

Ingegneria del software

Progettazione:

La progettazione indirizza la realizzazione e la codifica.

Si tratta di una fase molto importante che porta a:

- definire la struttura che avrà il sistema
- determinare la qualità del software

Alla fine, si realizza un documento con tutti i requisiti, che possono essere sia hardware che software.

Cos'è la qualità del software?

È un insieme di **caratteristiche del software** che ne influenzano l'uso e la manutenzione, possono essere divise in due grandi gruppi:

- **Esterne:** possono essere apprezzate da un utente del software.
- **Interne:** sono rilevabili da un esperto di software, non sono visibili agli utenti, ma influenzano le caratteristiche esterne.

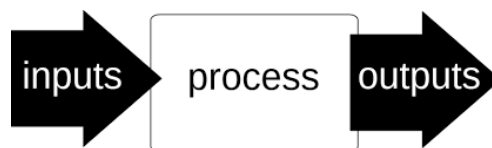
Il costo prevalente di un prodotto software si registra durante la fase di utilizzo:

- Un software di qualità è meno costoso da mantenere.
- Nella produzione industriale molti parametri formano la qualità, non tutti sono contemporaneamente ottimizzabili, spesso possono essere in conflitto tra loro!

Parametri di qualità del software:

Correttezza: capacità del software di eseguire correttamente i suoi compiti (secondo le relative specifiche).

Un prodotto software può essere visto come una funzione computabile: è corretto se per ogni input viene fornita la risposta appropriata. Il prodotto è la realizzazione della funzione che l'utente vuole vedere computata:



Efficienza: capacità del software di utilizzare in modo economico e razionale le risorse di calcolo (nello spazio e nel tempo). Ad esempio:

Un buon prodotto software dovrebbe:

- Utilizzare poco spazio di memoria centrale e di memoria di massa.
- Non intasare i canali di comunicazione.
- Svolgere le sue funzionalità in breve tempo

Essendo il costo dell'hardware (memorie e processori) decrescente, si sono introdotti ulteriori parametri di qualità.

Robustezza: capacità del software di funzionare anche in condizioni anomale.

La funzione computabile del processo f : $PRE(i) \rightarrow POST(f(i))$, dove $PRE(i)$ è il dominio di f , la correttezza di base solo sugli input che soddisfano le precondizioni di f .

Un software si **definisce robusto** se non succede nulla di male quando un input non rispetta le **precondizioni**.

- La robustezza si raggiunge definendo un comportamento per la funzione quando l'input non è corretto!

– **Estensione del dominio:**

si definisce un valore per f quando l'input non è corretto (non sempre è possibile, ad es. se la funzione è la radice di x , la precondizione è $x \geq 0$, se viene fornito un x negativo si potrebbe restituire un numero complesso, e se viene fornito un carattere?)

– **Gestione delle eccezioni:**

nella forma più semplice viene emesso un messaggio di errore e viene richiesto un nuovo input

Assicurare la robustezza di un sistema è

estremamente costoso, infatti qualunque input fornisca l'utente il sistema deve essere in grado di rispondere senza bloccarsi!

Affidabilità: capacità del software di presentare rari guasti.

- Si tratta della probabilità che non succeda qualche guaio non dipendente da input scorretto (robustezza).
- Riguarda il sistema hardware e software, è misurabile attraverso statistiche.
- Unità di misura dell'affidabilità (ad esempio): tempo medio tra due guasti, frequenza media dei guasti.
- Il grado di affidabilità richiesto da un sistema dipende dal sistema stesso: il sistema di gestione di un aeroplano deve avere altissima affidabilità, mentre questo è un parametro non essenziale per un sistema di gestione di una biblioteca.
- Il grado di affidabilità richiesto al sistema è un parametro che di solito entra a far parte del **contratto**: è una specifica non funzionale.
- I sistemi di gestione delle **centraline telefoniche** e degli aerei di linea sono sistemi ai quali deve essere garantita un'elevatissima affidabilità, che viene raggiunta normalmente duplicando o triplicando il sistema, facendo eseguire ogni operazione in parallelo e facendo controlli sulla consistenza dei risultati.
Se la probabilità di errore in un sistema è di 0,001 nel sistema duplicato scende a 0,000001.

