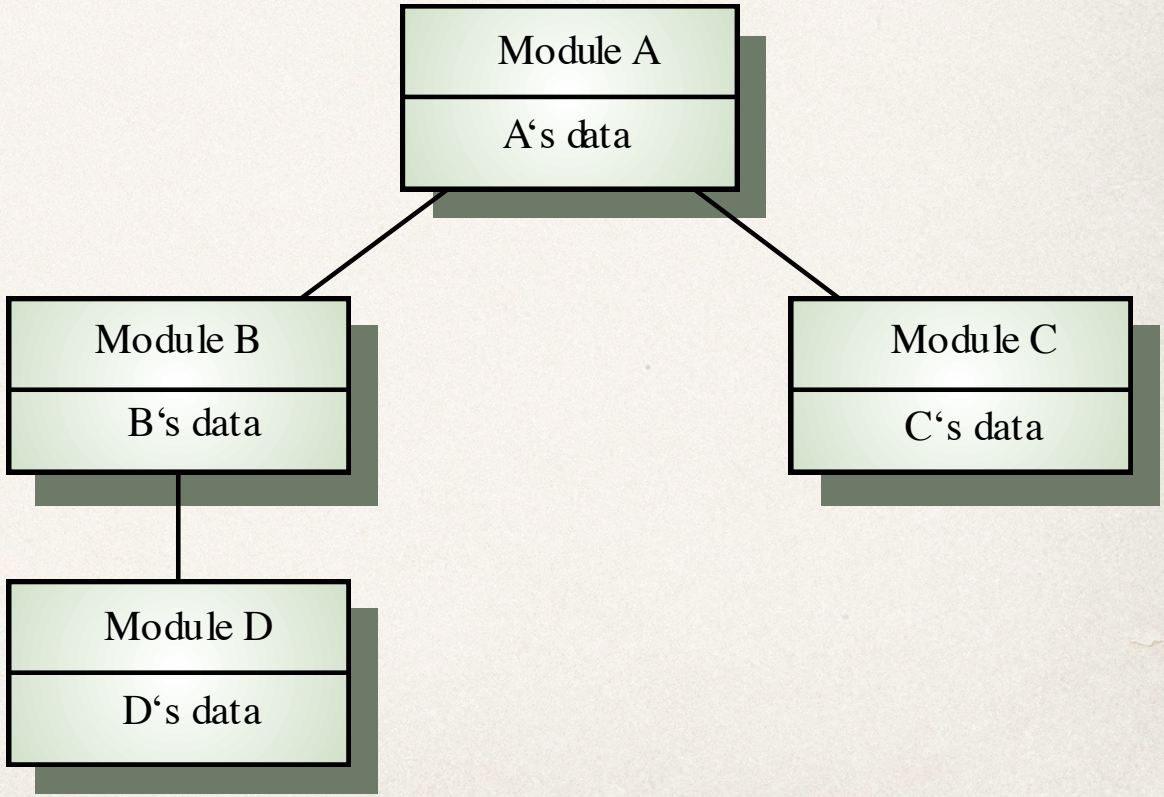
- Misura la “forza” di connessione tra le componenti di un sistema: quanto le componenti “si usano” tra di loro
- Loose coupling (accoppiamento lasco) implica che cambiamenti di una componente non hanno forti effetti sul comportamento delle altre
- Variabili condivise o lo scambio di informazioni di controllo porta ad accoppiamento stretto (tight coupling)
- L’accoppiamento lasco può essere ottenuto decentralizzando gli stati e realizzando la comunicazione con passaggio di parametri o di messaggi

Coupling

Accoppiamento stretto



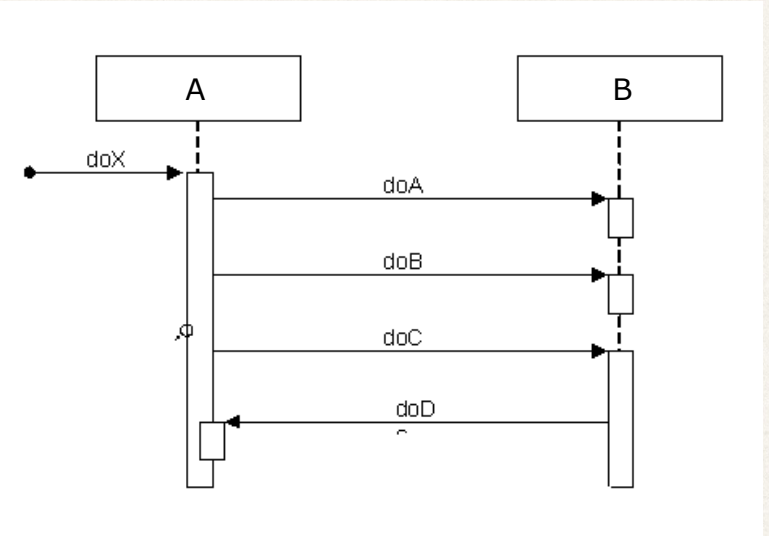
Accoppiamento lasco



Coupling

La classe A è accoppiata alla classe B. Infatti:

