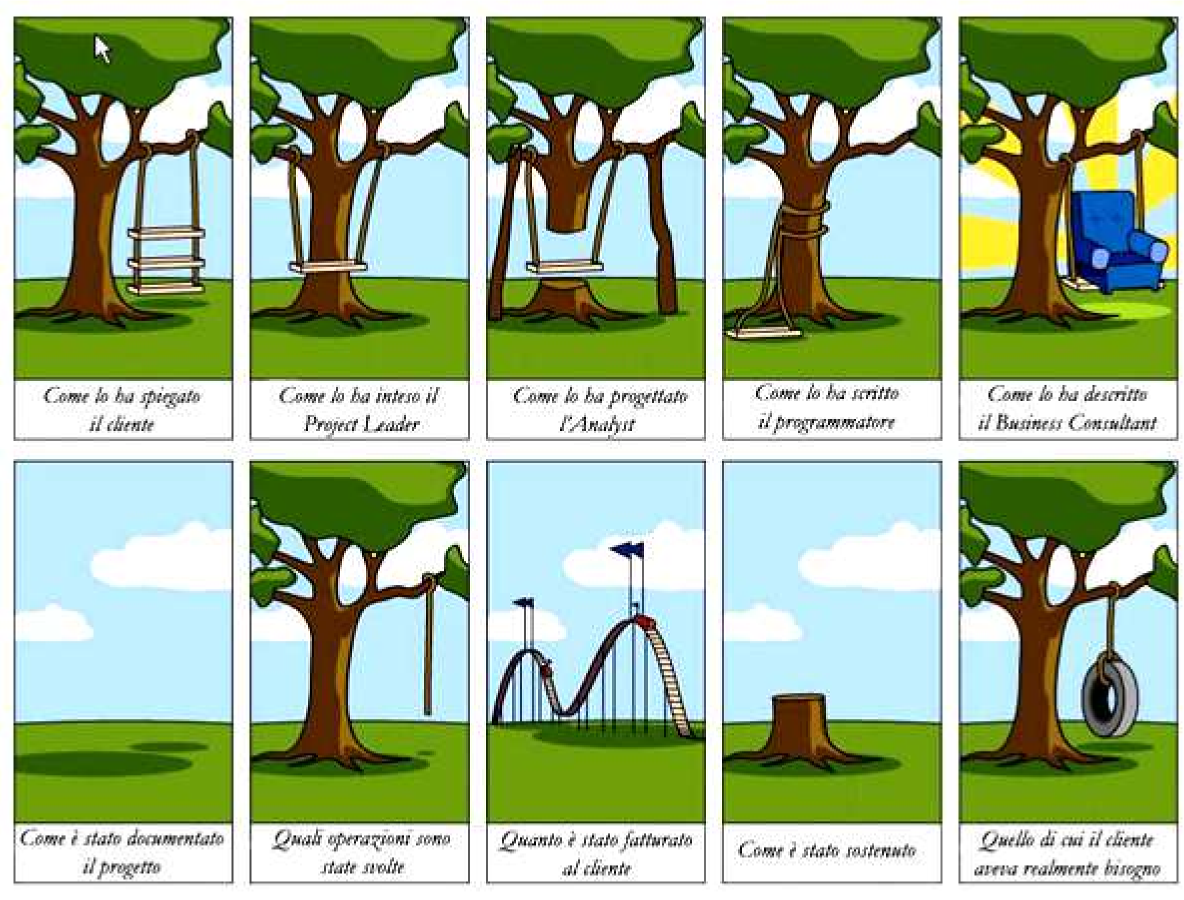
paolo macchi

Tecno2

L'ingegneria del software



ISIS Facchinetti 2015

Tecno 2 - l'ingegneria del software – ISIS Facchinetti 2014-15
Gli appunti contenuti in questa dispensa si basano essenzialmente sugli appunti del prof. Moreno Marzolla a cui sono totalmente debitore: http://www.moreno.marzolla.name/teaching/IngSoftware2005/index-unife.php
Alcune immagini sono tratte da: http://www.youtube.com/playlist?list=PLgqAc4E1vc2vxNkgLK6EZXCkBbsaAvwvC
Approfondimenti si possono trovare anche in :
http://studenti.di3.units.it/Sistemi%20Informativi%20I/Slide%20Progettazione%20%28e%29.pdf)
http://www.federica.unina.it/ingegneria/ingegneria-del-software-ingegneria/introduzione-ingegneria-software-2/
http://home.deib.polimi.it/morzenti/IngSw/cicliDiVita.pdf

Questo testo è pubblicato sotto licenza Creative Commons - Attribuzione - Non commerciale - Condividi allo stesso modo 3.0 Unported - Per le condizioni consulta: http://creativecommons.org/licenses/by-nc-sa/3.0/deed.it . Le utilizzazioni consentite dalla legge sul diritto d'autore e gli altri diritti non sono in alcun modo limitati da quanto sopra.

Il documento è scaricabile da www.isisfacchinetti.it , sezione download , per fini esclusivamente didattici e non commerciali

Segnalazioni di errori, critiche e consigli sono molto graditi e possono essere inoltrati a paolo.macchi@libero.it , oppure lasciando un commento al momento del download per gli studenti registrati.

Tecno2 - L'ingegneria del software – paolo macchi - ISIS Facchinetti 151005

ISIS "Cipriano Facchinetti" via Azimonti, 5 - 21053 Castellanza (VA) - http://www.isisfacchinetti.it/
Tel. 0331635718 fax 0331679586 info@isisfacchinetti.it

Convenzioni usate nel testo:

rappresenta una curiosità, un approfondimento

NOTA rappresenta una nota

rappresenta una esercitazione o esempio

http://qrcode.kaywa.com/ link di riferimento

Rappresenta un codice o dei risultati o una segnalazione storica

Sommario

Perché siamo qui? 4
Qualche esempio reale 5
Costi del Software........ 6
Cos'è il Software?........ 6
Le caratteristiche del software........ 7
Caratteristiche di un prodotto Software........ 7
I diversi tipi di software 7
Che cos'è l'ingegneria del software? 8
Perchè l'ingegneria del software è importante? 8
Costi nel processo di produzione del Software 9

Il processo di produzione del Software........ 10
Intro 10
Modello dei principi dell'ingegneria del Software 11
Principi fondamentali: il cuore della disciplina........ 11
Il processo di produzione (sviluppo) del software........ 14
Problemi nel processo di sviluppo software 14
Il ciclo di vita del software 14
Perché avere un modello? 14
Modelli di processo 14
Modello a cascata (waterfall) 15
Considerazioni sul modello a cascata........ 17
La Prototipazione e lo Sviluppo Evolutivo 17
Sviluppo incrementale (iterativo) 18
Processo agile (usato in open source)........ 21

Analisi e specifica dei requisiti (cosa) 23
Requisiti e specifiche 23
Cosa sono i requisiti? 23
Cosa sono le specifiche? 24
Stili di specifica........ 27
Linguaggi di specifica 28

La fase di progettazione (come) 30
Metodologie per la progettazione di sistemi........ 31
Qualità di un progetto........ 31
La Specifica del Progetto 31
Le fasi di progettazione 31
Progettazione top-down e bottom-up 32
La Programmazione top-down /bottom-up........ 36
Indipendenza Modulare........ 37
Il principio dell'information hiding........ 37
Qual è la migliore strategia per lo sviluppo?........ 38
Strategia di progetto orientata alle funzioni........ 38
EX Un esempio di programmazione Top Down (funzionale) 39
Strategia di progetto orientata agli oggetti 41
Differenze e analogie tra approccio funzionale e ad oggetti........ 42

L'interfaccia utente 43
Regole d'oro dell'Interfaccia Utente 43
Processo di Progettazione dell'Interfaccia 44
NOTA........ 44

"L'interazione uomo-macchina: linee guida e standard........ 45
Valutazione del Progetto 46

Soluzioni per la progettazione........ 48
UML 48

Tipi di diagrammi UML *48*
Diagrammi relativi agli aspetti statici di un sistema 49
Diagrammi relativi agli aspetti dinamici di un sistema 51

Collaudo del software 55
Strategie Problemi e Limitazioni *56*

- **Esercitazioni 58**

Fare progetto *58*
Obiettivi 58
Come? 58
I Leader 58
Il team 59
RICORDA *59*
Cos'è un progetto software? 59
Tipi principali di progetto: 59
Progetti software 59
- *Progettare con gli EAS* *60*
- *DUPLone Assembler* *60*

Esempi di progetti di ordine generale *61*
- Progettare una mensola porta-oggetti 61
- Progettare la cuccia del vostro cane 61
- Progettare la potatura di una siepe 61
- Progettare un giardino 61

Appendice 62
Classificazione dei sistemi *62*
Automi a stati finiti *63*
Esercizi 70

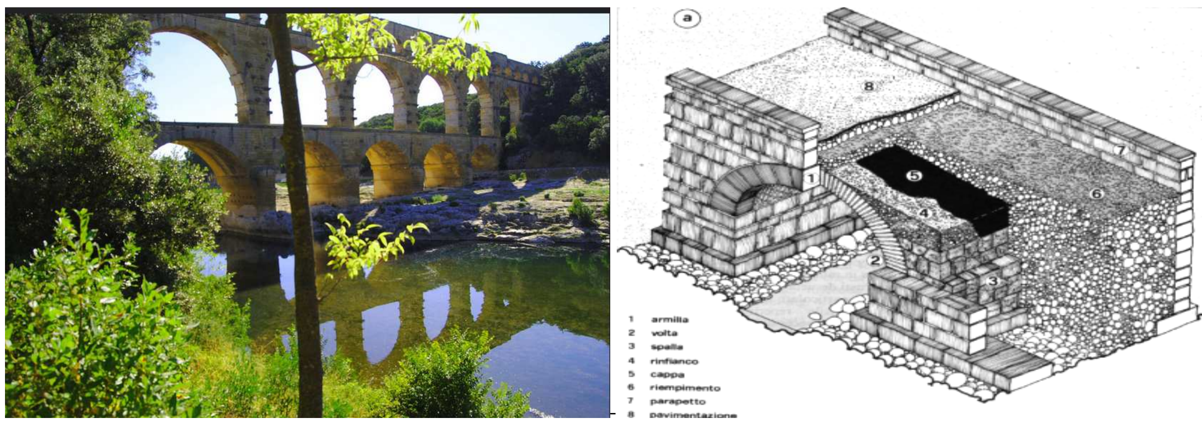
Perché siamo qui?

