Copyright © 2004 Moreno Marzolla

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 2.5 Italy License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/2.5/it/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Progettazione

Crediti: E. Leonardi, Corso di Ingegneria del Software, Università di Ferrara, AA 2000/2001

La Progettazione
(detta anche la fase di *Design*)

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

I 3 Requisiti Progettuali
(McGlaughlin, 1991)

- 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 del 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

I Tool per la Progettazione

- Le tecniche di progettazione sono tutte basate su quattro tipi di tool così catalogabili:
  - 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 di Progetto / 1

- Il progettista non deve procedere col paraocchi
  - Bisogna sempre essere 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
- Evitare di riscoprire l'acqua calda
  - Riutilizzare ove possibile schemi o strutture dati già sviluppati in altri progetti
- Il progetto finale deve apparire uniforme ed integrato
  - Definire da subito regole di formato e di stile se si lavora in team
- Il progetto deve poter accogliere modifiche

Regole Empiriche di Progetto / 2

- Il software deve reagire in maniera controllata alle situazioni aberranti
  - Un software ben concegnato dovrebbe poter reagire a condizioni inusuali e arrestarti, se necessario, in modo regolato
- Il progetto NON E' la stesura del codice (e viceversa!)
  - Il livello di astrazione deve mantenersi più elevato
- La qualità del progetto e i metodi per mantenerla vanno decisi durante lo sviluppo del progetto e non alla fine
  - Questo vale sia per la qualità esterna (percepibile dall'utente) sia per quella interna (percepibile da sviluppatore e mantenitore)
- Al termine del progetto va sempre prevista una revisione formale che lo riesamini nel suo complesso
  - La revisione non deve perdersi nei dettagli sintattici

Fasi del progetto

- Comprensione del problema
  - Guardare al problema da diversi punti di vista per scoprire i requisiti
- Identificare una o più soluzioni
  - Valutare le possibili soluzioni e scegliere la più appropriata in base alle risorse a disposizione e all'esperienza del progettista
- Descrivere in astratto le soluzioni
  - Utilizzare notazioni grafiche, formali o intuitive per descrivere le componenti del progetto
- Ripetere il processo per ciascuna astrazione individuata, fino a quando il progetto è espresso in termini primitivi
  - Ossia: procedere a livelli, progettando in dettaglio le componenti e le loro sotto-componenti fino a quando si raggiunge un livello in cui le componenti non possono più essere ulteriormente suddivise

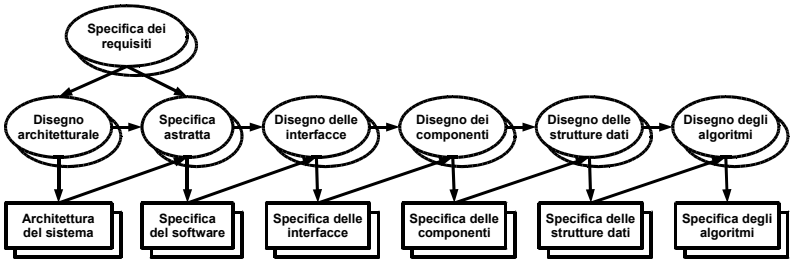
Astrazione

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

Raffinamento

- Il metodo di raffinamento utilizza tecniche di scomposizione per passare da astrazioni funzionali ad alto livello alle linee di codice
- Raffinamento e astrazione possono essere considerate attività di tipo complementare
  - 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 disegno / 1

- *Disegno Architetturale*
  - Identificare e documentare i sottosistemi e le loro relazioni
- *Specifica astratta*
  - Specifica i servizi forniti da ciascun sottosistema, e i vincoli sotto cui il sottosistema deve operare
- *Disegno delle interfacce*
  - Descrive l'interfaccia dei sottosistemi verso altri sottosistemi. La specifica delle interfacce deve essere non ambigua dato che deve consentire di definire i sottosistemi ignorando come operano al loro interno
- *Disegno dei componenti*
  - Allocare i servizi ai diversi componenti e definire le interfacce di questi componenti

