ITS INFORMATION AND COMMUNICATIONS TECHNOLOGY ACADEMY

MODULO: Progettazione

UNITÀ: Progettazione.1

Prof. Toni Mancini

Dipartimento di Informatica

Sapienza Università di Roma

Slides Progettazione.1.I (S.Progettazione.1.I)

Progettazione

Unità 1

Cenni di Ingegneria del Software

Versione 2023-06-01

Indice

Queste slide sono composte dalle seguenti sottounità:

ITS INFORMATION AND COMMUNICATIONS TECHNOLOGY ACADEMY

MODULO: Progettazione

UNITÀ: Progettazione.1

Prof. Toni Mancini

Dipartimento di Informatica

Sapienza Università di Roma

Slides Progettazione.1.I.1 (S.Progettazione.1.I.1)

Progettazione

Unità 1

Cenni di Ingegneria del Software

Obiettivi

Versione 2023-06-01

Cosa impareremo in questo modulo?

Impareremo a progettare applicazioni di reali e di dimensioni non banali

- impossibile scrivere direttamente il codice
- gran parte del tempo deve essere speso nella comprensione delle funzionalità e dei dati di interesse e delle loro interrelazioni
- non esiste una soluzione unica, bisogna fare delle scelte ragionate
- non esiste una ricetta che "basta applicare"
- il metodo deve essere molto generale, e va applicato e adattato intelligentemente al problema in esame

Cosa impareremo in questo modulo? (2)

Di conseguenza, impareremo a:

- ragionare logicamente
- considerare le conseguenze delle nostre scelte
- valutare le alternative
- scomporre i problemi complessi in sottoproblemi

Esempio

Progettare una applicazione che permetta di mantenere informazioni su un insieme di contatti (telefonici e email).

Per ogni contatto è necessario mantenere:

- nome e cognome
- numeri di telefono di casa, ufficio e mobile
- indirizzo email

I contatti possono appartenere a gruppi. L'applicazione deve permettere all'utente di aggiungere un nuovo contatto, modificare i dati di un contatto, cancellare un contatto, assegnare/rimuovere contatti a/da un gruppo e ricercare i contatti per nome e/o cognome.

$\Longrightarrow$ Non c'è bisogno di progettare molto. Probabilmente siamo già in grado di scrivere un programma in Python o Java per l'applicazione!

Esempio (2)

Si vuole sviluppare un sistema che permetta ad una banca di gestire i conti correnti dei clienti, i loro investimenti, oltre che la propria rete di promotori finanziari.

Il sistema deve tenere traccia di tutti gli acquisti e vendite di azioni, obbligazioni, etc. effettuati dai clienti, e deve poter calcolare in tempo reale la valorizzazione corrente del loro portafoglio. Inoltre, l'applicazione deve assistere i promotori finanziari nella scelta degli strumenti finanziari più adeguati da proporre ai clienti, e deve permettere ai responsabili di agenzia di controllare la professionalità dei promotori.

$\Longrightarrow$ Impossibile scrivere direttamente il programma per l'applicazione!

Esempio (3)

Impossibile scrivere direttamente il programma per l'applicazione!

Progetto software complesso:

- pool di Ingegneri del SW, progettisti, programmatori: 1.5 anni
  - tempo per capire il problema: 6 mesi (33%)
  - tempo per la progettazione: 9 mesi (50%)
  - tempo per la realizzazione: 3 mesi (solo il 17%)
- più: tempo per test&verifica, etc. etc.