"Prof, le assicuro che a casa andava…"
-Tipica espressione dello studente quando si prova un progetto in laboratorio -

Quando a uno studente si propone un progetto di laboratorio (un programma, un'applicazione software), la prima cosa che deve fare è capire in cosa consiste il PROBLEMA posto. Capire, cioè la questione posta partendo da elementi noti con un ragionamento. Dall'analisi del problema si arriverà, successivamente, alla PROGRAMMAZIONE che è l'attività che, a partire dal problema, conduce alla creazione del programma che verrà eseguito da un calcolatore.
Purtroppo, spesso, si preferisce partire dal codice ,così "non si perde tempo".
In realtà è vero ciò che afferma la legge di Mayers:
" E' bene trascurare la fase di analisi e progetto, e passare immediatamente all'implementazione così da gudaganre il tempo necessario per rimediare agli errori commessi per aver trascurato la fase di analisi e progetto".
Capire un problema però è, in molti casi, molto complesso perché si tratta di affrontare elementi nuovi e soprattutto perché ci si deve calare nella mentalità del "clinte", cioè di colui al quale deve essere fornito il prodotto.

Ma come avere conoscenza del problema?
Prima di tutto **Cosa** si deve fare?
Analizzare la situazione reale in tutte le sue sfaccettature: ascoltare il cliente, studiare le problematiche aperte, acquisire comptenza sulle tematiche in gioco. Infine **creare un modello** che rappresenti questa realtà.

In secondo luogo (ma solo dopo!) **Come** si deve fare?
Progettare una soluzione usando tecniche, metodologie e strumenti adeguati
Un aiuto ci viene dall'Ingegneria del Software" che cerca di mettere ordine in una materia che è ancora "giovane" (si pensi che già i Romani, più di 2000 anni fa, erano esperti nella costruzione di Ponti, mentre il software è una materia relativamente giovane che risale a50,60 anni fa. Il termine fu coniato durante la seconda guerra mondiale (si veda: http://it.wikipedia.org/wiki/Software#Storia_del_software))



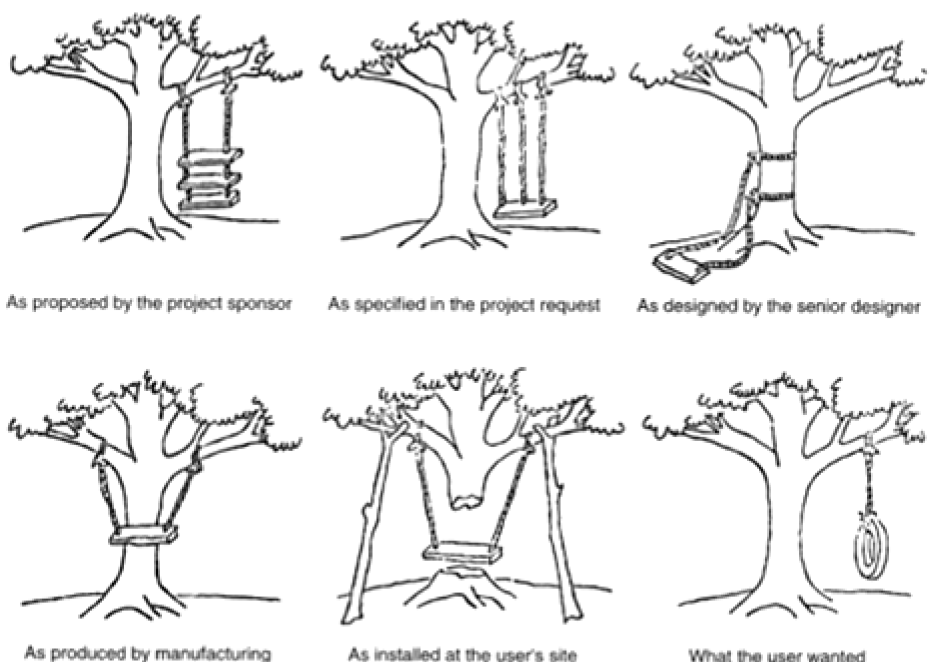
pont du Gard – ponte romano nel sud della Francia https://www.flickr.com/photos/22996675@N07/19631190694/in/dateposted-public/

Marco Polo descrive un ponte, pietra per pietra.
– Ma qual è la pietra che sostiene il ponte? – chiede Kublai Kan.
– Il ponte non è sostenuto da questa o quella pietra, – risponde Marco, – ma dalla linea dell'arco che esse formano.
Kublai Kan rimane silenzioso, riflettendo. Poi soggiunge:
– Perché mi parli delle pietre? È solo dell'arco che m'importa.
Polo risponde: – Senza pietre non c'è arco. [Le città invisibili, Italo Calvino]

Ecco perché non basta la "buona volontà", come spesso si dimostra osservando i risultati di progetti di laboratorio, ma

occorre una disciplina che sia in grado di far emergere con chiarezza i passi da fare per arrivare a un software che, in ultima analisi, sia utile al cliente, senza dimenticare che "il cliente sei proprio TU!" e come tale vuoi un prodotto che soddisfi le tue esigenze, ma che , anche, sia facile da usare, immediato, affidabile, aggiornabile, del giusto prezzo e così via…

"Utente - Di lui si dice spesso che non sa quello che vuole. Sarà anche vero, ma una cosa è certa: l'utente sa benissimo quello che non vuole."
-Il Dizionarietto del Diavolo del DP -



Qualche esempio reale

"Se i costruttori costruissero come i programmatori programmano, il primo picchio che passa potrebbe distruggere la civiltà."
- Weinberg (Legge di)-

Il 4 giugno 1996 il razzo **Ariane 5** è esploso in volo dopo 40 secondi dal decollo
Il razzo trasportava un cluster di satelliti del valore di 500M$ (di allora)
Il costo totale di sviluppo del razzo era circa 8 Miliardi di $
L'esame dei dati ha indicato in un malfunzionamento software la causa del disastro
http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html

Il **Therac-25 / 1** Era un dispositivo computerizzato per la radioterapia dei pazienti affetti da cancro
Tra giugno 1985 e gennaio 1987 sei pazienti sono stati uccisi o feriti seriamente da dosi eccessive di radiazioni
Rsponsabile era il software di controllo, scritto da un unico programmatore:
La sincronizzazione di diversi task era realizzata in modo dilettantesco e improvvisato, e ciò ha causato i problemi
Non erano stati effettuati test formali o verifiche, non esisteva alcuna documentazione del software
I problemi esistevano nel precedente modello Therac-20, ma moduli di controllo hardware bloccavano la macchina evitando la somministrazione di dosi eccessive I controlli hardware erano stati eliminati e sostituiti da controlli software, probabilmente per contenere i costi
http://courses.cs.vt.edu/cs3604/lib/Therac_25/Therac_1.html
http://it.wikipedia.org/wiki/Therac-25

Esercizio
Trovare 3 esempi di errori software che hanno causato incidenti o problemi

Costi del Software

Le economie di tutti i paesi sviluppati dipendono dal software, e la maggior parte dei sistemi sono controllati da software. Sempre più sistemi sono controllati dal software.
Gli investimenti per il software rappresentano una parte significativa del PIL di tutte le nazioni industrializzate.
Il software costa più dell'hardware.
Il mantenimento di un software complesso costa più dello sviluppo dello stesso, specialmente per sistemi con lunga vita.
É più costoso manutenere il software piuttosto che svilupparlo, soprattutto per sistemi di vecchia data (i cosiddetti sistemi legacy).

L'Ingegneria del software ha come obiettivo riuscire a sviluppare software in maniera efficace e con costi contenuti (cost-effective).

Cos'è il Software?

Non solo programmi, ma l'insieme degli 'artifatti' che lo compongono, prodotti durante il suo sviluppo e la documentazione associata: manuale di configurazione, manuale utente...

Software:

- Generici. Sviluppati per essere venduti ad una vasta gamma di utenti. Contribuiscono alla maggior spesa di software
- Personalizzati. Sviluppati per un utente specifico in base alle sue esigenze. Richiedono il maggiore sforzo per lo sviluppo

Un sistema software, essendo rivolto ad altri utenti, dovrà essere usabile, portabile, affidabile, etc…
La definizione IEEE (Institute of Electrical and Electronic Engineers)
"insieme di programmi, procedure, regole, e ogni altra documentazione relativa al funzionamento di un sistema di elaborazione dati"

Il processo software:

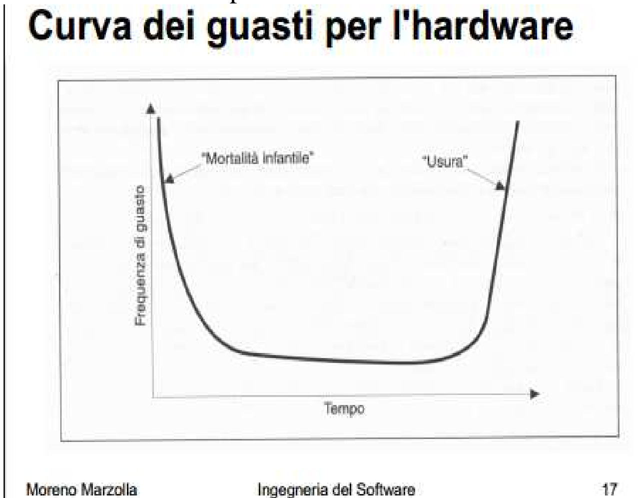
- Problemi della produzione del software.
- Cicli di vita del software.