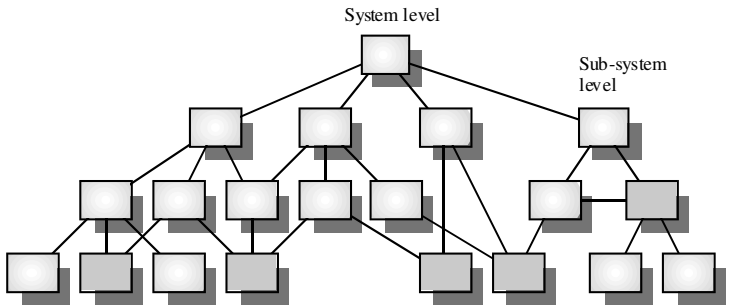
Fasi del disegno / 2

- *Disegno delle strutture dati*
  - Specificare come sono fatte le strutture dati
- *Disegno degli algoritmi*
  - Specificare gli algoritmi utilizzati

Progettazione top-down

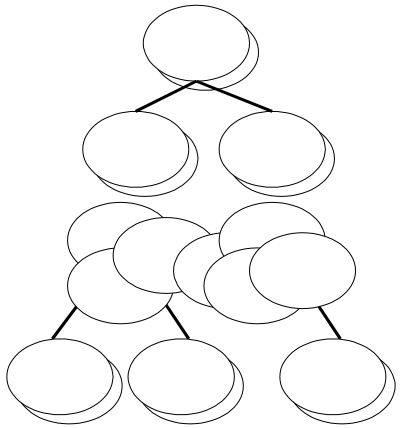
- E' un modo per affrontare l'attività di progettazione dei sistemi
- Il problema viene partizionato ricorsivamente in sotto-problemi fino a che si identificano dei sottoproblemi trattabili
- In teoria, la progettazione top-down richiede di iniziare con il componente radice della gerarchia, e procedere verso il basso livello dopo livello
  - In pratica, la progettazione di sistemi di grosse dimensioni non è mai completamente top-down. Alcuni rami sono sviluppati prima di altri. I progettisti riutilizzano l'esperienza (e talvolta le componenti) durante il processo di progettazione

Schema top-down



Strategie di decomposizione

- Top-down
  - Decomposizione di problemi
- Bottom-up
  - Composizione di soluzioni
- Sandwich
  - Soluzione naturale

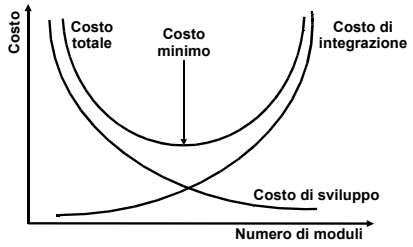


Modularità e Integrazione

- Un sistema consistente in un unico blocco monolitico di software è difficile (impossibile) da comprendere, implementare e mantenere
- Individuare moduli con funzionalità definite e limitate e con interfacce ben definite è l'unica strada che permette ad un programma di essere intellettualmente gestibile
  - Attenzione: Una eccessiva modularità richiede grossi sforzi per la integrazione

Costi di sviluppo e integrazione

- *C(p)* = complessità percepita per la risoluzione 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 si applica in pratica?

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

Criteri di Meyer (1988)

- Scomponibilità
  - Un metodo che permette la scomposizione del problema in sottoproblemi riduce la complessità
- Componibilità
  - Un metodo che permette l'assemblamento di componenti pre-esistenti migliora la produttività
- Comprensibilità
  - Un modulo le cui interfacce con altri moduli siano minime è di più facile costruzione e modificabilità
- Continuità
  - Modifiche ai requisiti del sistema che comportano solo modifiche a singoli moduli e non all'architettura generale sono di facile controllo
- Protezione
  - Se effetti anomali in un modulo non si propagano si ha un miglioramento della mantenibilità

L'Architettura del Software / 1

- Il termine si riferisce alla "struttura complessiva del software e ai modi in cui tale struttura sorregge l'integrità concettuale di un sistema" (Shaw, 1995)
  - In parole povere, descrive la struttura gerachica dei moduli di un programma, le loro interazioni e la struttura dei dati da essi manipolati
- Uno degli scopi della progettazione è la definizione di una architettura appropriata per il sistema da sviluppare
  - E' opportuno tenere presente alcune proprietà dell'architettura