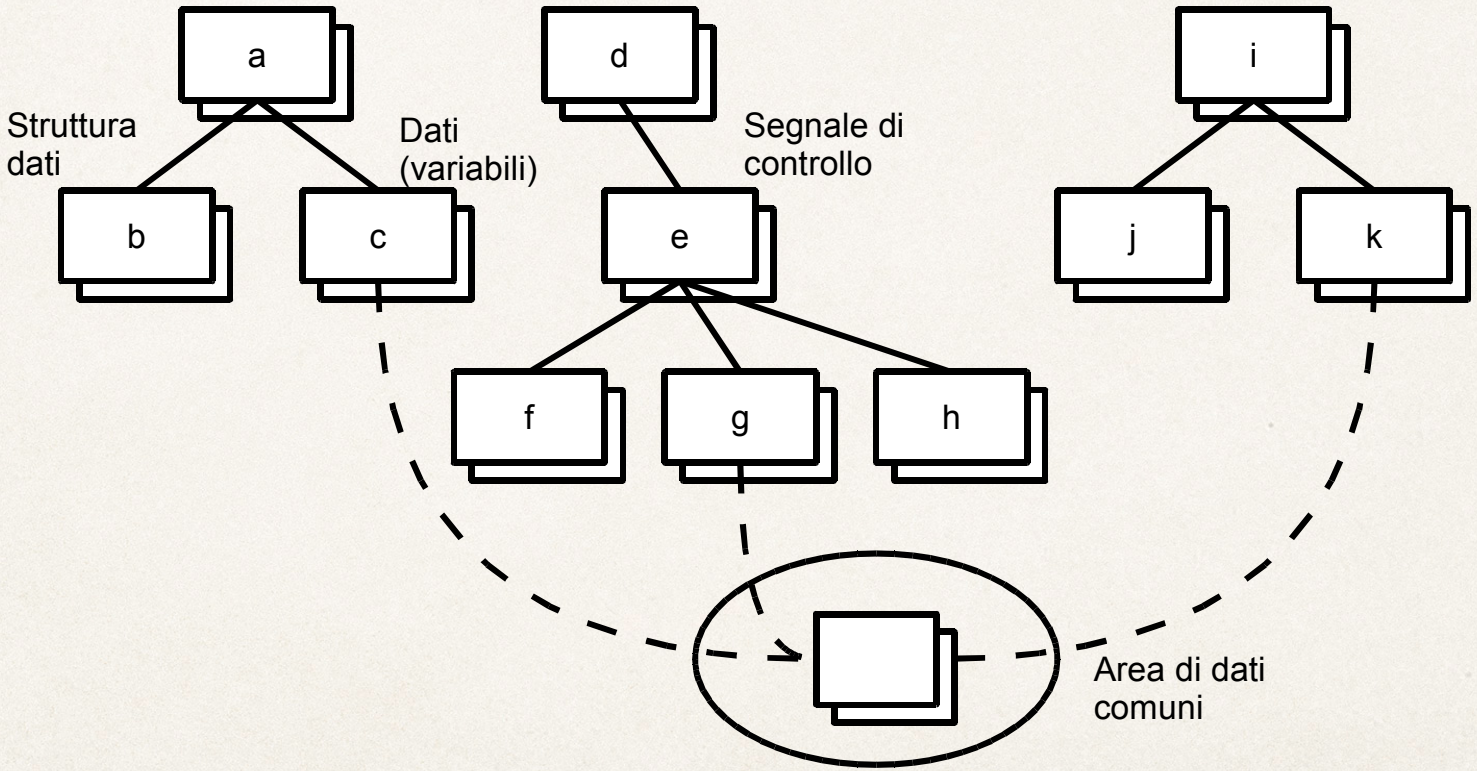
- A dipende da B
  - l'operazione doX di A dipende dalle operazioni doA, doB, doC di B
- cambiamenti di comportamento delle operazioni doA, doB, doC provocano cambiamenti di comportamento di doX
- il riuso della sola classe A non è possibile, è necessaria anche la classe B