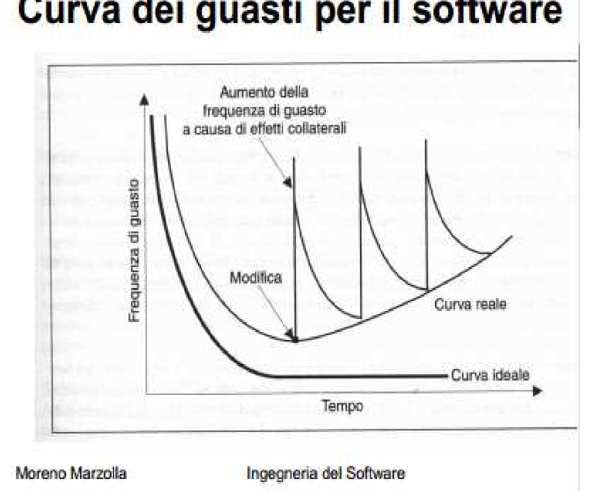
Analisi e progettazione:

- Aspetti generali dell'analisi e della progettazione.
- Analisi e progettazione orientata agli oggetti.
- UML come linguaggio di analisi e progettazione. Pattern architetturali e di progettazione.
- Progettazione dell'interfaccia.
- Documentazione.
- Testing, validazione e debugging: Obiettivi e pianificazione del testing. Progettazione e valutazione dei casi di test. Debugging.
- Manutenzione: tipi e processi di manutenzione software. Gestione della configurazione.

Le caratteristiche del software

Il software si sviluppa o si struttura, non si fabbrica nel senso tradizionale.
Lo sviluppo del software e la progettazione dell'hardware sono attività profondamente diverse.
Il software non si “consuma” come avviene per l’hardware
Mentre l'industria si dirige sempre più verso un assemblaggio a componenti, la maggior parte del software viene realizzato in modo specifico





Caratteristiche di un prodotto Software

- **Mantenibilità** Evolvere in rapporto alla modifica di requisiti
- **Affidabilità** Ci si deve poter fidare del prodotto Software; Correttezza, Robustezza, Verificabilità, Sicurezza – Innoquità
- **Efficienza** Non deve sprecare risorse (memoria, tempo,...)
- **Usabilità** Deve avere interfaccia e documentazione appropriate

I diversi tipi di software

- Software di sistema Collezione di programmi al servizio di altri programmi (compilatori, editor, strumenti per la gestione di file...)
- Software real-time E' il software che sorveglia, analizza, controlla eventi esterni mentre avvengono
- Software gestionale Elaborazione di dati aziendali
- Software scientifico e per l'ingegneria Algoritmi di calcolo intensivo (astronomia, vulcanologia, biologia molecolare, terremoti...)
- Software embedded Risiede generalmente in memorie per sola lettura e ha lo scopo di controllare prodotti e sistemi di consumo o Iindustriali
- Software per i personal computer Elaborazione testi, fogli elettronici, grafica, programmi multimediali...
- Software basato sul Web CGI, PHP, JSP...
- Software per l'intelligenza artificiale Algoritmi non numerici (euristici) per la risoluzione di problemi complessi

Esercizio
Elencare 2 applicazioni software per ogni tipologia

Che cos'è l'ingegneria del software?

"L'Ingegneria del Software è un insieme di teorie, metodi e strumenti per sviluppare software di qualità in maniera professionale"

L'ingegneria del software è una disciplina ingegneristica che si occupa di tutti gli aspetti relativi alla produzione del software e propone un approccio sistematico e organizzato per il loro lavoro, usando strumenti e tecniche appropriate.

Perchè l'ingegneria del software è importante?

"Io sviluppo, Tu installi, Egli prega." *-Anonimo -*

Il Software è :

- intangibile
- Difficile comprenderne la complessità, la qualità, lo sforzo necessario per lo sviluppo
- Il Software è facile da riprodurre
- I Costi maggiori sono nel processo di progettazione e sviluppo, diversamente da altri prodotti industriali
- L'industria del software richiede un grosso impegno intellettivo
- É difficile da automatizzare

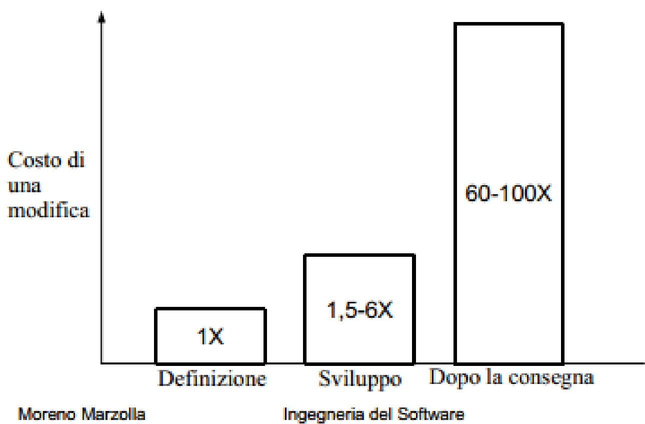
L'ingegneria del software :

- punta a produrre software affidabile, sicuro, usabile e manutenibile.
- diversamente dalla programmazione, non si preoccupa solo della funzionalità o di particolari caratteristiche del sistema.
- è particolarmente importante per sistemi da cui dipendono **persone e processi** che vengono usati per molti anni.

software vs Ingegneria del software	
L'informatica si occupa delle **teorie e dei metodi** alla base dei sistemi software ed informatici;	l'ingegneria del software si occupa degli aspetti pratici relativi alla produzione del software.

Anche se le teorie ed i principi informatici sono fondamentali, spesso vengono trascurati dagli ingegneri del software per questioni di praticità (diversamente da quanto accade in altre branche ingegneristiche).

L'effetto delle modifiche



I miti del …

management	cliente	programmatore
• Abbiamo standard e procedure da seguire nello sviluppo. Non ci serve altro” • Abbiamo i più moderni sistemi di sviluppo. • Acquistiamo i computer sempre più recenti” • Sviluppare non è una attività facilmente automatizzabile. • Se siamo in ritardo, possiamo recuperare aumentando il numero di programmatori “Adding manpower to a late project makes it later” - Fred Brooks “se una donna partorisce in 9 mesi, due donne lo fanno in 4 mesi e mezzo” - paradosso della partoriente -	• “Un'affermazione generica di cosa deve fare un programma è sufficiente per iniziare a scrivere codice” • I requisiti mutano di continuo, ma i cambiamenti si gestiscono facilmente grazie alla flessibilità del software”	• Il solo prodotto di un progetto concluso è il programma funzionante” • L'ingegneria del software ci farà scrivere un'inutile e voluminosa documentazione che inevitabilmente rallenterà la cose”

Costi nel processo di produzione del Software

Circa il 60% dei costi è legato allo sviluppo, il 40% sono costi per la verifica e validazione (testing).
I sistemi software sono intangibili pertanto è necessario documentare e tenere traccia di ciò che si sta facendo.
Ogni fase del processo di produzione deve sfornare qualche documento
Tali documenti rendono visibile il processo di produzione del software.
I manager si basano sui documenti per prendere le decisioni
I documenti potrebbero non essere pronti quando richiesti, perché i tempi di sviluppo possono non coincidere con quelli in cui si devono prendere le decisioni.
La necessità di approvare documenti rallenta il processo di sviluppo.
Il tempo necessario per revisionare ed approvare i documenti può essere significativo

Responsabilità (Etica) professionale
Non limitarsi agli aspetti tecnici, ma guardare anche ai risvolti etici, sociali e alle responsabilità professionali: essere onesti non è solo rispettare le leggi:
- Confidenzialità
- Competenza
- Diritti di proprietà intellettuale
- Uso inappropriato dei computer

(ACM/IEEE, Code of Ethics” http://www.acm.org/about/se-code)

Il processo di produzione del Software

EX I tools per la produzione del software: …..

Intro

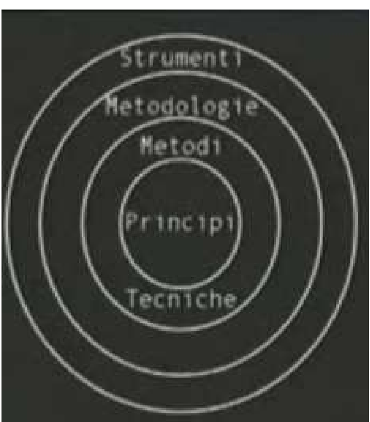
"Un **processo software** – o processo per lo sviluppo del software – è un insieme strutturato di attività che porta alla creazione di un prodotto software.
Un processo definisce chi fa che cosa, quando e come per raggiungere un certo obiettivo.
Esistono numerosi processi software, organizzati attorno ad attività comuni. Il loro scopo è soddisfare le aspettative dei clienti, fornendo prodotti di qualità, nei tempi e nel budget previsto, rendendo i prodotti remunerativi e i processi affidabili, prevedibili ed efficienti.

Tutti i processi software sono basati su un certo numero di attività fondamentali comuni: allora, perché ci sono tanti approcci diversi che descrivono il processo? In che cosa si differenziano tra loro? Processi diversi fanno riferimento a scelte differenti per alcune funzioni primarie del processo: come viene determinato l'ordine delle attività? Per quanto tempo continueremo a fare questa cosa? Che cosa faremo dopo?

Per questo ci serviremo di "modelli". Un modello di processo software è una rappresentazione astratta di un processo software. Ogni modello rappresenta un processo da un particolare punto di vista fornendo però solo informazioni parziali a riguardo."

