Dipartimento di Informatica e Sistemistica
Antonio Ruberti

“Sapienza” Università di Roma

Introduzione alla Progettazione del Software

Corso di Tecniche di Programmazione

Laurea in Ingegneria Informatica

(Canale di Ingegneria delle Reti e dei Sistemi Informatici - Polo di Rieti)

Anno Accademico 2007/2008

Prof. Paolo Romano

Si ringrazia il Prof. Giuseppe De Giacomo per aver reso disponibile il materiale didattico sul quale si basano queste slides

Introduzione alla Progettazione del Software

Ciclo di Vita

Ciclo di vita del software

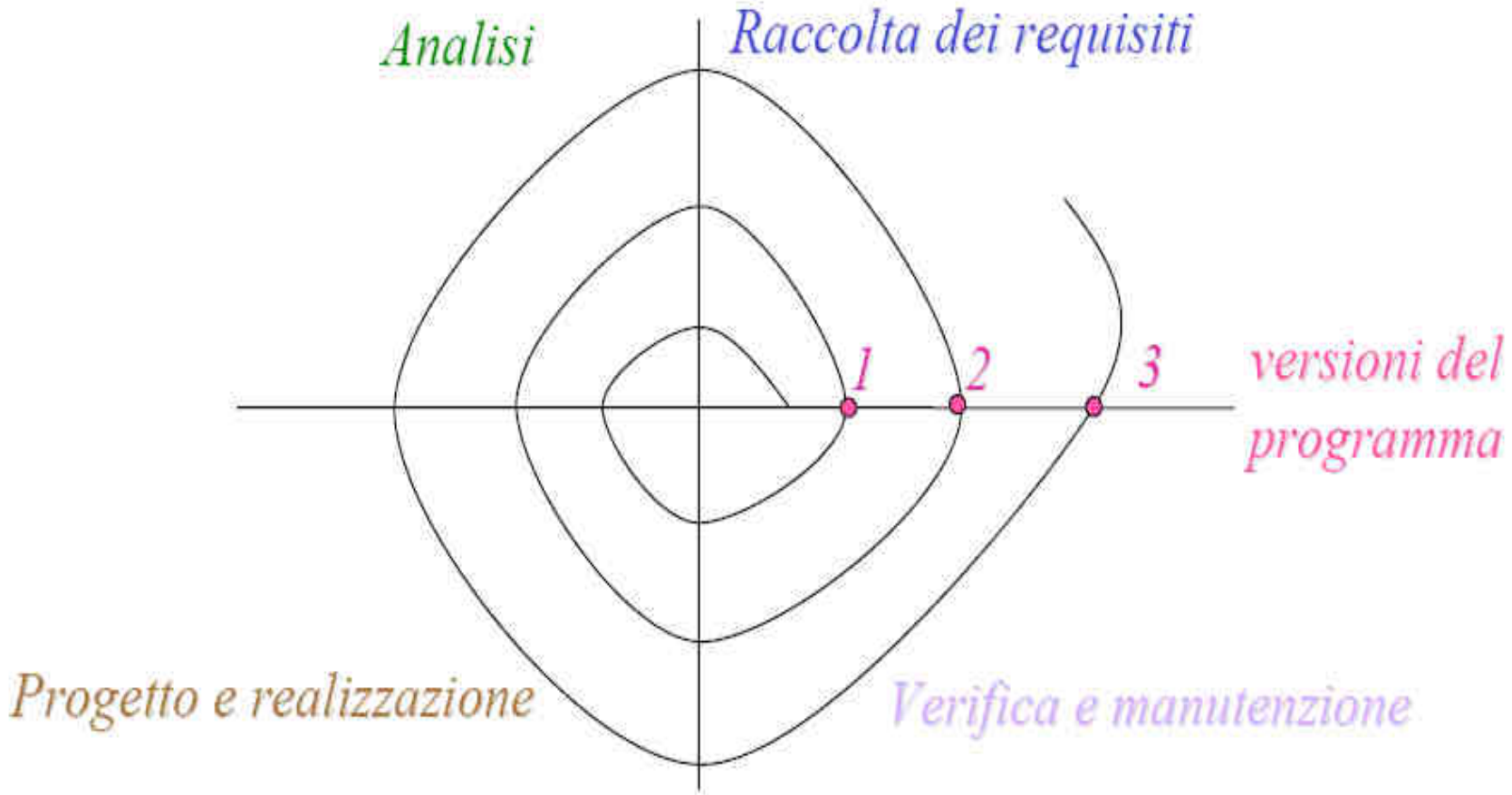
1. Studio di fattibilità e raccolta dei requisiti
   - valutare costi e benefici
   - pianificare le attività e le risorse del progetto
   - individuare l'ambiente di programmazione (hardware/software)
   - raccogliere i requisiti
2. Analisi dei requisiti
   - si occupa del cosa l'applicazione dovrà realizzare
   - descrivere il dominio dell'applicazione e specificare le funzioni delle varie componenti: lo schema concettuale
3. Progetto e realizzazione
   - si occupa del come l'applicazione dovrà realizzare le sue funzioni
   - definire l'architettura del programma
   - scegliere le strutture di rappresentazione
   - scrivere il codice del programma e produrre la documentazione