Tipi di accoppiamento

- Basso accoppiamento
  - moduli diversi si scambiano parametri “semplici”
  - una alternativa si ha quando attraverso l’interfaccia di un modulo passa una struttura dati anziché dati semplici o atomici (accoppiamento a stampo)
- Accoppiamento di controllo
  - un “segnale di controllo” viene scambiato tra due moduli
- Accoppiamento comune
  - moduli diversi hanno dati in comune; questo tipo di accoppiamento è insidioso e può causare problemi difficili da diagnosticare

Tipi di accoppiamento



Accoppiamento

- In un sistema software gli elementi collaborano tra loro quindi è normale che ci siano dipendenze.
  - per cui l'accoppiamento è inevitabile ma va mantenuto il più basso possibile
- Nella programmazione OO le forme più comuni di accorpamento tra due tipi TypeX e TypeY sono
  - TypeX ha un attributo TypeY o referenzia una istanza TypeY
  - un oggetto TypeX richiama servizi di un oggetto TypeY
  - TypeX ha un metodo che referenzia oggetti di tipo TypeY (variabili locali, parametri o tipi ritornati)
  - TypeY è una interfaccia e TypeX implement direttamente o indirettamente questa interfaccia

Basso accoppiamento: come?

- Problema: come ridurre l'impatto dei cambiamenti, aumentare manutenibilità e riusabilità?
  - Assegnare la responsabilità in modo che l'accoppiamento rimanga basso
  - Usare questo principio per valutare le alternative progettuali. A parità di altre condizioni si consideri il progetto con il minore accoppiamento

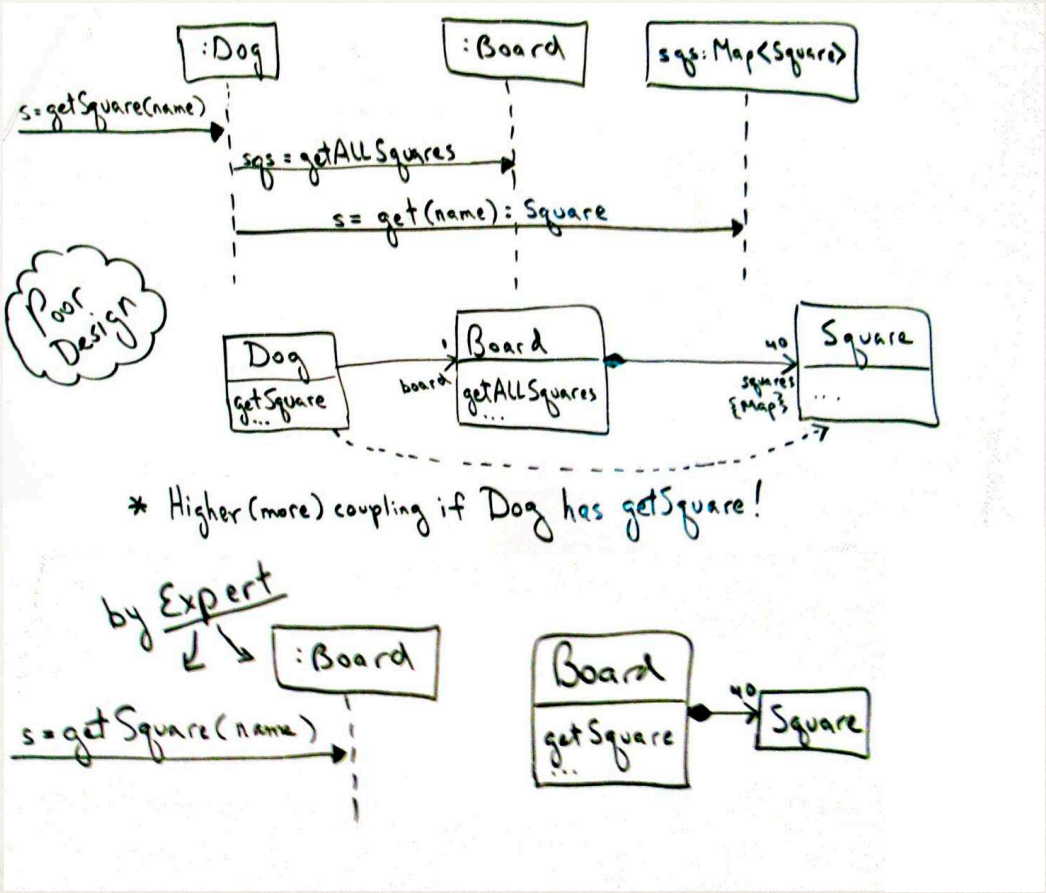
Basso accoppiamento: esempio

Monopoli: ho una classe "Board" che contiene tutte le "Square"

Problema: dato il nome di una Square chi deve avere la responsabilità di restituire la Square corrispondente?