(http://www.dia.uniroma3.it/~cabibbo/asw/pdf/asw210-processi-software.pdf)

Modello dei principi dell'ingegneria del Software

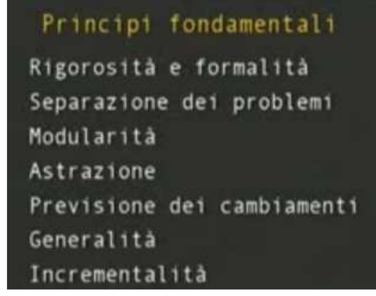


http://www.youtube.com/playlist?list=PLgqAc4E1vc2vxNkgLK6EZXCkBbsaAvwvC

I campi dell'ingegneria del software:

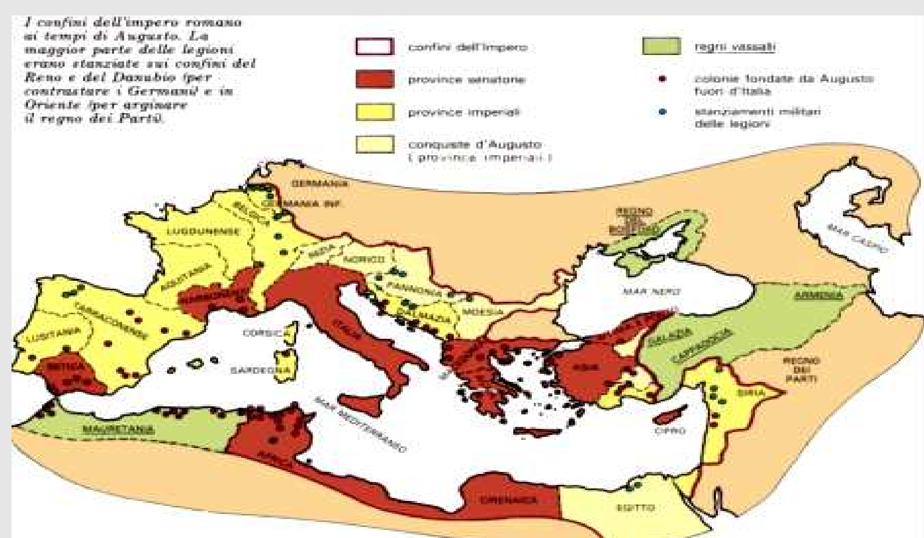
- **Principi**: il cuore della disciplina , sono *le linee guida* , astrarre dai dettagli. Dicono il MODO con cui va fatta una certa cosa, però NON dicono come.
- **Metodi e Tecniche**: implementano i principi, applicazione dei principi al problema, strumenti per automatizzare il processo. Soluzioni basate sui principi: COME va fatta una certa cosa.
- **Metodologie**: *come integrare i principi e le tecniche*; quali tecniche usare in un certo contesto, quando applicarle e come usare i metodi. Come mettere insieme le tecniche.
- **Strumenti** sono i tools che applichiamo alle metodologie

Principi fondamentali: il cuore della disciplina



- **Rigorosità e formalità**
  - *L'uomo è sempre l'elemento essenziale* ma si migliora, si amplifica con dei complementi alla **CREATIVITA'** umana, ad esempio con il *rigore e* la *formalità*: trovare la soluzione espressa in modo formale **per comprendere i risultati e condividerli** (ad es. la matematica: prova formale di correttezza di un teorema)
  - Permette di rendere la creatività condivisibile e capibile.
  - Ad esempio la descrizione formale e la **prova formale di un protocollo di comunicazione**.
  - Ad esempio **documentazione precisa**, con forme definite per semplificare la comunicazione: capitoli, regole, rigore. Esiste anche una documentazione informale o formale, cioè forme definite e precise per semplificare la documentazione: ci sono moduli, capitoli, regole specifiche
- **La separazione dei problemi**
  - è fondamenale perché oltre un certo livello di complessità non siamo più in grado di vedere TUTTI i dettagli (ad es. Sistema Operativo)

E’ importante capire che noi, esser umani, NON SIAMO IN GRADO DI PRENDERE IN CONSIDERAZIONE TUTTI I DETTAGLI: è un limite intrinseco alla mente umana.(ad es. Ssitema Operativo)
Come si fa allora?   Sottoproblemi , **“Divide et Impera”.**
(Ai tempi dei romani tale strategia era un mezzo adoperato per governare il territorio, evitando che le singole popolazioni si coalizzassero in rivolte e sommosse contro Roma. Un tipico esempio fu l'invasione della Macedonia e la sconfitta del re Perseo di Macedonia nella battaglia di Pidna (168 a.C.), dopo la quale la Macedonia fu divisa in quattro repubblichedipendenti da Roma, cui furono pesantemente limitati i rapporti reciproci nonché quelli con gli altri stati ellenici - Wikipedia).



http://www.romanoimpero.com/2009/06/augusto-27-ac-14.html

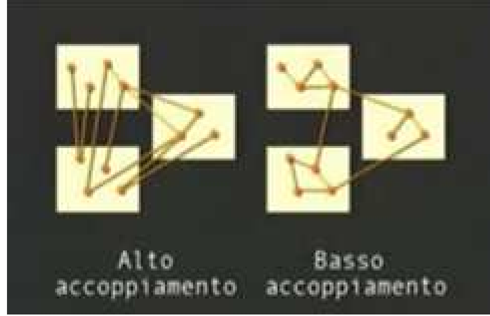
Ad es. Chi conosce tutti i dettagli di un’automobile? NESSUNO!!! Motoristi, meccanici,… divido la macchina in…
Ad es. il software lo divido in basi dati, funzioni, presentazione.

- **Modularità**
  Identificare moduli che possono essere riusati

Modularità = Dividere il sistema in moduli che hanno alta coesione e basso accoppiamento (pochi link (ricorda: legami forti e deboli!!)). Riusati e integrati. Divisione Processo in fasi, fasi in attività, attività in sottoattività…

Es il test in varie fasi per collaudare il sistema per gestire la complessità.

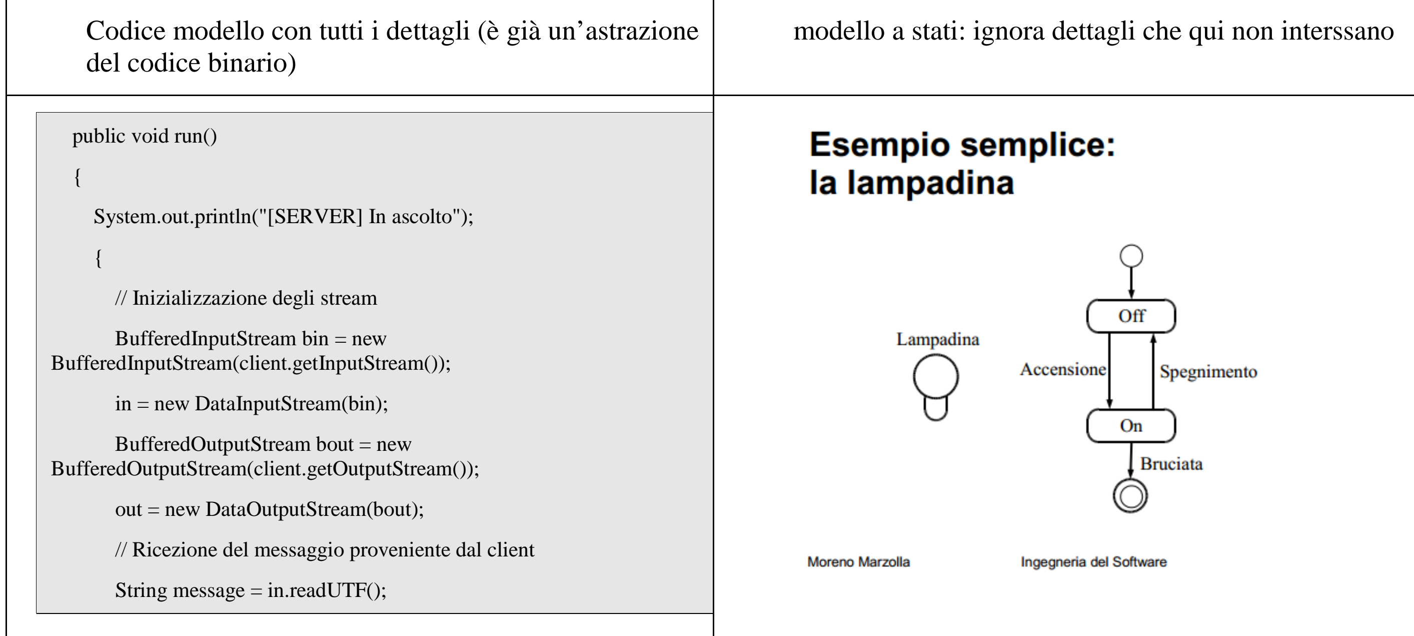
Attenzione, però, i **link DEVONO ESSERCI,** se no si torna a *due diversi sistemi indipendenti*! Moduli diversi, disaccoppiati sono due sistemi completamente diversi.



- **Astrazione**

Ignorare i dettagli. *Ragionare sui **MODELLI*** (ex circuiti elettrico descritti con equazioni differenziali che comprende *solo* alcuni elementi del circuito, astrazione del circuito fisico: non è il circuito ma mi permettono di ragionare meglio).
Ex F=m*a
Ex. S.O. modello a cipolla. Ragioniamo su un aspetto del modello ex. Memroia o kenel
Ex. Livelli OSI
Ex il codice dal modello senza linguaggi di programmazione : assembly -> Java-> modello astratto. Qui perdo una serie di elementi ma so cosa

Ex. Ingegneria del software basata su modelli , come si fa nell’hardware. Passi meccanici che non richiedono la creatività
Ex. Compilatori, oggetti, astrazione dai linguaggi di programmazione con la creazione di un modello



Codice modello con tutti i dettagli (è già un’astrazione del codice binario)	modello a stati: ignora dettagli che qui non interssano

```
public void run()
{
  System.out.println("[SERVER] In ascolto");
  {
    // Inizializzazione degli stream
    BufferedInputStream bin = new
BufferedInputStream(client.getInputStream());
    in = new DataInputStream(bin);
    BufferedOutputStream bout = new
BufferedOutputStream(client.getOutputStream());
    out = new DataOutputStream(bout);
    // Ricezione del messaggio proveniente dal client
    String message = in.readUTF();
```


Ovviamente è sempre l’uomo deve capire cosa astrarre cosa no.

- **Previsione dei cambiamenti**

Il software evolve: verificare se il sw potrà essere diverso in futuro o usato in maniere diverse. Metodologie e tecniche per verificare se il sw dovrà cambiare.

Ad ex. Versione e modifiche e compatibilità tra le versioni

- **Generalità**

Risolvere il problema generale. **Riusabilità.**

Ex. Funzione di ordinamento per campi diversi, crescente, decrescente. Più la soluzione è generale più è usabile.
Ma .. ovviamente *il costo è funzione della generalità e della riusabilità*. Quindi qual è il livello a cui fermarsi.