Oggetto di questo modulo: saper realizzare questo genere di progetti come applicazioni software da dispiegare sul cloud

ITS INFORMATION AND COMMUNICATIONS TECHNOLOGY ACADEMY

MODULO: Progettazione

UNITÀ: Progettazione.1

Prof. Toni Mancini

Dipartimento di Informatica

Sapienza Università di Roma

Slides Progettazione.1.I.2 (S.Progettazione.1.I.2)

Progettazione

Unità 1

Cenni di Ingegneria del Software

Contesto Organizzativo

Versione 2023-06-01

Il contesto organizzativo

Attori nella progettazione del software:

- Committente
- Esperti del domino
- Analista
- Progettista
- Programmatore
- Utente finale
- Manutentore

Il contesto organizzativo (2)

Esempio:

Il Comune di XYZ intende automatizzare la gestione delle informazioni relative alle contravvenzioni elevate sul suo territorio. In particolare, intende dotare ogni vigile di una app per smartphone che gli consenta di comunicare al sistema informatico il veicolo a cui è stata comminata la contravvenzione, il luogo in cui è stata elevata e la natura dell'infrazione. Il sistema informatico provvederà a notificare, tramite posta ordinaria, la contravvenzione al cittadino interessato. Il Comune bandisce una gara per la realizzazione e manutenzione del sistema, che viene vinta dalla Ditta ABC.

Quali sono gli attori coinvolti in questa applicazione software?

Il contesto organizzativo (3)

- Committente: Comune di XYZ
- Esperto del domino: funzionario del Comune, o altro professionista designato, esperto del Codice della Strada
- Utenti finali: vigili
- Analisti, progettisti, programmatori, manutentori: personale della ditta ABC

ITS INFORMATION AND COMMUNICATIONS TECHNOLOGY ACADEMY

MODULO: Progettazione

UNITÀ: Progettazione.1

Prof. Toni Mancini

Dipartimento di Informatica

Sapienza Università di Roma

Slides Progettazione.1.I.3 (S.Progettazione.1.I.3)

Progettazione

Unità 1

Cenni di Ingegneria del Software

Ciclo di Vita del Software

Versione 2023-06-01

Ciclo di vita del software

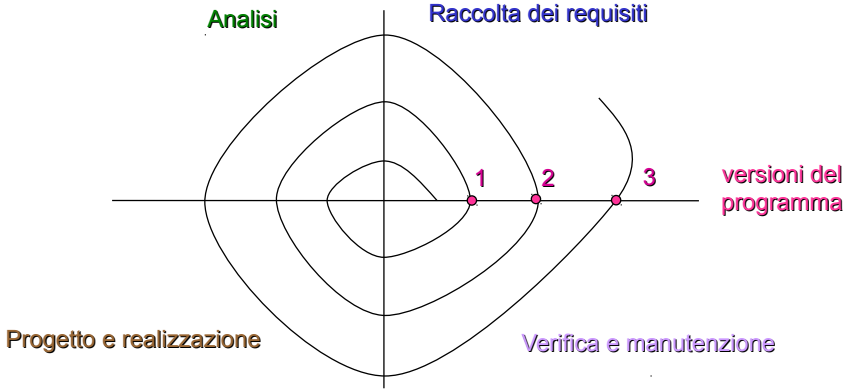
1. Studio di fattibilità e raccolta dei requisiti
   - valutare costi e benefici
   - pianificare le attività e le risorse del progetto
   - individuare l'ambiente di programmazione (hardware/software)
   - raccogliere i requisiti
2. Analisi dei requisiti
   si occupa del cosa l'applicazione dovrà realizzare
   - descrivere il dominio dell'applicazione e specificare le funzioni delle varie componenti nello 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 (2)

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

Al termine di ogni fase, se necessario, si può tornare indietro

Modello a spirale del ciclo di vita del SW



Analisi dei requisiti: cosa è?

- Fase del ciclo di sviluppo del software caratterizzata da:
  Input: requisiti raccolti
  Output: schema concettuale dell'applicazione
- Obiettivo: costruire un modello dell'applicazione che sia:
  - completo
  - preciso
  - leggibile
  - traducibile in un programma eseguibile
- concentrarsi su cosa e non su come
  (indipendenza da aspetti realizzativi/tecnologici)

Analisi dei requisiti: a cosa serve?