L'Architettura del Software / 2

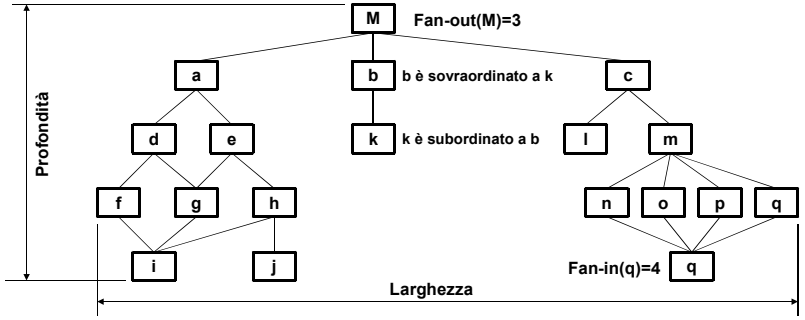
- *Proprietà strutturali*
  - L'architettura deve descrivere le componenti del sistema e il loro assemblaggio (interazioni)
  - Es, gli oggetti incapsulano dati e operazioni, e interagiscono (comunicano) tramite chiamata di metodi
- *Proprietà extrafunzionali*
  - L'architettura deve esplicitare in che modo vengono soddisfatti i requisiti di prestazioni, affidabilità, sicurezza, modificabilità, eccetera
- *Affinità*
  - Il progetto dell'architettura deve permettere di usare strutture e schemi simili per progetti simili

L'Architettura del Software / 3

- Specificate le proprietà precedenti, l'architettura può essere presentata usando uno o più modelli
  - *Modelli strutturali*: presentano l'architettura come come una collezione organizzata di componenti
  - *Modelli schematici*: tentano di individuare schemi progettuali ricorrenti in applicazioni dello stesso tipo
  - *Modelli dinamici*: descrivono gli aspetti comportamentali dell'architettura, indicando in che modo il sistema muta a seguito di eventi esterni
  - *Modelli di processo*: mettono in evidenza 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

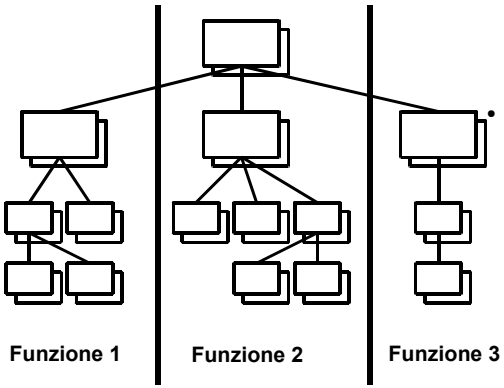
- Descrive l'organizzazione gerarchica dei moduli di un programma.
  - Un esempio classico è la struttura ad albero call and return



Ripartizione strutturale

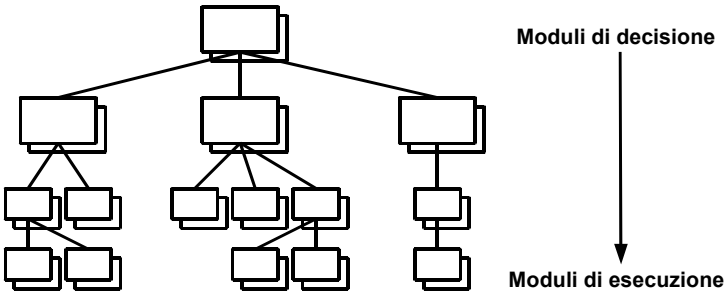
- In un sistema a organizzazione gerarchica, la struttura del programma può essere partizionata in senso orizzontale o verticale

Ripartizione orizzontale



- L'approccio più semplice definisce tre partizioni
  - Input, trasformazioni, output
- Vantaggi
  - Software più facile da collaudare
  - La manutenzione del software è 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 di dati elementare
    - bit, numero intero, numero reale, stringa
  - vettore sequenziale: gruppo contiguo di elementi scalari omogenei
  - spazio n-dimensionale (o matrice): vettore a 2 o più dimensioni
  - lista: gruppo di elementi connessi