- **Incrementalità**

Procedere a passi successivi. Ad ex. Sviluppo a modelli per prototipi incrementali. Sviluppo a passi successivi, **prototipi incrementali** che posso verificare di passo in passo e togliere l’ansia al cliente con convalide successive. C’è un costo ovviamente e il pericolo di troppi prototipi.

NOTA: I principi ci devono guidare ma non c’è la soluzione universale di tutti i problemi però ci aiutano a capire cosa è meglio e cosa è peggio.
Avere pero’ un principio è importante!

EX Esercizio : come vengono implementati questi punti ad esempio nella costruzione di una automobile? Dov’è la modulartità, incrementalità, astrazione?

Il processo di produzione (sviluppo) del software

Il processo di produzione software è un insieme di attività il cui fine è lo sviluppo oppure la modifica di un prodotto software
Attività generiche in tutti i processi di produzione del software:

- **Specifica** – cosa deve fare il sistema e quali sono i vincoli per la progettazione
- **Sviluppo** – produzione del sistema software
- **Validazione** – verifica che il software faccia ciò che il cliente richiede
- **Evoluzione** – modificare il software in base alla modifica delle esigenze

Problemi nel processo di sviluppo software

- Specifiche incomplete/incoerenti
- Mancanza di distinzione tra specifica, progettazione e implementazione
- Assenza di un sistema di validazione

Il software non si consuma: la manutenzione non significa riparare alcune componenti rotte, ma ***modificare il prodotto rispetto a nuove esigenze***
I sistemi software sono intangibili. Pertanto è necessario **documentare** e tenere traccia di ciò che si sta facendo

Il ciclo di vita del software

Definisce un **modello** per il software, dalla sua concezione iniziale fino al suo sviluppo completo, al suo rilascio, alla sua successiva evoluzione, fino al suo ritiro. Definisce dunque il processo attraverso cui il software evolve si parla di software lifecycle o software Process.

Perché avere un modello?

Un modello permette l'esistenza di un processo pianificato e lo sviluppo NON avviene in maniera spontaneistica (caotica?), e cio` implica:

- controllo dei tempi
- controllo dei costi
- qualita` dei prodotti

Qualunque "industria" ha un modello per la produzione dei beni. Il modello consente:

- di pianificare le attivita` e le risorse necessarie
- di prevedere e controllare i costi del processo e la qualita` dei prodotti

L'ingegneria del software definisce metodi e procedure per lo sviluppo del software, utili ad ottenere sistemi di grandi dimensioni, di alta qualità, a basso costo, ed in breve tempo. Per conseguire tali obiettivi occorre puntare sulla qualità del processo di sviluppo del software il software come altre industrie manifatturiere.

Modelli di processo

Processi di produzione del Software: insieme coerente di attività per la specifica, il progetto, l'implementazione, la verifica di sistemi software
Un **modello di processo** è una rappresentazione astratta di un processo. Descrive un processo da una particolare prospettiva.
Ci sono modelli di processo generici:

- A cascata (il primo che risale agli anni '60)
    - Fasi distinte di specifica e sviluppo
- Modello evolutivo
    - Specifica e sviluppo interagiscono (prototipi)
- Modello trasformazionale
    - Un sistema matematico è trasformato formalmente in una implementazione
- Sviluppo basato sul riutilizzo
    - Il sistema è ottenuto combinando componenti esistenti
- Noi ci soffermeremo sul Modello a Cascata e sul Modello Evolutivo. Il primo perché è storicamente il più rilevante e perché da esso si evolvono i successivi. Il secondo perché è quello attualmente più usato.

Modello a cascata (waterfall)

È il modello più tradizionale di sviluppo del Software, che prevede una sequenza di fasi, ciascuna delle quali produce un ben preciso output che viene utilizzato come input per la fase successiva (da cui la metafora della cascata). Pur essendo stato soggetto a profonde critiche revisioni negli ultimi decenni (il modello è degli anni '60 del secolo scorso) , rimane il metodo di riferimento universalmente accettato, rispetto al quale tutti gli altri risultano delle varianti piuttosto che delle vere alternative.

Attività che vengono una dopo l'altra; prima completata, seconda completata…
E' un modello per sua natura DOCUMENTALE: fase, documento, fase, documento…

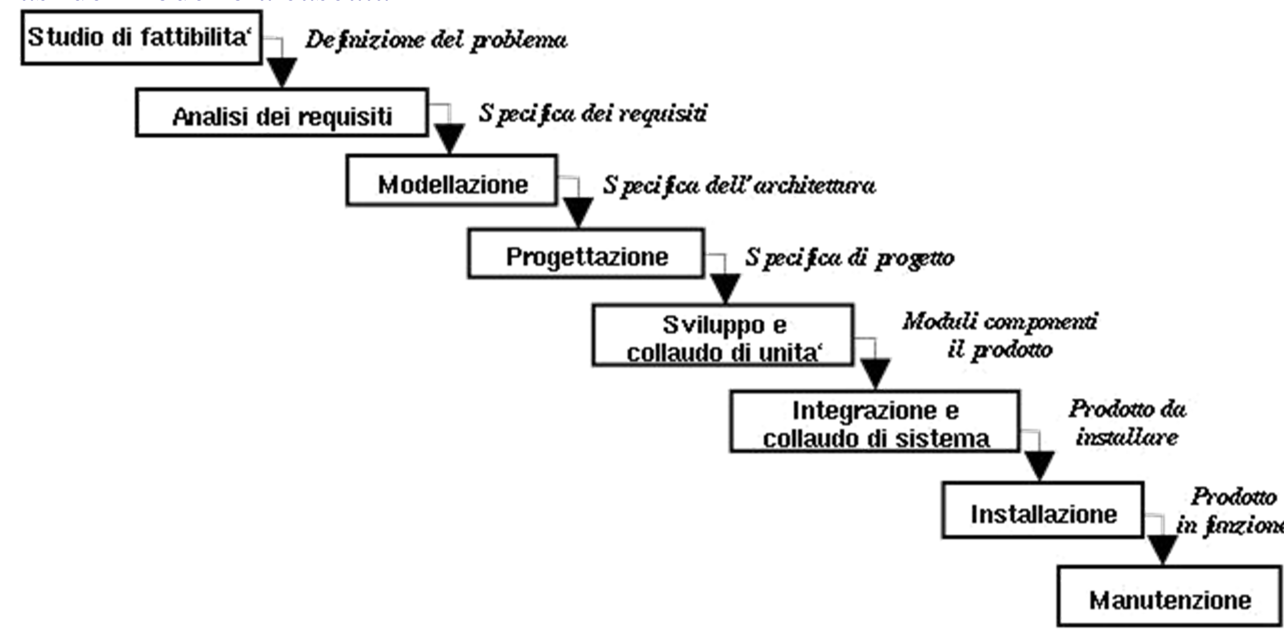
Obiettivi del modello a cascata

Cercare un metodo sistematico che consenta di

- identificare fasi e attivita` attraverso cui procedere
- standardizzare gli output di ciscuna fase (semilavorati—"artifacts")
- forzare un procedimento lineare di passaggio tra una fase e la successiva, a completamento avvenuto della fase
- nessun ritorno all'indietro, considerato dannoso perché impedisce una buona pianificazione e controllo

In definitiva: la produzione di software come catena di montaggio!

Fasi del modello a cascata



1 Studio di fattibilità (cosa)

Definizione di che **COSA deve essere fatto** e non come deve essere fatto!

- stabilire se lo sviluppo debba essere avviato (**COSTI/BENEFICI**)
- quali le **alternative possibili** e le scelte più ragionevoli.
- **stima delle risorse finanziarie e umane** necessarie, per ciascuna possibilie soluzione.
- redazione di un un Documento di Fattibilità.
-
- In pratica: siamo in grado? Ne vale la pena? Se sì, produciamo un Documento.

2 Analisi e specifica dei requisiti

CAPIRE il problema e formularlo correttamente per chi deve risolverlo.
Si va dal cliente e si cerca di capire il problema. NB il "cliente" può essere il "mercato" e non necessariamente in carne e ossa.

- **Definizione del problema**, tramite '**interviste' con il commitente**, di funzioni, vincoli, prestazioni, interfacce e di qualsiasi altra caratteristica che il sistema dovrà soddisfare.
- redazione di un **Documento di Specifica dei Requisiti Software**, che sia completo, preciso, consistente, non ambiguo, comprensibile in maniera adeguata sia al committente che allo sviluppatore.
- **predisposizione di un piano di test** e della versione o del manuale utente.
- Si produce un documento (una specifica dei requisiti) da passare alla fase di progettazione.

3 Progettazione (come)

Capito il problema nelle fasi precedenti, si cerca di trovare una **SOLUZIONE al problema** che si era individuato e capito nelle fasi precedenti.
Si parte dal Documento di Specifica e si cerca di capire **COME** realizzarlo

- **definizione dell'architettura del sistema**: definizione macroscopica dei componenti del sistema e delle relazioni tra questi (progetto di alto livello – HLD) e, via via, definiremo i dettagli: partiamo dallo stile architettonico (proprio come nella costruzione di una casa) per arrivare i componenti di dettaglio e le loro relazioni
- **definizione della struttura interna** di ciascun componente (progetto di dettaglio – LLD).
- definizione della struttura dei dati e delle interfacce utente.
- redazione di un documento di **Specifica di Progetto.**
-
- Ad esempio se l'analisi ci ha fornito un documento per la progettazione di una applicazione Web, la progettazione dovrà dividere il progetto nella parte relativa alla logica dei dati, all'interfaccia e presentazione verso l'uomo, alla sicurezza etc.

Cosi' si Dividono le Responsabilità: gruppi di lavoro relativi alle fasi stabilite (ad esempio un gruppo per la logica dei dati, un altro per l'interfaccia, un altro per la sicurezza etc.).