Usabilità: il software presenta facilità di apprendimento e di utilizzo per l'utente.

Si tratta di un **parametro molto soggettivo** rispetto ai precedenti: è infatti legato a fattori umani. Un sistema è usabile se l'utente si trova a proprio agio con esso, fa pochi errori, lo trova gradevole, user-friendly. L'usabilità si può quantificare in qualche modo, ad esempio:

"il sistema è usabile se dopo un addestramento di 4 ore, l'utente medio non fa più di n (piccolo) errori al giorno".

In questo caso l'usabilità può rientrare tra i parametri presenti nel contratto: l'usabilità è il fattore che determina il successo di un sistema (si pensi ad esempio al DOS e a Windows).

L'usabilità costa e va calibrata sulla specializzazione dell'utenza:

Un editor o un data-entry devono essere usabili.

Un sistema per calcolare equazioni differenziali lo può essere molto meno essendo indirizzato ad un'utenza ristretta e specialistica, ad esempio MatLab.

Sicurezza capacità del software di non consentire utilizzi non autorizzati.

Costo: il software presenta un adeguato costo (tempi di sviluppo, risorse) rispetto ai benefici.

Estendibilità: il software presenta facilità ad essere adattato a cambiamenti (nelle specifiche).

Riusabilità: possibilità di utilizzare il software (in tutto o in parte) per diverse applicazioni.

Strutturazione: grado di organizzazione interna in parti con funzioni specifiche ed interagenti.

Leggibilità: capacità del programma di presentare esplicitamente le scelte fatte dal progettista.

Manutenibilità: il software presenta facilità di gestione dei problemi durante la fase di utilizzo.

Il grado di manutenibilità di un sistema è funzione **della facilità di gestione dei problemi** durante la fase di utilizzo.

Anche la manutenibilità può essere considerata nel contratto, dipende dall'applicazione e dal numero di volte che verrà usata l'applicazione.

La manutenibilità è strettamente collegata **al costo di manutenzione** e riguarda essenzialmente due aspetti:

1. **Modificabilità:** il software presenta facilità di modifica a seguito della scoperta di un errore o a causa di una variazione delle necessità applicative.

Indica la difficoltà di modificare il sistema a seguito della scoperta di un errore o a causa di una variazione delle necessità applicative e dipende da:

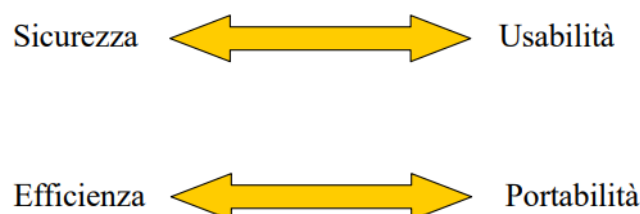
- modularità del sistema.
- leggibilità e comprensibilità del codice (codice auto-documentante).
- documentazione che accompagna il sistema.
- Portabilità riguarda la necessità di **portare il sistema su piattaforma hardware e/o software diverse da quelle considerate in partenza**, operazione che deve poter essere fatta con un costo limitato.

2. **Portabilità:** il software presenta facilità nel trasferimento in ambienti hardware e software di base diversi.

Compatibilità: facilità di combinare fra loro differenti prodotti software (necessità di standard per protocolli, interfacce, strutture dati standard).

Compromessi nella qualità del software

Di solito non è mai possibile massimizzare contemporaneamente tutte le qualità del software, bisogna sempre scendere a qualche compromesso



Modularità: Come suddividere un sistema in moduli?

Un sistema va diviso logicamente in parti componenti, in sottocomponenti, in componenti elementari: **ogni componente è un modulo.**

Un modulo è un'unità che:

- ha un compito preciso
- offre un insieme di servizi agli altri moduli
- può utilizzare i servizi degli altri moduli

Una buona modularizzazione è essenziale per la qualità di un sistema di grandi dimensioni.