- Analizzare i requisiti:
  - coglie le loro implicazioni
  - li specifica con l'obiettivo di formalizzarli e di eliminare:
    - incompletezze
    - inconsistenze
    - ambiguità
- Creare un modello (schema concettuale) che sarà un riferimento per tutte le fasi successive del ciclo di vita del software
- Verificare i requisiti con l'utente finale
- Prendere decisioni fondamentali sulla strutturazione e sulla modularizzazione del software
- Fornire la specifica delle funzionalità da realizzare

Output della fase di Analisi: lo schema concettuale

Lo schema concettuale è composto da

- diagrammi in opportuni linguaggi grafici di modellazione
- documenti di specifica

che descrivono completamente e precisamente il sistema da realizzare secondo diverse prospettive:

- quali sono e come sono organizzati i dati di interesse
- quali sono le funzionalità da offrire e a quali utenti
- come evolvono i dati di interesse nel tempo
- ...

Unified Modelling Language (UML)

- Vedremouna metodologia di Analisi basata su un linguaggio unificato di modellazione:

  Unified Modelling Language (UML)

- UML fornisce costrutti per modellare gran parte degli aspetti, sia statici che dinamici (dati, funzioni, evoluzione), usando un approccio Object Oriented

Unified Modelling Language (UML) (2)

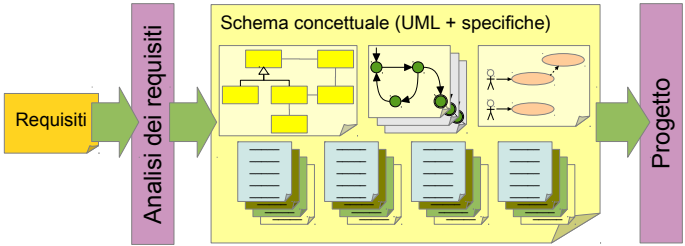
- UML è un linguaggio estremamente vasto (ed anche estendibile), che permette di produrre diagrammi per descrivere:
  - l'organizzazione degli oggetti (dati) di interesse in gerarchie di classi (diagrammi delle classi e degli oggetti)
  - l'evoluzione permessa da singoli oggetti nel tempo (diagrammi degli stati e transizioni)
  - le funzionalità offerte dal sistema e i relativi attori (utenti umani o sistemi esterni) che possono utilizzarle (diagrammi degli use case)
  - le interazioni tra oggetti e processi del sistema (diagrammi di sequenza, di collaborazione, delle attività)
  - l'architettura del sistema (diagrammi dei componenti, diagrammi di deployment)
  - altro

Approccio unificante alla progettazione del software

Analisi $\rightarrow$ Progetto $\rightarrow$ Realizzazione

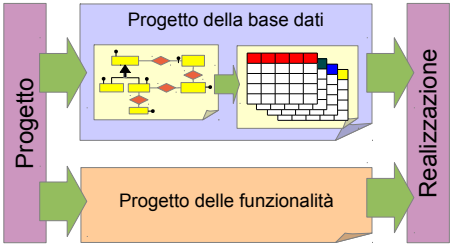
Approccio unificante alla progettazione del software (2)

Analisi



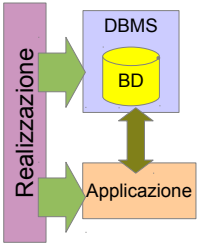
Approccio unificante alla progettazione del software (3)

Progetto



Approccio unificante alla progettazione del software (4)

Realizzazione



ITS INFORMATION AND COMMUNICATIONS TECHNOLOGY ACADEMY

MODULO: Progettazione

UNITÀ: Progettazione.1

Prof. Toni Mancini

Dipartimento di Informatica

Sapienza Università di Roma

Slides Progettazione.1.I.4 (S.Progettazione.1.I.4)

Progettazione

Unità 1

Cenni di Ingegneria del Software

Qualità del Software

Versione 2023-06-01

Fattori di Qualità del Software

Qualità esterne

- correttezza
- affidabilità
- robustezza
- sicurezza
- innoquità
- usabilità
- estendibilità
- riusabilità
- interoperabilità

Qualità interne

- efficienza
- modularità
- verificabilità
- manutenibilità
- portabilità

Qualità esterne: percepibili anche da chi non è specialista; non richiedono l'ispezione del codice sorgente.