1° soluzione (cattivo design)
Si crea una classe arbitraria "Dog" a cui si da questa responsabilità. Dog è accoppiato sia a Board che a Square (accoppiamento a 2 classi)

2° soluzione
Chi ha le informazioni per assolvere alla responsabilità? Board.
Board è accoppiato a Square (accoppiamento a 1 classe)



Accoppiamento ed ereditarietà

- I sistemi orientati ad oggetti sono sistemi “loosely coupled”
  - non condividono uno “stato”
  - gli oggetti comunicano passando messaggi attraverso i metodi
- Tuttavia una classe è accoppiata con la sua superclasse
  - i cambiamenti effettuati su una classe si propagano a tutte le sottoclassi

Comprensibilità

- Legata a diverse caratteristiche delle componenti
  - coesione
    - la componente può essere considerata da sola?
  - naming
    - i nomi usati sono significativi
  - documentazione
    - il progetto è ben documentato
  - complessità
    - si usano algoritmi complicati
- Di solito un'elevata complessità significa molte relazioni tra diverse parti del sistema e quindi scarsa comprensibilità
- Metriche di qualità di progettazione: misurano la complessità

Adattabilità

- Un progetto è adattabile quando
  - le sue componenti sono debolmente accoppiate
  - è ben documentato e la documentazione è aggiornata
  - c'è una corrispondenza stretta tra livelli della progettazione
  - ogni componente è auto-contenuta (coesione forte)
- Per adattare un disegno si devono poter individuare tutti i collegamenti tra componenti diverse, così che le conseguenze di una modifica a un componente possano essere analizzate

Adattabilità e ereditarietà

- L'ereditarietà migliora molto l'adattabilità
  - le componenti possono essere adattate senza cambiamenti attraverso la derivazione di una sotto-componente e la modifica solo di quest'ultima
- Tuttavia man mano che la profondità dell'ereditarietà cresce adattare il progetto diviene sempre più complesso
  - deve essere periodicamente rivisto e ristrutturato

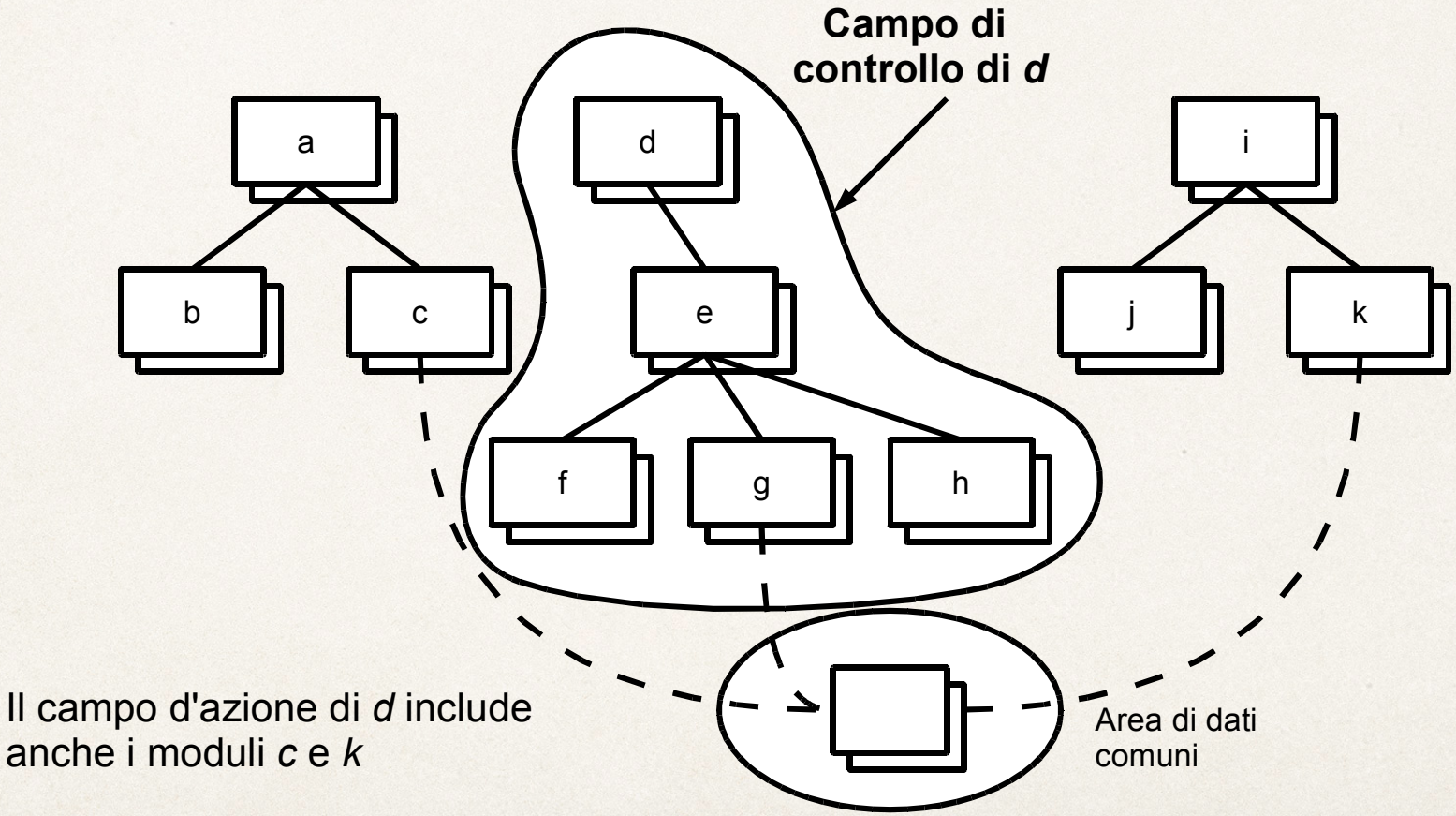
Come migliorare la modularità di un progetto