Ciclo di vita del software

1. Studio di fattibilità
2. Analisi dei requisiti
3. Progetto e realizzazione
4. Verifica
   - Il programma svolge correttamente, completamente, efficientemente il compito per cui è stato sviluppato?
5. Manutenzione
   - Controllo del programma durante l'esercizio
   - Correzione e aggiornamento del programma

Se, necessario, si torna alla fase di raccolta dei requisiti

Ciclo di vita del software (modello a spirale)



Introduzione alla Progettazione del Software

Qualità

Qualità esterne ed interne

- Qualità esterne:
  - **Sono le qualità visibili agli utenti del sistema**
  - Percepibili anche da chi non è specialista
  - Non richiedono l'ispezione del codice sorgente

- Qualità interne:
  - **Sono le qualità che riguardano gli sviluppatori**
  - Valutabili da specialisti
  - Richiedono conoscenza della struttura del programma

A volte la distinzione fra qualità esterne ed interne non è perfettamente marcata

Fattori di qualità del SW

ESTERNE	INTERNE
Correttezza	Efficienza
Affidabilità	*Strutturazione*
Robustezza	*Modularità*
Sicurezza	*Comprensibilità*
Innocuità	Verificabilità
Usabilità	Manutenibilità
Estendibilità	Portabilità
Riusabilità	
Interoperabilità	

Esempio

Considerare le seguenti proprietà di un impianto Hi-Fi e stabilire quali di esse sono esterne e quali sono interne.

1. il tempo dedicato al collaudo,
2. la fedeltà sonora,
3. la probabilità di malfunzionamenti nel primo anno di utilizzazione,
4. la dimensione del trasformatore per l'alimentazione,
5. l'ergonomia dei comandi,
6. la linearità della risposta in frequenza.

Soluzione

- Interne: 1, 3, 4, 6
- Esterne: 2, 5
- Nota: 3 potrebbe ragionevolmente essere considerata esterna

Correttezza, Affidabilità, Robustezza, Sicurezza, Innocuità

- **Correttezza** (è fondamentale)
  il software fa quello per il quale è stato progettato
- **Affidabilità**: si può fare affidamento sulle funzionalità del software: vincoli rispettati → risultati disponibili
- **Robustezza**: comportamento accettabile anche nel caso di situazioni non previste nella specifica dei requisiti
- **Innocuità**: il sistema non entra in certi stati (pericolosi)
- **Sicurezza**: riservatezza nell'accesso alle informazioni

Usabilità

- Enfasi sull'utente
- Psicologia cognitiva
- Fattori fisici/ergonomici
- Mentalità dell'utente
- Interfacce grafiche/visuali
- Viene, spesso, totalmente ignorata

Estendibilità

Facilità con cui il SW può essere adattato a modifiche delle specifiche

Nota: Importantissimo in grandi programmi

Nota: Due principi per l'estendibilità:

- Semplicità di progetto
- Decentralizzazione nell'architettura del SW

Riusabilità

Facilità con cui il SW può essere re-impiegato in applicazioni diverse da quella originaria

Nota: Evita di reinventare soluzioni

Nota: Richiede alta compatibilità

- Libreria di componenti riutilizzabili
- Progetto più generale possibile
- Documentazione
- Maggiore affidabilità