Si opera come nella costruzione di un edificio: qual è lo stile (romanico, neoclassico, post-moderno..), in base a questo quali colonne mettere, come dipingerlo etc. Dividere il lavoro a gruppi: gruppo dell'edificio di base, gruppo delle colonne, infissi, tetto…

4 Codifica

Implementazione dei vari componenti definiti nel Progetto.
Traduciamo i componenti individuati nel progetto in **codice** (in realtà non si tratta solo di"tradurre" ma occore una buone dose di intuizione e creatività)
NB qui il modello dimostra tutta la sua età, perché si è ignorato, fino a qui, il controllo della qualità che verrà attuata solo alla fine del lavoro.

5 Testing

Definizione ed esecuzione di "casi di prova" sia per i singoli componenti (**Test delle singole unità)** che per l'intero sistema con l'intento di rilevare malfunzionamenti.
Si procede poi con i Test di integrazione delle varie componenti

6 Istallazione (Messa in esercizio, deployment)

Insieme di tutte le operazioni necessarie per il rilascio, **l'installazione sul campo** e rendere operativo il sistema realizzato presso il committente.

- Test di sistema

7 Manutenzione

Processo di modifica di un sistema o di un componente software dopo il suo rilascio al fine di eliminare anomalie, migliorare le prestazioni o altri attributi di qualità, o adattarlo a mutamenti dell'ambiente operativo e/o del dominio applicativo.

Considerazioni sul modello a cascata

Con il modello di sviluppo a cascata, la produzione del Software è stata sottoposta per la prima volta ad una disciplina ben definita, assumendo i connotati del processo industriale. La suddivisione del processo in fasi correlate ha consentito di assegnare a ciascuna fase compiti specifici, rendendo più ordinato l'indirizzamento delle varie problematiche. Uno dei limiti maggiori di questo approccio è che il livello di parallelizzazione delle attività è piuttosto basso e questo provoca, specialmente nei progetti di grandi dimensioni, un allungamento dei tempi che può risultare inaccettabile per le esigenze del business.

Vantaggi	Svantaggi
• la fasi da seguire sono ben definite; gli output di ciascuna fase precisamente individuati. • basato sul ben consolidato modello ingegneristico per la risoluzione dei problemi • È semplice da spiegare e da capire: prima si raccolgono tutti i requisiti, poi si fa tutta l'analisi, poi tutto il design, poi tutta la codifica, ... • È semplicissimo organizzare il piano di progetto (non ci sono dubbi sulla sequenza delle fasi). • Si adatta bene a logiche organizzative e politiche del personale basate su una divisione del lavoro accentuata. • modello adeguato quando i requisiti sono ben compresi e non soggetti a modifiche	• le fasi avanzano in ordine tipicamente sequenziale . Il processo è poco efficace rispetto alla evolvibilità. Va bene se NON ci saranno cambiamenti futuri (ammesso sia possibile). Quindi… • **Difficile operare cambiamenti** una volta che il processo è in corso. Pertanto è difficile soddisfare cambiamenti nei requisiti da parte del committente (grosso limite!) • Il partizionamento in fasi può apparire arbitrario o artificioso • Difficoltà a stimare in modo accurato i costi e le risorse necessarie nella fase iniziale del progetto, quando ancora mancano sufficienti elementi di dettaglio ma è già necessario definire budget e piano di lavoro. • Necessità di aderire a precisi standard nella produzione dei documenti di progetto, con il rischio di introdurre una eccessiva burocratizzazione delle attività. • Il documento che specifica i requisiti non sempre soddisfa le effettive esigenze degli utenti perché spesso gli utenti stessi non sono in grado di conoscere e quindi di descrivere con efficacia tutti i requisiti dell'applicazione. Considerando che questo documento vincola il prodotto da sviluppare, la sua rigidità sin all'inizio del processo rappresenta un limite piuttosto significativo per la qualità del prodotto finale.

-Vantaggi e limitazioni del modello a cascata-

La Prototipazione e lo Sviluppo Evolutivo

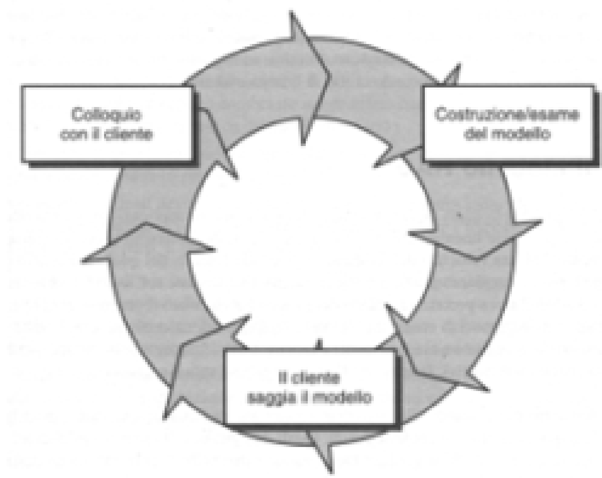
Spesso i requisiti del modello a cascata non sono abbastanza chiari. **Il cliente stabilisce gli scopi generali del**

sistema software, ma non ne chiarisce subito i requisiti in modo dettagliato. I programmatori sono *incerti sul significato dei requisiti*, o su come strutturare l'interfaccia o gli algoritmi. Inoltre i requisiti cambiano nel tempo e non posso prevederli a priori.
Ad esempio

- faccio un software per la vendita che calcola l'IVA che potrà cambiare.
- Cambio del sistema operativo e delle sue versioni. Non ho la possibilità di sapere se Microsoft cambierà le sue versioni e quando.
- Ambiente software classico spostato in ambiente client-server su Web.

In tali casi viene in aiuto una metodologia di sviluppo basata su **PROTOTIPI** (paradigma prototipale) **orientata ai cambiamenti**
I prototipi approssimano il sistema definitivo in modo incrementale.
Talvolta si usano prototipi usa e getta usati solo per provare un concetto: si prova e poi si butta via (si è capaci di "buttare"?).
Due tipi di cambiamento del processo:

- Cambiamenti nel corso dello sviluppo
- Cambiamenti in essere (una volta sviluppato il prodotto)



Il modello è legato a uno **Sviluppo Evolutivo**:
primo tipo: producono feedback e concetti nuovi
secondo tipo: **costruzione incrementale** del sistema : sviluppiamo vari livelli

Sviluppo incrementale (iterativo)

Un'idea fondamentale dello **sviluppo incrementale** (o **iterativo**) è quella che alcune parti di alcune attività vengono rimandate (ad es., la comprensione di certi requisiti) anticipando al loro posto altre attività (ad es., l'implementazione di altri requisiti) in modo da perseguire alcuni obiettivi (ad es., produrre il prima possibile un insieme utile di funzionalità, affrontare il prima possibile).
Lo stile iteartivo suddivide il progetto in sottoinsiemi con funzionalità diverse.

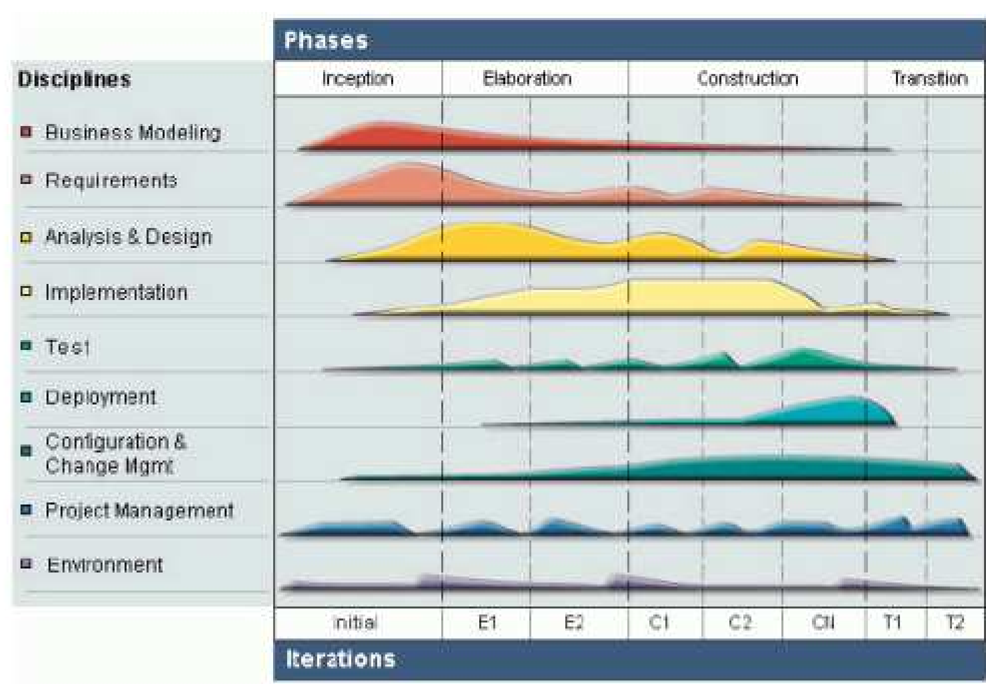
Ad esempio il corso di Teconologia viene suddiviso in due quadrimestri. Nella prima parte si svolge un intero ciclo con un test finale (i risultati sono codificati nella pagella). Il ciclo è, a sua volta diviso, in periodi di circa un mese che comprendono una attività didattica definita. Ogni periodo si conclude con una verifica. L'intero anno scolastico copre tutti i requisiti ma essi sono ripartiti in cicli ciascuno delle quali adotta tutte le fasi : spiegazione, studio ed elaborazione personale, verifica. Ogni ciclo, in se stesso è completo e si ripete circolarmente.
Se avessimo adottato un modello a cascata avremmo avuto una situazione differente: ad esmpio il primo quadrimestre sarebbe stato dedicato alle spigazioni, il secondo allo studio e alle prove pratiche e, infine avremmo avuto un test finale.
Naturalmente questa è una descrizione semplificata ma mostra le differenze tra i due approcci.
Nella pratica, spesso, i due approcci non sono così rigidi e c'è una "contaminazione" tra i due. Se ci riferiamo all'esempio precedente a cascata, potrebbe capitare che durante il secondo quadrimestre siano richieste ulterirori spiegazioni e approfondimenti. Questo è abbastanza ovvio. Tuttavia occorre, nel modello a cascata, cercare di rendere

minimi questi ritorni e rivistazioni del progetto che dovrebbero essere delle eccezioni e non la regola.