La bontà di una modularizzazione non è misurabile, la bontà di un modulo si può ricondurre a tre parametri:

- Dimensione
- Coesione
- Accoppiamento

Dimensione dei moduli:

Il sistema dovrebbe essere costituito da moduli “piccoli” e quindi facilmente comprensibili.

È una considerazione molto qualitativa:

Nella realtà è meglio suddividere in pochi moduli grandi ma ben tagliati, che in tanti piccoli moduli organizzati male.

È necessario definire una dimensione media ottimale per un modulo, questa dimensione però dipende dallo strumento di realizzazione, varia quindi per ogni linguaggio di programmazione e dalle strutture che questo mette a disposizione (routine, funzioni, procedure ecc.)

Modularità – Coesione interna

La coesione interna è il grado di **intercorrelazione dei componenti un modulo.**

Un modulo si può definire buono **quando ha la massima coesione interna**, cioè non contiene nessuna componente “esterna”.

La gerarchia dei gradi di coesione di un modulo:

Occasionale: non vi è alcun motivo per il quale i componenti del modulo sono stati messi assieme, sono privi di relazioni.

Temporale: i componenti del modulo sono un insieme di funzioni che vengono richiamate in sequenza secondo un ordine prefissato.

Per dati: le varie parti del modulo operano su dati comuni definiti all'interno del modulo.

Per accesso a dati astratti: un modulo contiene tutte e sole le funzioni che definiscono un tipo di dati astratto ed eventualmente una sua rappresentazione.

Funzionalità specifiche: un modulo contiene tutto e solo ciò che serve alla realizzazione delle funzioni.

Modularità – Accoppiamento

L'accoppiamento è il grado di interrelazione di un modulo con gli altri moduli del sistema.

- I moduli devono presentare il minimo accoppiamento.
- Se l'accoppiamento è alto modificare un modulo implica la modifica anche dei moduli che sono in relazione con questo!
- La gerarchia dei gradi di accoppiamento di un modulo:
 - **Per dati in comune:** vi sono dati in comune sui quali i moduli lavorano (variabili globali).
 - **Per controllo:** quando un modulo può eseguire una parte selezionata di un altro (ad esempio chiamando il modulo con un parametro che funge da switch tra le varie operazioni che realizza).
 - **Per parametri:** quando non ci sono variabili globali ma i moduli comunicano solo per mezzo di parametri.

Quando si ottiene una buona modularizzazione?

Una buona modularizzazione: **Ogni modulo deve avere la massima coesione ed il minimo accoppiamento.**

I principi da adottare per la modularità:

Alta coesione: omogeneità interna del modulo.

Basso accoppiamento: indipendenza del modulo da altri.

Interfacciamento esplicito: modalità di utilizzo chiare.

Information hiding: dettagli nascosti nel modulo.

In genere questi obiettivi **si ottengono definendo un modulo per ogni funzione** (oppure per ogni tipo di dato astratto) e **facendo comunicare i moduli tramite il solo passaggio di parametri**, con al più qualche variabile globale ben specificata.

Modularità – Accorgimenti da adottare

Unitarietà: un modulo corrisponde ad una unità concettuale, ne incorpora tutti gli aspetti.

Interfacce esplicite: la comunicazione tra due moduli è evidente dall'esame del loro testo.

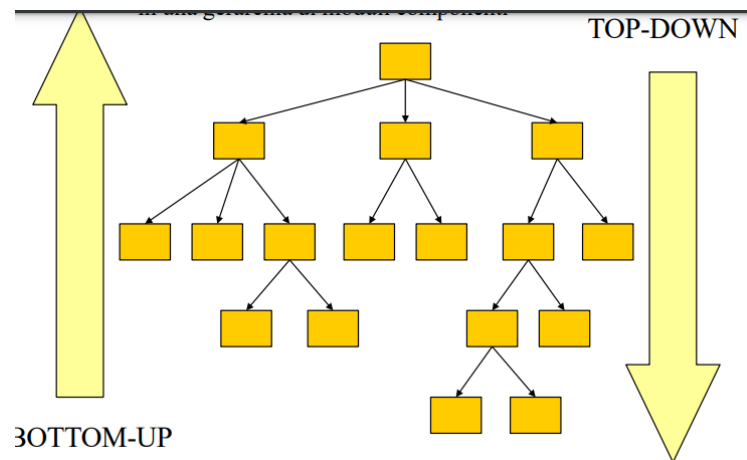
Poche interfacce: ogni modulo comunica con il minor numero possibile di altri moduli.