Interoperabilità

- Facilità di interazione con altri moduli al fine di svolgere un compito più complesso
- Problemi tecnologici e semantici
- Favorisce la riusabilità

Nota sulle qualità esterne

- Correttezza
- Estendibilità
- Riusabilità

sono qualità chiave, fortemente favorite da un approccio orientato agli oggetti

Efficienza

- Si riferisce al "peso" che il software ha sulle risorse del sistema
  - Tempo di esecuzione
  - Utilizzo di memoria
- Teoria della complessità
  → corso di *Algoritmi e Strutture di Dati*
  - limiti asintotici
  - caso medio
  - caso peggiore
- Simulazioni (e.g., teoria delle reti di code)
- Legata alle prestazioni (quest'ultime percepibili dall'utente, quindi qualità esterne)

Strutturazione, Modularità

- Strutturazione

  Capacità del SW di riflettere con la sua struttura le caratteristiche del **problema** trattato e delle **soluzioni** adottate

- Modularità

  Grado di organizzazione del SW in parti ben specificate ed interagenti

Comprensibilità

Capacità del SW di essere compreso e controllato anche da parte di chi non ha condotto il progetto

- si applica sia al software che al processo
- facilitata da:
  - strutturazione
  - modularità
- la comprensibilità del software facilita l'analisi della correttezza ed il riuso

Verificabilità

- La possibilità di verificare che gli obiettivi proposti siano stati raggiunti

- È una caratteristica sia del processo che del prodotto
- Facile: il codice soddisfa gli standard di codifica?
- Difficile: il codice fa ciò che deve fare? (vedi correttezza)

Manutenibilità, Portabilità

- Manutenibilità
  - Facilità nell'effettuare modifiche
- Portabilità
  - Facilità nell'operare su diverse piattaforme modifiche
    - linguaggi compilabili su più piattaforme
    - macchine virtuali (html/java)

Nota sulle qualità interne

- Strutturazione
- Modularità

sono caratteristiche di estremo interesse, favorite dall'approccio orientato agli oggetti

Qualità in Contrasto

- Non tutte le qualità possono essere massimizzate
- Alcune sono intrinsecamente **in contrasto** fra loro. Ad esempio:
  - Usabilità e sicurezza
  - Efficienza e portabilità
- È necessario scegliere un adeguato bilanciamento

Tendenza nello sviluppo di applicazioni SW

+ Complessità delle informazioni da gestire
+ Progetti di medio-grandi dimensioni
+ Eterogeneità degli utenti
- Durata media dei sistemi
+ Bisogno di interventi di manutenzione
- Costi di produzione e tempi di produzione
+ Qualità del prodotto finito

Principi guida nello sviluppo del software

In base alle tendenze descritte precedentemente si attualmente si considerano fondamentali nello sviluppo del software i seguenti principi:

- **Rigore e formalità**
  - Lo sviluppo del software è una attività creativa che va accompagnata da un approccio rigoroso (o addirittura formale: in logica o matematica) permette di realizzare prodotti affidabili, controllarne il costo, aumentare la fiducia nel loro corretto funzionamento.
- **Separazione degli interessi**
  - Affrontare separatamente i diversi aspetti per dominare la complessità
- **Modularità**
  - Realizza la separazione degli interessi in 2 fasi:
    - Tratta i dettagli di singoli modi in modo separato
    - Tratta separatamente dai dettagli interni dei singoli moduli le relazione che sussistono tra i moduli stessi
- **Astrazione**
  - Identifica aspetti fondamentali ed ignora i dettagli irrilevanti
- **Anticipazione del cambiamento**
  - Per favorire l'estendibilità e il riuso
- **Generalità**
  - Ricerca si soluzioni generali
- **Incrementalità**
  - Per anticipare feedback dell'utente per facilitare verifiche di correttezza per predisporsi alla estendibilità e al riuso