Nel caso del software, ogni ciclo deve essere completo: analisi, progettazione, codifica e testing. Se il progetto è diviso in 2 iterazioni, alla fine della prima avreste avreste ottenuto la metà delle funzionalità richieste.
Alla fine di ogni iterazione, non è detto che si debba necessariamente rilasciare un prodotto (così come un o studente che è sufficiente nel primo quadrimestre non può passare alla classe successiva!), anche se spesso si decide di rilasciare una più release in modo da avere un riscontro dagli utenti.
Nella figura sottostante è mostrato il modello incrementale perseguito da Microsoft:

Esempio di modello incrementale . Modello usato in Microsoft: Sviluppo 4-8 settimane, stabilizzazione 2-4 settimane

Nella figura sottostante è mostrato il modello incrementale in cui le fasi sono parallelizzate e sono parzialmente sovrapposte:



Un esempio di sviluppo incrementale -

Problemi

- Mancanza di visibilità (quando produrre i documenti?)
- I sistemi prodotti sono spesso poco strutturati
- Competenze particolari sono spesso richieste (es. in linguaggi per la prototipazione rapida)
- Occorre resistere alla tentazione di trasformare il prototipo in un prodotto da porre in produzione
- Al cliente il prototipo potrebbe apparire funzionante, ma in realtà potrebbe avere tali e tanti problemi da renderlo di fatto inutilizzabile in condizioni reali. Quindi il sistema va rifatto

Modello a Spirale

Il processo di sviluppo è rappresentato come una spirale, piuttosto che come una sequenza. Ogni ciclo nella spirale è una fase del processo

Al termine di ogni "giro" il risultato può essere un progetto, un prototipo, un sistema funzionante o un prodotto software completo

Non ci sono fasi predefinite. Il management del progetto deve decidere come strutturarlo in fasi

La spirale è divisa a "spicchi:"

- **Comunicazione con il Cliente**. Specificare gli **obiettivi e i vincoli** di quella fase, identificare i rischi ed eventualmente proporre delle strategie alternative
- **Pianificazione**. Attività rivolte a definire scadenze e risorse
- **Analisi dei rischi**. Attività rivolte a stimare i rischi tecnici e di gestione. Vuole ridurre al minimo i rischi di sviluppo. In un processo classico si tende a posticipare le cose difficili. Qui si cerca di anticipare ad esempio le parti nuove e di intergrazione più a rischio.
- **Strutturazione**. Attività rivolte a costruire una o più rappresentazioni (strutture alternative) del sistema
- **Costruzione e rilascio**. Attività di sviluppo, effettuata secondo un modello generico (cascata, evolutivo...)
- **Valutazione da parte del cliente**. Attività rivolte a ricevere le reazioni del cliente su quanto fin qui realizzato (nei primi giri della spirale si creano "idee" o "proposte", nei successivi anche "prodotti" o "prototipi")

Nella prima fase c'è la fattibilità etc. Vuol dire che man a mano che ci allontaniamo dal centro andiamo verso la codifica e il test. Ma qui ogni volta si ricomnicia da capo con un subrìstrato sempre più grande.

Il modello a spirale, a differenza degli altri, tiene conto dei **rischi**.
Rischio è evento imprevisto che può causare problemi o difficoltà
Esempio: Stiamo usando un compilatore per un nuovo linguaggio. C'è il rischio che il compilatore sia bacato.
I rischi sono conseguenza di informazioni insufficienti. Si risolvono acquisendo maggiori informazioni per ridurre l'incertezza.

Vantaggi del Modello a Spirale

- Valutazione esplicita dei rischi
- Adatto allo sviluppo di sistemi complessi di grandi dimensioni
- Interazione con il cliente ad ogni giro della spirale

Svantaggi

- La stima dei rischi richiede competenze specifiche
- Se i rischi non vengono valutati correttamente, potranno sorgereproblemi nelle fasi successive
- Il modello è relativamente nuovo e poco sperimentato

Processo agile (usato in open source)

Nel mondo Open Source è difficile controllare persone e processi. Ecco perché sono nati i cosiddetti "processi agili" (tra i più noti, eXtreme Programming) che sono processi di tipo iterativo. Il manifesto agile (http://www.agilemanifesto.org/iso/it/ con i suoi principi http://www.agilemanifesto.org/iso/it/principles.html) recita:

Manifesto per lo Sviluppo Agile di Software

Stiamo scoprendo modi migliori di creare software,
sviluppandolo e aiutando gli altri a fare lo stesso.
Grazie a questa attività siamo arrivati a considerare importanti:

Gli individui e le interazioni più che i processi e gli strumenti
Il software funzionante più che la documentazione esaustiva
La collaborazione col cliente più che la negoziazione dei contratti
Rispondere al cambiamento più che seguire un piano

Ovvero, fermo restando il valore delle voci a destra,
consideriamo più importanti le voci a sinistra.

I processi agili si prestano ad essere male interpretati. Semplificatori inesperti possono usarli come alibi per gettarsi subito a creare codice senza prestare attenzione ai requisiti e alla progettazione. I processi agili, in realtà, sono molto disciplinati, essendo basati su pratiche rigorose e misurabili, e sulla trasparenza degli avanzamenti lavoro e dei risultati nei confronti di tutti i partecipanti al progetto.
Non a caso, la maggioranza degli autori più rilevanti del software engineering "tradizionale" guarda ai processi agili in modo positivo.

Che cos'è l'Agilità? (cfr. http://cs.unibg.it/scandurra/material/PINF3_0809/processo.pdf)

- Reazione efficace (rapida e adattiva) ai cambiamenti
- Comunicazione efficace fra tutti gli stakeholder
- Assorbimento del cliente nel team di sviluppo
- Organizzazione del team che lo ponga in diretto controllo del proprio lavoro

Producendo …

- Consegne incrementali e frequenti di software
- Ogni iterazione è un piccolo progetto a sé stante: pianificazione (planning), analisi dei requisiti, analisi, implementazione, test e documentazione. Alla fine di ogni iterazione il team deve rivalutare le priorità di progetto

I Processi agili sono :

- Guidati dalle descrizione del cliente di che cosa gli serve (scenario)
- Basati sull'assunzione che i piani hanno vita breve
- Sviluppano software in maniera iterativa con forte enfasi sulle attività di costruzione
- Producono e consegnano molteplici incrementi software
- Si adattano ai cambiamenti

Il problema: La cooperativa "Lacoop4u" (fondata nel 2001) e' costituita da 40 soci, ognuno dei quali e' proprietario di un negozio di Ferramenta e/o Casalinghi. La cooperativa svolge la funzione di magazzino all'ingrosso per gli affiliati, cioè è in grado di ottenere i prodotti di consumo (ordinati dai soci) a prezzi favorevoli.
La cooperativa vuole finalmente dotarsi di un applicativo Web in grado di raccogliere gli ordini dei soci in modo automatico.

Obiettivo: L'applicativo Web deve permettere ad ogni utente (i soci) di accedere al sistema, previa autenticazione, consentendo di controllare i prodotti che il magazzino ha a disposizione. Inoltre al sistema è richiesta la possibilità di gestire le anagrafiche dei clienti, dei fornitori e di ogni prodotto presente in magazzino.
L'applicativo deve essere di facile comprensione e utilizzo.

A questo scopo “Lacoop4u”   emette una gara di appalto in cui è richiesto di fornire un progetto software del sistema.

Alla gara di appalto ciascuno di voi decide di partecipare con la propria società (ad esempio “Bionda-Techno & Co.”). Per questa ragione ciascun studente **deve:**

*Progettare il sistema usando la **metodologia a cascata:***
Illustrare il modello usato con un disegno e una legenda che lo spieghi
Per ogni fase citare esplicitamente cosa riceve in ingresso e cosa produce in uscita
Soffermarsi sui lati positivi e negativi di questo modello

*Progettare il sistema usando la **metodologia a spirale:***
Illustrare il modello usato con un disegno e una legenda che lo spieghi
Per ogni fase citare esplicitamente cosa riceve in ingresso e cosa produce in uscita
Comparare questo modello con il precedente spiegandone i miglioramenti

NB l’esposizione deve essere fatta in modo sintetico, rispettando esattamente i punti citati, nella successione con cui sono stati espressi.

Analisi e specifica dei requisiti (cosa)

Un progetto ha successo se soddisfa i requisiti!
Descrivere male i requisiti porta a risultati fallimentari.
Qui ci occupiamo di definire cosa un sistema debba fare, le sue proprietà essenziali ed i vincoli a cui deve rispondere.
Intuitivamente un requisito è ciò che ci aspettiamo dal sistema, ciò che andiamo a realizzare.

Un requisito del software è un'affermazione sul software che riguarda proprietà, comportamenti, vincoli (di varia natura)

E' importante scoprire,analizzare, documentare e verificare i requisititi. Questa analisi presenta forte interazione con il cliente.

Definizione e Specifica dei requisiti => Tecniche per analizzare, definire e specificare i requisiti dei sistemi software

Difficile soddisfare i requisisti perché:

- interpretazione del problema diversa (le controparti non sanno esattamente cosa vogliono). Scritti mali con modi di esprimersi diversi
- Fattori politici e organizzativi possono influenzare i requisiti del sistema
- I requisiti sono inevitabilmente soggetti al cambiamento e correggere i requisiti è traumatico e costoso
- diversi **stakeholder** (cioè chiunque ha interesse nel progetto: committente, progetto, implementazione. Marketing. Guadagno…) hanno un diverso tipo di requisiti con contraddizioni, ad esempio:
  - costo/qualità.
  - Funzionalità/semplicità.
  - Usabilità/funzionalità

Requisiti e specifiche

Qui non parliamo di come (piattaforme, codici, linguaggi) ma di **cosa**:

- **Requisiti :** cosa è richiesto nel Dominio Applicativo (tipicamente l'utente finale che vive nel D. A.)
- **Specifiche:** cosa deve fare il sistema per soddisfare le richieste del Dominio Applicativo

Cosa sono i requisiti?