Interfacce piccole: due moduli comunicano scambiandosi il minor numero di informazioni possibili (minima Q di informazione).

Privatezza dell'informazione: le informazioni non soggette a scambio tra moduli devono essere gestite privatamente all'interno del modulo.

Progettazione – Direzione di sviluppo

Richiede la suddivisione del sistema da realizzare in una gerarchia di moduli componenti.



1) Strategia di sviluppo **Top-down**

La **strategia top-down** si basa sull'idea che la struttura del problema deve determinare la struttura del sistema (viene considerato come un'entità astratta priva di dettagli circa il modo nel quale è realizzata).

- Viene identificato **un insieme di componenti**, visti a loro volta come entità astratte che costituiscono il sistema.
- Ogni componente viene raffinata nelle sue parti costituenti.
- Il processo continua fino a quando non vengono **identificate le componenti elementari** (che potranno essere direttamente realizzate con gli strumenti scelti).
- **La gerarchia viene derivata partendo dalla radice ed andando verso le foglie, a ogni livello corrisponde un livello di astrazione del sistema.**

Vantaggi e svantaggi del Top-Down:

La decomposizione a partire dalle specifiche porta, in assenza di altri vincoli, ad una modularità molto buona: **la gerarchia è ad albero**.

Quando si arriva all'ultimo passo della decomposizione i moduli fondamentali sono completamente dipendenti dal progetto complessivo e quindi: – è raro che siano stati già realizzati in precedenza – l'approccio top-down rende difficile il riutilizzo del software.

2) Strategie di sviluppo **Bottom-up**

La strategia bottom-up opera nel senso inverso.

Vengono prima definiti i moduli di più basso livello, vengono poi combinati per ottenere i moduli di livello superiore. Così di seguito fino ad arrivare al nodo radice, cioè al modulo che definisce il sistema nel suo complesso.

Vantaggi e svantaggi del Bottom-Up:

Risulta semplice partire da componenti esistenti per costruire il sistema complessivo.

La gerarchia dei moduli è tipicamente a grafo.

La strategia bottom-up massimizza il riutilizzo del software ma solitamente non porta alla migliore modularizzazione ed efficienza.

Qual è la migliore strategia per la direzione di sviluppo?

Normalmente la definizione della gerarchia dei moduli avviene con un procedimento intermedio tra top-down e bottom-up.

Non si attua in un solo passaggio ma in modo ciclico:

- Inizialmente si segue una strategia prevalentemente **top-down** (che ben si adatta alla soluzione del problema).
- Si tiene in considerazione per il riutilizzo eventuale software già implementato o moduli per i quali risulta subito chiaro che devono realizzare determinate funzionalità.
- Si confrontano i moduli ottenuti con quelli esistenti: se si trovano moduli simili o già realizzati, si modifica il progetto per riutilizzarli.
- Il progetto viene visto e rivisto più volte.

Strategia di progetto orientata alle funzioni:

- Ortogonalmente alla direzione di sviluppo bisogna decidere cosa mettere nei moduli.
- Vi sono diverse strategie di progetto.
- **Nella strategia orientata alle funzioni un modulo racchiude una funzionalità:**
 - È un metodo di razionalizzare la progettazione spesso seguito istintivamente.

Il progetto viene scomposto in un insieme di unità interagenti, ognuna corrispondente ad una funzione chiaramente definita.

Questo approccio garantisce una coesione di tipo funzionale e, se la decomposizione è fatta bene, determina spesso un basso accoppiamento.