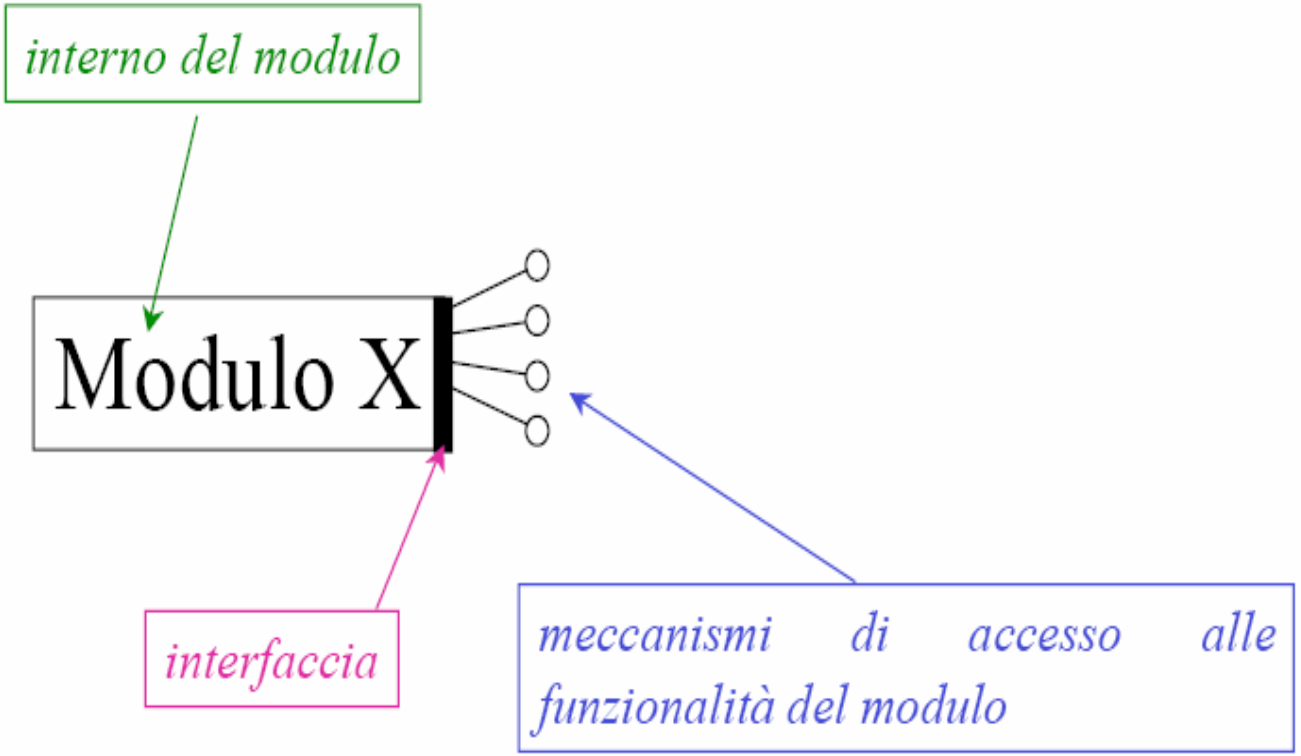
Introduzione alla Progettazione del Software

Modularizzazione

Modularizzazione

- Principio secondo il quale il software è strutturato secondo unità, dette appunto **moduli**
- Un modulo è una unità di programma con le seguenti caratteristiche:
  - ha un obiettivo chiaro
  - ha relazioni strutturali con altri moduli
  - offre un insieme ben definito di servizi agli altri moduli (server)
  - può utilizzare servizi di altri moduli (client)
- Principio fondamentale sia per la strutturazione sia per la modularità
- Concettualmente alla base del concetto di architettura client/server del software

Componenti di un modulo



Principi per la modularità

- Principio di unitarietà *(incapsulamento, alta coesione)*
  - un modulo deve corrispondere ad una unità concettuale ben definita e deve incorporare tutti gli aspetti relativi a tale unità concettuale
- Poche interfacce *(basso accoppiamento)*
  - un modulo deve comunicare con il minor numero di moduli possibile (quelli necessari)
- Poca comunicazione *(basso accoppiamento)*
  - un modulo deve scambiare meno informazione possibile (quella necessaria!) con gli altri moduli
- Comunicazione chiara *(interfacciamento esplicito)*
  - informazione scambiata predeterminata e più astratta possibile
- Occultamento di informazioni inessenziali *(information hiding)*
  - le informazioni che non devono essere scambiate devono essere gestite privatamente dal modulo