1. Studia la prima versione del progetto individuando le possibilità di aumentare la coesione e diminuire l'accoppiamento
   - ✦ esplosione di moduli: trasformazione di una funzionalità comune a più moduli in un modulo separato (aumento di coesione)
   - ✦ implosione di moduli: più moduli con interfacce reciproche complesse possono collassare in un unico modulo per diminuire l'accoppiamento generale del progetto
2. Diminuire il fan-out ad alto livello gerarchico e aumentare il fan-in a basso livello gerarchico
   - ✦ una struttura "ovale" migliora la ripartizione verticale del progetto (compiti sempre più elementari scendendo nella gerarchia)

Come migliorare la modularità di un progetto

3. Mantenere il campo di azione di un modulo entro il suo campo di controllo
   - **campo di azione**: insieme dei moduli che dipendono da una decisione presa localmente
   - **campo di controllo**: insieme dei moduli subordinati (anche indirettamente)
4. Studiare le interfacce per ridurre l'accoppiamento
5. Definire i moduli con funzioni prevedibili
   - modulo come scatola nera: si sa cosa fa anche se non si sa la struttura interna
   - evitare moduli con memoria, cioè il cui comportamento dipende dalla sequenza di azioni che hanno fatto
6. Evitare collegamenti patologici tra moduli
   - salti interni al modulo o simili

Campo di azione e di controllo



La Specifica del Progetto

- La Specifica del Progetto è il documento che descrive il progetto finale con il seguente formato
  - Descrizione della portata globale del progetto ricavata dalla specifica dei requisiti
  - Descrizione del progetto dei dati: struttura del DB, file esterni, dati interni, riferimenti fra dati
  - Descrizione dell'architettura con riferimento ai metodi utilizzati per ricavarla, rappresentazione gerarchica dei moduli
  - Progetto delle interfacce interne ed esterne, descrizione dettagliata dell'interazione utente/sistema con eventuale prototipo
  - Descrizione procedurale dei singoli componenti in linguaggio naturale

La Specifica del Progetto

- In ogni punto vanno inseriti riferimenti alla specifica dei requisiti da cui quella particolare parte del progetto è ricavata e va indicato quando le specifiche progettuali dipendono da un vincolo del sistema
- Nella descrizione delle interfacce va inclusa una prima versione della documentazione di collaudo con i test utili per stabilire la corretta implementazione dell'interfaccia
- In coda alla documentazione vanno aggiunti dati supplementari quali descrizione dettagliata degli algoritmi, eventuali alternative di realizzazione e una bibliografia che punti a tutti i documenti sviluppati nella fase di progetto