Gli step della strategia di progetto orientata alle funzioni:

- 1) **Analisi del flusso di dati:**
Viene schematizzato il **flusso di dati** all'interno del sistema determinando le **unità funzionali di trasformazione** dei dati.
- 2) **Progetto statico:**
Sulla base di quanto prodotto nel passo precedente **si individuano i moduli del sistema**, che vengono a loro volta suddivisi in sotto-moduli fino a quando si arriva alle **componenti elementari, determinando la gerarchia dei moduli**.
- 3) **Progetto dettagliato:** Viene definita la **struttura interna dei moduli**, vengono prese decisioni realizzative per le strutture dei dati da utilizzare ed eventualmente per il controllo. Si tratta di una fase immediatamente precedente alla codifica vera e propria, ma di livello più alto (significa ad un livello di astrazione più alto, ancora concettuale e non codificato totalmente in linguaggio di programmazione).
Ad esempio, potrebbe essere un **programma parziale** con **istruzioni del linguaggio intervallate da espressioni in linguaggio naturale** ed altro...

I punti di decisione:

Non sempre la strategia di **progettazione orientata alle funzioni** è la migliore, anzi può portare alla definizione di moduli fortemente accoppiati (ovvero dipendenti troppo gli uni dagli altri!).

Anche in presenza di un'ottima coesione si può presentare un **pessimo accoppiamento** a causa di **moduli che comunicano per mezzo di dati in comune**: **la minima modifica della rappresentazione di questi dati si ripercuote su tutti i moduli che devono essere quindi modificati!**

Per **risolvere il problema** dell'accoppiamento è necessario che le funzioni, cioè i moduli, **interagiscano con una visione astratta dei dati**: una soluzione è l'utilizzo di tipi di dati astratti per rappresentare le varie categorie di dati.

Il **criterio di Parnas** vede la progettazione come un insieme di punti di decisione

Fasi di progettazione e punti di decisione:

- La prima fase della progettazione deve individuare:
 - Le decisioni importanti (cioè, critiche), che influenzano pesantemente il funzionamento del progetto.
 - Le decisioni soggette a cambiamenti.
- Si associa poi ad ogni decisione importate un modulo.
- I moduli vengono quindi a coincidere con le decisioni: il modulo non rende nota al mondo esterno una decisione ma la funzionalità che realizza.

Information hiding:

Il principio dell'information hiding:

Un modulo offre all'esterno **un insieme di funzionalità** senza rivelare come sono realizzate.

- Ogni modulo deve nascondere una decisione ma deve fornire all'esterno certe funzionalità.
- Ogni modulo offre all'esterno un certo numero di funzioni di interfaccia per manipolare l'entità astratta nascosta nel modulo, senza però mai rivelare agli altri moduli la sua effettiva realizzazione.
- Ogni modulo ha delle strutture dati private non accessibili se non tramite le funzioni di interfaccia.
- Viene così risolto il problema dell'accoppiamento: i moduli comunicano solamente per mezzo di funzioni di interfaccia e per mezzo dei loro parametri

Strategia di progetto orientata agli oggetti:

La tecnica di progettazione basata sulle decisioni è stata individuata da Parnas nel 1972.

Queste idee sono state integrate nella progettazione orientata agli oggetti: che porta a moduli di altissima coesione e debolissimo accoppiamento.

L'idea consiste nel progettare un sistema come un insieme di oggetti interagenti:

- Ogni oggetto ha uno stato, cioè una rappresentazione interna nascosta agli altri oggetti (information hiding).
- Ha un insieme di funzionalità che gli altri oggetti possono utilizzare in modo controllato, per accedere e modificare informazioni contenute nell'oggetto (e cioè il suo stato).
- Un oggetto è quindi noto all'esterno come un insieme di funzionalità, le funzioni di interfaccia costituiscono il protocollo dell'oggetto

- I dettagli realizzativi di un oggetto non sono noti all'esterno (information hiding)
- Gli oggetti modellano i concetti del dominio, sono i moduli del software che viene costruito.
- Le funzioni sono preposte alla manipolazione degli oggetti ed alla loro comunicazione.
- Gli oggetti comunicano mediante message passing (cioè, logicamente).
- Sono entità attive e la chiamata di un modulo è vista come la richiesta ad un oggetto di eseguire una sua funzionalità.
- Il sistema risultante è altamente modificabile.
- L'obiettivo della progettazione orientata agli oggetti è di progettare il sistema in sottosistemi mutuamente indipendenti che comunicano per mezzo di interfacce astratte.