Qualità interne: valutabili da specialisti; richiedono conoscenza della struttura del programma.

Fattori di Qualità del Software (2)

A volte la distinzione non è perfettamente marcata.

Qualità Esterne

- Correttezza: il sistema fa quello per il quale è stato progettato?
- Affidabilità: il sistema garantisce che i risultati saranno disponibili se tutti i vincoli sono rispettati?
- Robustezza: il sistema ha un comportamento corretto anche nel caso di situazioni non specificate?
- Sicurezza: è garantita la riservatezza nell'accesso alle informazioni?
- Innocuità: il sistema non entra mai in certi stati (pericolosi)?
- Usabilità: il sistema è utilizzabile in modo semplice e proficuo?

Qualità Esterne (2)

- Estendibilità: il sistema può essere facilmente adattato a modifiche delle specifiche?
  - Cruciale in grandi programmi!
  - Due principi per l'estendibilità:
    - Semplicità di progetto
    - Decentralizzazione nell'architettura del sw
- Riusabilità: il sistema può essere facilmente re-impiegato in applicazioni diverse da quella originaria?
  - evita di reinventare soluzioni
  - richiede alta compatibilità.

  Chiavi per il successo:
  - progetto/uso di librerie di componenti riutilizzabili
  - progetto più generale possibile
  - buona documentazione
  - maggiore affidabilità.

Qualità Esterne (3)

- Interoperabilità: il sistema è facilmente in grado di interagire in modo semplice con altri sistemi/moduli al fine di svolgere un compito più complesso?
  - Problemi tecnologici e semantici
  - Favorisce la riusabilità.

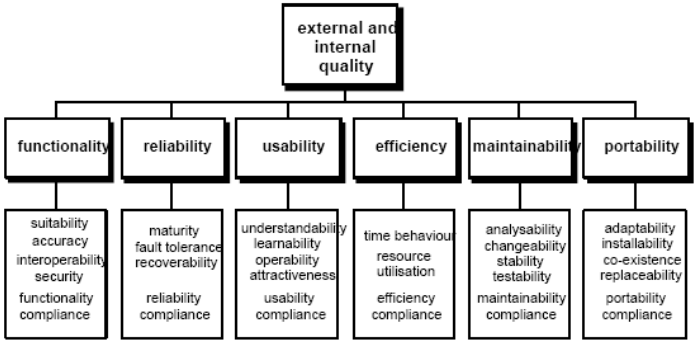
Qualità Interne

- Efficienza: legata alle prestazioni
  - in termini di:
    - tempo di esecuzione
    - utilizzo di memoria
  - approcci:
    - teoria della complessità
      (v. corsi di algoritmi e di calcolabilità e complessità)
    - limiti asintotici
    - caso medio e caso peggiore
    - simulazioni.
- 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

Qualità Interne (2)

- 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. Una buona comprensibilità facilita l'analisi della correttezza ed il riuso.
  Chiavi per il successo:
  - strutturazione
  - modularità.
- Verificabilità: possibilità di verificare che gli obiettivi proposti siano stati raggiunti.
  È una caratteristica sia del processo che del prodotto:
  - versione facile: il codice soddisfa gli standard di codifica?
  - versione difficile: il codice fa ciò che deve fare? (vedi correttezza)
- Manutenibilità: facilità nell'effettuare modifiche
- Portabilità: linguaggi compilabili su più piattaforme, utilizzo di componenti (ad es. DBMS) disponibili su più piattaforme.

Lo Standard ISO/IEC 9126



Standard del 1991, aggiornato nel 2001. Nel 2011 rimpiazzato da ISO/IEC 25010:2011.