Principi per la modularità

In sintesi, i quattro dogmi della modularizzazione sono:

- ***Alta coesione*** *(omogeneità interna)*
- ***Basso accoppiamento*** *(indipendenza da altri moduli)*
- ***Interfacciamento esplicito*** *(chiare modalità d'uso)*
- ***Information hiding*** *(poco rumore nella comunicazione)*

Esempi

- Esempi negativi:
  - *Bassa coesione*

    Allo stesso sportello degli uffici postali si pagano i bollettini di conti corrente e contemporaneamente si ritirano le pensioni (servizi disomogenei!)
  - *Alto accoppiamento*

    Per usufruire di alcuni servizi delle circoscrizioni, occorre acquistare delle marche da bollo dal tabaccaio (accoppiamento tra tabaccaio e ufficio circoscrizionale)
- Esempi positivi:
  - *Interfacciamento esplicito*

    In molte pagine web (ad esempio per prenotazione posti in un teatro), le informazioni da fornire sono chiaramente espresse mediante campi da riempire
  - *Information hiding*

    L'utente non conosce esattamente cosa è memorizzato nella banda magnetica delle carte di credito

Esempio di cattiva modularizzazione

```
public class Server {
   public static int alfa;
   // ALTO ACCOPPIAMENTO: impone al cliente di
   // usare alfa in maniera coerente
   public static bool Cerca(Lista x){
      // BASSO INTERFACCIAMENTO ESPLICITO:
      // non è esplicito che alfa è il valore cercato
      // BASSA COESIONE: non effettua il controllo
      di
      // lista vuota
      do {  if (x.info == alfa)  return true;
            x = x.next;
         }
      while (x != null);
      return false;
   }
}
```

```
public class Client {
   static Lista MiaLista() { ... }
   public static void main(String[] args) {
      Lista s = MiaLista();
       if (s != null) {
         // ALTO ACCOPPIAMENTO: il cliente
         // deve controllare che la lista non sia vuota
         System.out.print("Inserisci un intero: ");
         Server.alfa = InOut.readInt();
         if (Server.Cerca(s))
            // ALTO ACCOPPIAMENTO: il cliente deve
            //  definire e usare alfa come vuole il server
            System.out.print(Server.alfa+" presente");
         else
            System.out.print(Server.alfa+" non
      presente ");
       }
   }
}
```

Esempio di buona modularizzazione

```
public class Server {
   public static bool Cerca(Lista x, int alfa){
      // ALTO INTERFACCIAMENTO ESPLICITO:
      // è esplicito che alfa è il valore cercato
      // ALTA COESIONE: effettua il controllo di
      // lista vuota
      while (x != null)
      {  if (x.info == alfa)  return true;
         else x = x.next;
      }
      return false;
   }
}
```

```
public class Client {
   static Lista MiaLista() { ... }
   public static void main(String[] args) {
      Lista s = MiaLista();
      System.out.print("Inserisci un intero: ");
      int alfa = InOut.readInt();
      if (Server.Cerca(s, alfa))
            // BASSO ACCOPPIAMENTO: il cliente si
            // concentra sulla richiesta del servizio
           System.out.print(Server.alfa+" presente");
      else
           System.out.print(Server.alfa+" non
      presente");}
}
```

Note sull'esempio

- Anche semplicissimi programmi Java possono essere modularizzati male
- Esempio:
  - **Cattiva modularizzazione** (server con basso interfacciamento esplicito, bassa coesione, alto accoppiamento; conseguenze negative sul client)
  - **Buona modularizzazione** (riprogettazione del server e del client)
  - I due programmi sono **entrambi corretti**, ma hanno **diversa qualità**

Effetti di una buona modularizzazione

- Il progetto, le competenze, ed il lavoro possono essere distribuiti (i moduli sono indipendenti tra loro)
- Rilevare eventuali errori nel software è più semplice (si individua il modulo errato e ci si concentra su di esso)
- Correggere errori e modificare il software è più semplice (si individua il modulo da modificare e ci si concentra su di esso)
- Attenzione: l'efficienza del programma ne può risentire!