Incapsulamento: ogni modulo (classe) è una implementazione di un tipo astratto, contiene l'implementazione delle funzioni del tipo ed i dati necessari.

Information hiding: l'interfaccia della classe fornisce i servizi disponibili all'esterno (funzioni del tipo astratto), il resto rimane nascosto e non visibile alle classi interagenti.

Genericità: si possono far corrispondere ad uno o più domini dei tipi parametrici (template).

Ereditarietà: una classe può essere definita (ereditata) in termini di differenze da un'altra.

Polimorfismo: un identificatore può denotare funzioni comuni a classi diverse.

Binding dinamico: vengono invocati i metodi della classe alla quale l'oggetto appartiene.

Differenze tra approccio funzionale e ad oggetti:

Nell'approccio funzionale le decisioni sulla rappresentazione dei dati e la condivisione di informazioni sullo stato devono essere prese prima, questo tende ad aumentare l'accoppiamento del sistema e quindi a ridurre la modificabilità dello stesso.

- Entrambi i metodi portano ad una buona coesione.
- Il metodo funzionale porta a coesione intorno alle funzionalità.
- La progettazione ad oggetti porta a coesione attorno ad entità.
- In quali condizioni è opportuno usare la progettazione orientata alle funzioni ed in quali la progettazione orientata agli oggetti?

La migliore strategia?

Si possono vedere come due strategie complementari piuttosto che alternative, nella progettazione possiamo individuare fasi temporalmente consequenziali:

- Inizialmente il sistema da realizzare si presenta logicamente costruito da un insieme di sottosistemi che realizzano delle funzionalità. Ogni sottosistema può essere modellato come un oggetto al quale si associano le relative funzionalità:
 - È opportuno usare una strategia orientata agli oggetti.
- In un successivo momento si determina per **certe operazioni** un preciso flusso di dati che consente di determinare una sequenza di operazioni:
 - È opportuno allora usare una strategia funzionale.
- Si entra poi nel dettaglio della struttura dei moduli, vengono prese decisioni che è opportuno mantenere nascoste all'esterno dei moduli vengono presentate solamente le funzionalità richieste:

- è opportuno ritornare di nuovo ad una strategia orientata agli oggetti.

Il progetto viene visto e rivisto più volte (iterazioni).

La progettazione ad oggetti:

L'approccio ad oggetti non è adeguato a modellare situazioni prive di uno stato. È molto adeguato a modellare situazioni nelle quali vi sono oggetti fisici:

- corrispondenza tra oggetti fisici ed oggetti logici
- oggetti omogenei si potranno raggruppare in classi
- che si potranno poi specificare in sottoclassi

Un esempio:

Un sistema di gestione amministrativa di posizioni lavorative, costituito da un database dei dipendenti, da documenti elettronici riferiti ad ogni dipendente, alcuni con allegati, altri firmati ecc.

La gestione dei documenti consente di associare e trattare a seconda delle diverse esigenze i documenti riferiti ad un dipendente.

Tutti i documenti hanno la stessa struttura, cioè, rappresentano il loro stato allo stesso modo e rispondono agli stessi messaggi, cioè hanno lo stesso protocollo.

La classe definisce la rappresentazione dello stato e le operazioni di tutti i possibili documenti.

- ➔ Un singolo documento è una istanza della classe dei documenti, e differisce da un altro documento solamente per i dati che ne costituiscono lo stato.

Si possono modellare documenti particolari, ad esempio i documenti che forniscono lo stato fiscale del dipendente:

- ➔ Si tratta di un normale documento con in più la struttura dati e le operazioni per la gestione della posizione fiscale.
- ➔ Si definisce **una sottoclasse** della classe dei documenti

Nella sottoclasse si elencano solamente le proprietà (dello stato e delle operazioni):

- ➔ Che distinguono le sue istanze dalle istanze della classe.
- ➔ Le altre proprietà vengono ereditate automaticamente.

Lo stesso vale per esempio per i documenti firmati elettronicamente.