La Procedura Software

- La gerarchia di controllo riassume le relazione gerarchiche tra i moduli ma non descrive la logica interna di ciascun modulo
- 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
- Il flow-chart è una rappresentazione grafica della procedura software

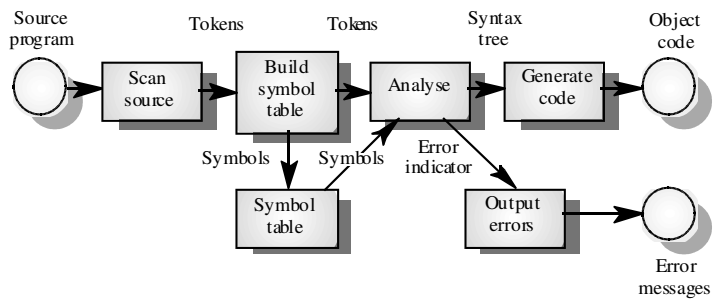
Information Hiding

- Il principio dell'information hiding (occultamento dell'informazione) 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
- Il vantaggio principale sta nella facilità di modifica di ciascun modulo poiché non ci si deve preoccupare degli 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

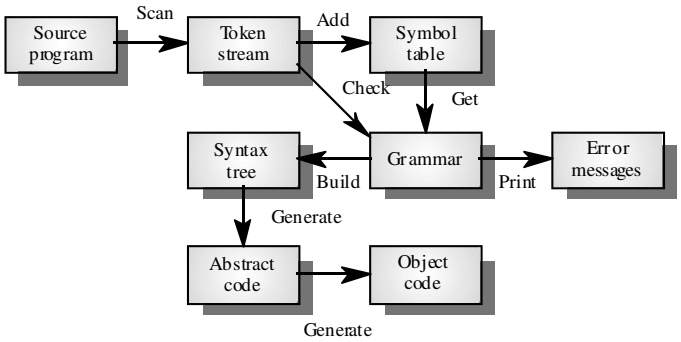
Strategie di design

- Disegno funzionale
  - Il sistema viene progettato da un punto di vista funzionale. Lo stato del sistema è centralizzato e viene condiviso dalle funzioni che operano su tale stato
- Disegno object-oriented
  - Il sistema viene visto come una collezione di oggetti che interagiscono tra loro. Lo stato del sistema è decentralizzato e ciascun oggetto gestisce il proprio stato. La comunicazione avviene tramite chiamata di metodi di altri oggetti.

Visione funzionale di un compilatore



Visione Object-Oriented di un compilatore



Qualità di un progetto

- La qualità di un progetto è un concetto elusivo. La qualità dipende da specifiche priorità di tipo organizzativo
  - Un "buon" progetto potrebbe essere il più efficiente, il meno costoso, il più mantenibile, il più affidabile...
- Gli attributi che descriviamo ora hanno a che fare con la mantenibilità del progetto
  - Un progetto mantenibile può essere adattato modificando funzionalità esistenti o aggiungendone di nuove. Il progetto dovrebbe rimanere comprensibile, e i cambiamenti dovrebbero avere effetto locale

Indipendenza Modulare

- Per ottenere una modularità effettiva i moduli devono essere indipendenti, ossia 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

Attributi di un sistema
(e/o dei moduli che ne fanno parte)

- Coesione
  - Una misura di quanto le diverse componenti sono in relazione
- Accoppiamento
  - E' un indicatore della forza del legame tra le componenti del sistema
- Comprensibilità
- Adattabilità
  - Quanto è facile cambiare il disegno?