Misura delle Qualità

- Ogni qualità deve essere valutata attraverso alcune proprietà misurabili in modo oggettivo e quantitativo possedute dalle entità
- Differenti entità possono essere collocate su una scala di valori in funzione dei livelli misurati per questi attributi.

Tendenza nello Sviluppo di Applicazioni SW

+ complessità delle informazioni da gestire
+ progetti di medio-grandi dimensioni
+ eterogeneità degli utenti
– durata dei sistemi
+ bisogno di interventi di manutenzione
– costi di produzione e tempi di produzione
+ qualità del prodotto finito

Modularizzazione e orientazione agli oggetti sono tecniche introdotte per rispondere a queste esigenze.

Modularizzazione

Principio secondo il quale il software è strutturato secondo unità, dette 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.

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 (2)

Quattro linee quida:

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

Principi per la Modularità (3)

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 dell'Ufficio Anagrafe, occorre acquistare delle marche da bollo dal tabaccaio (accoppiamento tra tabaccaio e ufficio anagrafe).

Principi per la Modularità (4)

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: Buona e Cattiva Modularizzazione

Anche semplicissimi programmi possono essere modularizzati male.

Esempio: programma Java per cercare un elemento in una lista.

Cattiva modularizzazione : server con basso interfacciamento esplicito, bassa coesione, alto accoppiamento
$\Longrightarrow$ conseguenze negative sul client.

Buona modularizzazione : riprogettazione del server e del client.

I due programmi sono entrambi corretti, ma hanno diversa qualità.

Esempio: Cattiva Modul. del Server e Impatto sul Client

```
public class Server {
 public static int alfa;
 /** Ritorna true se il valore di
    alfa e' contenuto nella lista x */
 public static bool cerca(Lista x) {
    do {
      if (x.info == alfa) {
        return true;
      }
      x = x.next;
    } while (x != null);
    return false;
 }
}
```

- basso interfacciamento esplicito: non è chiaro che alfa è il valore cercato
- alto accoppiamento: server impone al client di definire e usare alfa in modo coerente
- bassa coesione: server non effettua controllo di

```
public class Client {
  static Lista MiaLista() { ... }
  public static void main(String[] args) {
    Lista s = MiaLista();
    MioIO.print("Inserisci intero: ");
    /* ALTO ACCOPPIAMENTO: il client
       deve definire alfa come vuole il
       server prima di invocare cerca() */
    Server.alfa = MioIO.leggiInt();
    if (s != null) {
      /* ALTO ACCOPPIAMENTO: il client
         deve controllare che la lista non
         sia vuota */
      if (Server.cerca(s)) {
        /* ALTO ACCOPPIAMENTO: il client
           deve usare alfa come vuole il server */
        MioIO.print(Server.alfa+" presente\n");
        return;
      }
    }
    /* lista vuota o alfa non presente */
```

Esempio: Buona Modul. del Server e Impatto sul Client

```
public class Server {
 /** Ritorna true se il valore di
    alfa e' contenuto nella lista x */
 public static bool
           cerca(Lista x, int alfa) {
   while (x != null) {
     if (x.info == alfa) return true;
     else x = x.next;
   }
   return false;
 }
}
```

- alto interfacciamento esplicito: è esplicito che alfa è il valore cercato
- basso accoppiamento: server offre al client un servizio atomico
- alta coesione: server effettua controllo di lista vuota.

```
public class Client {
  static Lista MiaLista() { ... }
  public static void main(String[] args) {
    Lista s = MiaLista();
    MioIO.print("Inserisci intero: ");
    int alfa = MioIO.leggiInt();
    /* BASSO ACCOPPIAMENTO: il client
       si concentra sulla richiesta del
       servizio */
     if (Server.cerca(s, alfa)) {
       MioIO.print(Server.alfa +
          " presente\n");
     } else {
       MioIO.print(Server.alfa +
          " non presente\n");
     }
  }
}
```

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!