Coesione

- Un modulo è coeso quando esegue un numero di compiti *limitato* e *coerente* (caso ideale: un solo compito)
- Ogni modulo avrà un grado più o meno alto di coesione: va tenuta alta la coesione media e vanno eliminati i moduli a bassa coesione
- La coesione è una caratteristica desiderabile, dato che quando viene apportato un cambiamento, gli effetti sono limitati al singolo componente coeso
- Diversi livelli di coesione sono stati identificati

Livelli di coesione / 1

- Coesione incidentale (debole)
  - Le diverse parti di un componente sono semplicemente "impacchettate" assieme, ma non sono affatto correlate
- Associazione logica (debole)
  - Vengono raggruppate le componenti che svolgono compiti simili (es, 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, ossia che vengono attivati in sequenza uno dopo l'altro

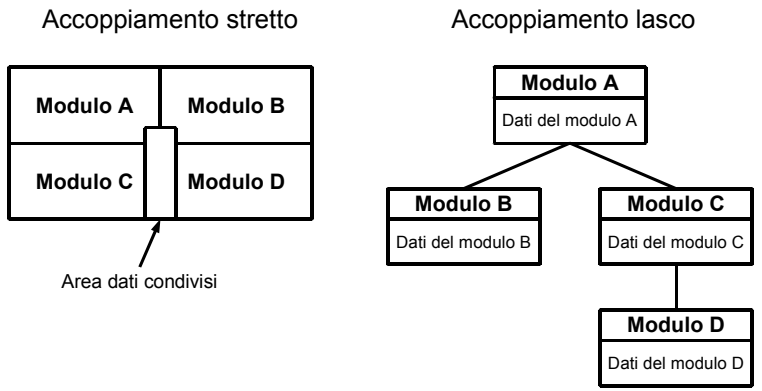
Livelli di coesione / 2

- Coesione di comunicazione (media)
  - Tutti gli elementi di una componente operano sullo stesso input e 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 per l'esecuzione di una singola funzione
- Coesione di oggetto (forte)
  - Ogni operazione fornisce delle funzionalità per osservare o modificare gli attributi di un oggetto

Accoppiamento

- Una misura della "forza" dell'interconnessione tra componenti del sistema
- Accoppiamento debole significa che i cambiamenti di una componente difficilmente avranno effetti su un'altra componente
- Variabili condivise portano ad un accoppiamento stretto
- L'accoppiamento debole può essere raggiunto mediante decentralizzazione dello stato (come negli oggetti) e comunicazione tramite passaggio di parametri o di messaggi

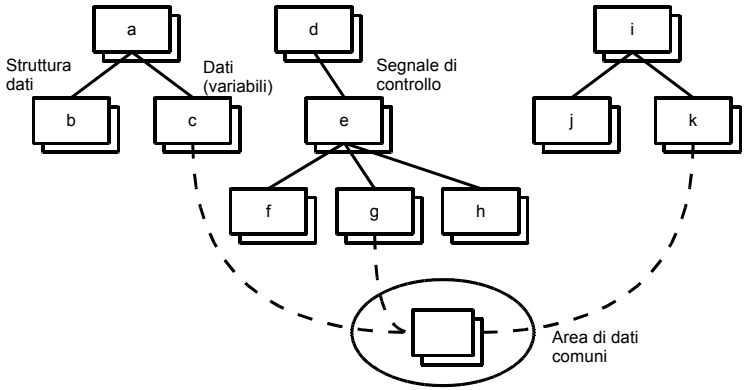
Accoppiamento



Tipi di accoppiamento

- Basso accoppiamento
  - Moduli diversi si scambiano parametri “semplici”
  - Una alternativa, l'*accoppiamento a stampo*, si ha quando attraverso l'interfaccia di un modulo passa una struttura dati, anziché dati semplici o atomici
- *Accoppiamento di controllo*
  - Un “segnale di controllo” viene scambiato tra due moduli, ad esempio d ed e nell'esempio che segue
- *Accoppiamento comune*
  - Moduli diversi hanno dati in comune. Questo tipo di accoppiamento è il più insidioso e può causare problemi difficili da diagnosticare

Tipi di accoppiamento



Accoppiamento ed ereditarietà

- I sistemi object-oriented sono debolmente accoppiati perché non ci sono variabili condivise (almeno, non dovrebbero essercene...) e gli oggetti comunicano tramite invocazione di metodi
- Tuttavia, una classe è strettamente accoppiata alle sue super-classi. Cambiamenti fatti negli attributi o operazioni di una super-classe si propagano a tutte le sotto-classi. Simili cambiamenti devono essere attentamente controllati

Comprensibilità

- Questo attributo è riferito a diverse caratteristiche di una componente
  - *Coesione*. La componente può essere considerata da sola?
  - *Nomi*. Sono stati usati nomi significativi per le variabili?
  - *Documentazione*. Il progetto è ben documentato?
  - *Complessità*. Sono stati usati algoritmi complessi?
- Informalmente, alta complessità implica molte relazioni tra diverse parti del progetto. Quindi il tutto diventa difficilmente comprensibile

Adattabilità

- Un progetto è adattabile se:
  - Le sue componenti sono *debolmente accoppiate*
  - E' *ben documentato*, e la documentazione è aggiornata
  - C'è una *corrispondenza ovvia tra livelli di progetto diversi*. Cioè, da un modello del sistema dovrebbe essere semplice individuare oggetti correlati in un altro modello dello stesso sistema (ricordare che un sistema si può rappresentare da diversi punti di vista con modelli diversi)
  - Ogni componente è *auto-contenuta* (forte coesione)
- Per adattare un disegno, deve essere possibile individuare tutti i collegamenti tra componenti diverse, in modo che le conseguenze dovute a modifiche di una componente possano essere analizzate

Adattabilità ed ereditarietà

- L'ereditarietà migliora l'adattabilità.
  - Le componenti possono essere adattate senza cambiamenti attraverso derivazione di una sotto-componente e modifica solo di quest'ultima
- Tuttavia, man mano che la profondità dell'albero di ereditarietà cresce, il progetto diviene via via più complesso (e quindi meno adattabile).
  - Deve essere periodicamente rivisto e ristrutturato

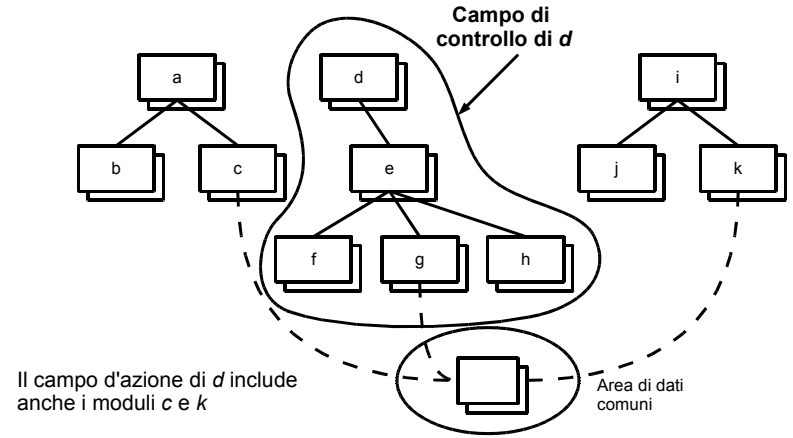
Come rendere un progetto più modulare / 1

- Studiare 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
- Diminuire il fan-out ad alto livello gerarchico e aumentare il fan-in a basso livello gerarchico
  - Una struttura gerarchica "ovale" migliora la ripartizione verticale del progetto (compiti via via più elementari scendendo nella gerarchia)

Come rendere un progetto più modulare / 2

- Mantenere il campo di azione di un modulo entro il suo campo di controllo
  - Campo di azione: inseme dei moduli che dipendono da una decisione presa localmente
  - Campo di controllo: insieme dei moduli subordinati (anche indirettamente)
- Studiare le interfacce per ridurre l'accoppiamento
- Definire i moduli con funzioni prevedibili
  - Un modulo dovrebbe funzionare come una scatola nera: si sa cosa fa anche senza conoscerne la struttura interna
  - Evitare moduli con memoria, ossia moduli il cui comportamento dipende dalla sequenza di azioni che hanno fatto
- Evitare collegamenti patologici tra moduli
  - Salti interni al modulo o simili

Campo d'azione e di controllo



La Specifica del Progetto / 1

- La specifica del Progetto è il documento che descrive il progetto finale con questo 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 / 2

- In ogni punto vanno inseriti riferimenti alla specifica dei requisiti da cui quella particolare parte del progetto è stata 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 di algoritmi, eventuali alternative di realizzazione e una bibliografia che punti a tutti i documenti sviluppati nella fase di progetto