I requisiti sono le caratteristiche che utente e committente desiderano che siano presenti in un prodotto software da realizzare

L'obiettivo di un progetto di sviluppo software, quando si realizza un nuovo sistema o si apportano modifiche ad un sistema già esistente, è sempre realizzare un prodotto che soddisfi i requisiti

Utente e committente valuteranno il risultato sulla base del fatto che soddisfi o meno i requisiti

Analisi dei requisiti:
• Il committente esprime una serie di vincoli
• Il fornitore formula una o più ipotesi di soluzione, in grado di rispondere, in tutto o in parte, ai requisiti espressi
• Il committente sceglie tra le soluzioni proposte quella migliore (dal suo punto di vista) in termini di rapporto tra costi e benefici, e stipula un contratto con il committente

Modello		IV-H1
Requisiti di sistema	Interfaccia	Interfaccia Ethernet (100BASE-TX)
	Sistema operativo	Windows 7 Home Premium/Professional/Ultimate 1. Windows XP Professional/HomeEdition; i suddetti sistemi operativi devono essere pre-installati
	Lingue	Giapponese / inglese / tedesco / cinese semplificato / cinese tradizionale / italiano / francese / spagnolo / portoghese / coreano
	Processore	Windows 7: deve rispettare i requisiti di sistema per il sistema operativo Windows XP: Pentium III o superiore, velocità di clock 1 GHz o superiore
	Memoria RAM	Windows 7: deve rispettare i requisiti di sistema per il sistema operativo Windows XP: 512 MB o più (consigliato 1 GB o più)
	Spazio disco richiesto per l'installazione	1 GB o più
	Monitor	Risoluzione 1024 x 768 pixel o superiore, colore di visualizzazione High Color (16 bit) o superiore
	Condizioni operative	Deve essere installato .NET Framework 2.0 SP2 2.

Cosa sono le specifiche?

Le specifiche sono la riformulazione dei requisiti in modo formalizzato per l'utilizzo nel processo interno del fornitore, in modo da definire precisamente che cosa deve fare il software da produrre

L'obiettivo di un processo di sviluppo software è realizzare un prodotto che soddisfi le specifiche
Il risultato del processo di sviluppo verrà valutato sulla base del fatto che soddisfi o meno le specifiche

Specifiche formali e informali
Il documento delle specifiche descrive in modo formale ciò che farà il software e le sue parti/moduli
• Le specifiche informali, che sono le più diffuse, sono formulate in linguaggio naturale, anche con l'ausilio di tabelle e grafici
• Le specifiche formali si basano su formalismi matematici, sono espresse in un linguaggio con sintassi e semantica definite formalmente
studenti.di3.units.it

Specifiche tecniche

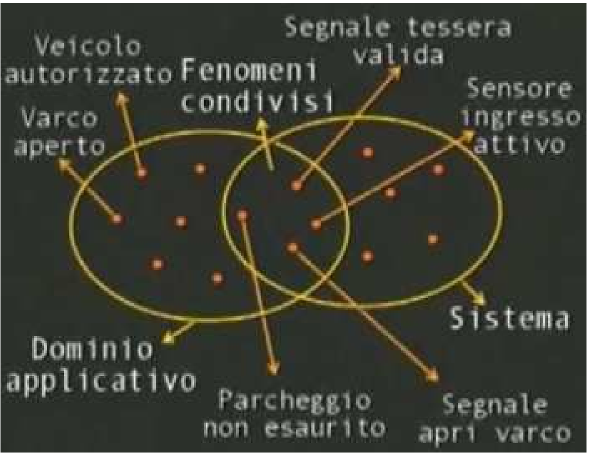
	ASUS G51J	ASUS M60J
Processore	Processore Intel® Core™ i7-720QM 1.6GHz con Turbo Boost fino a 2.8GHz	Processore Intel® Core™ i7-720QM 1.6GHz con Turbo Boost fino a 2.8GHz
Chipset	Chipset Mobile Intel® PM55	Chipset Mobile Intel® PM55
Sistema operativo	Windows® 7 Home Premium Autentico	Windows Vista® Home Premium Autentico con possibilità di upgrade a Windows® 7
Memoria	4GB SDRAM, 2 x DDR3 SODIMM 1066MHz	4GB SDRAM, 2 x DDR3 SODIMM 1333MHz
Display	15.6" Full HD (1920x1200) retroilluminato a LED	16" HD (1366x768) retroilluminato a LED con tecnologia Color-Shine
Video Scheda grafica	NVIDIA® GeForce® GTX 260M con 1GB DDR3 VRAM	NVIDIA® GeForce® GTX 240M con 1GB DDR3 VRAM
Hard Drive	640GB, 2 x 320GB 2.5" SATA 72000rpm	1 TB, 2 x 500GB 2.5" SATA 5400rpm
	Supporto Dual HDD	Supporto Dual HDD
Lettore ottico	Combo Blu-ray reader e DVD Super-Multi Dual Layer	Combo Blu-ray reader e DVD Super-Multi Dual Layer
Connettività	Intel® WiFi Link 5100 802.11 n , Bluetooth V2.1 + EDR, Gigabit LAN	Intel® WiFi Link 5100 802.11 n , Bluetooth V2.1 + EDR, Gigabit LAN
Audio	2 x 2W Altoparlanti stereo Altec Lansing con supporto EAX Advanced HD 4.0	2 x 2W Altoparlanti stereo Altec Lansing certificati Dolby® 2 Home Theatre
Video Camera	Integrata da 2M pixel	Integrata da 2M pixel
Dimensioni e peso	375mm x 265mm x 34.3~40.6mm, 3.5kg (con batteria da 6 celle)	392mm x 277mm x 32.24~44.6mm, 3.3kg con (con batteria da 6 celle)

Esempio: parcheggio controllato

se io (utente) devo entrare in un parcheggio, ciò che mi interessa è che la sbarra si alzi . Non mi intgeressa se l'apertura della sbarra è controllata o da un controllo software. Ciò è un requisito. Invece ciò che deve fare il sistema per aprire la sbarra (leggere un sensore, azionare un attuatore) e garantire i requisiti.

- **cosa** l'utente chiede al sistema e ciò che deve garantire il sistema (requisito del Dominio Applicativo).
- **cosa** deve fare il sistema per soddisfare il dominio applicativo (Specifiche), garantire i requisiti

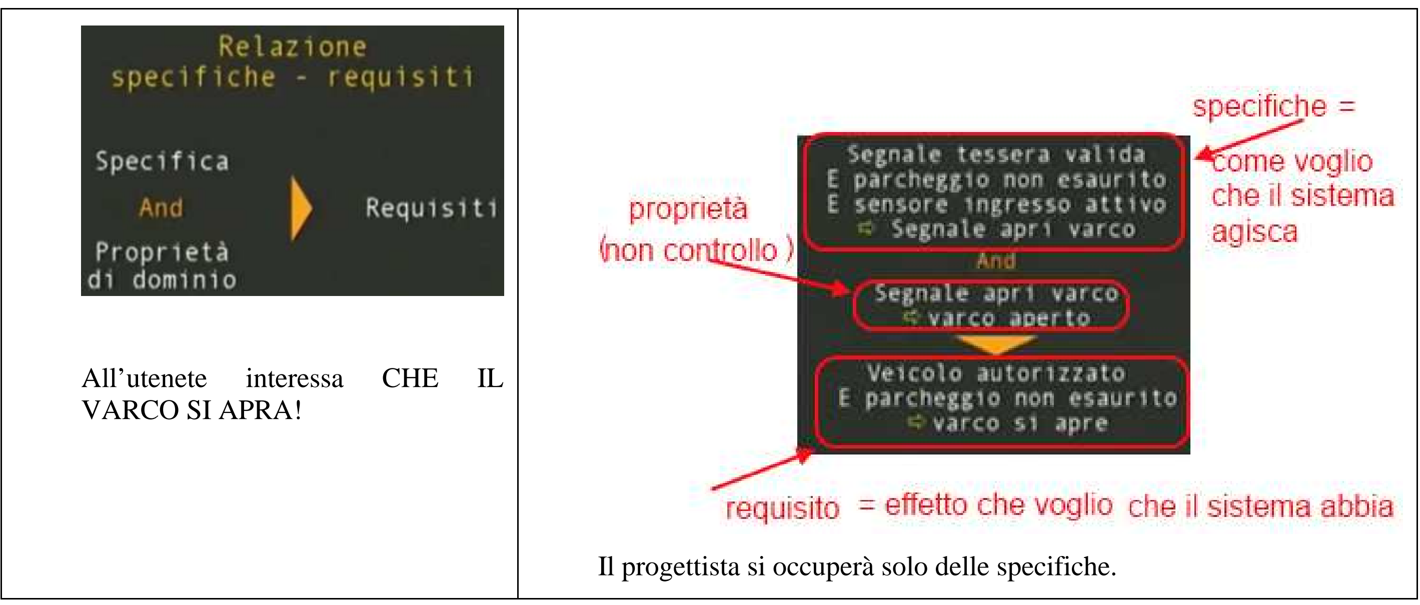
D.A. -> parcheggio (esterno al sistema)
Sistema -> controllo del parcheggio
Cosa comprende il D.A. e cosa il Sistema di controllo?
Ad esempio:
Sensore in ingresso attivo: comprende entrambe i domini (legge l'automobile) ma anche del sw che lavorerà sul segnale di ingresso
Tessera valida: entrambi
Veicolo autorizzato -> D.A.
Varco aperto -> D.A. (il sistema NON vede il varco aperto)
Apri varco : inviato alla sbarra da parte del sistema
I valori interni al sistema qui non mi ineterssano: mi interessano le realzioni (interasezioni) tra i due.
L'intersezione è necessaria tra i due domini:



Relazione tra fenomeni condivisi e sistema

Affermazione (**prescrittiva**) su D.A	Affermazione (**prescrittive**) su sistema	proprietà (**descrittiva**) del sistema, che NON impongoma osservo (descrizione del dominio : ex corpi cadono "quando il corpo non trattenuto , cade")

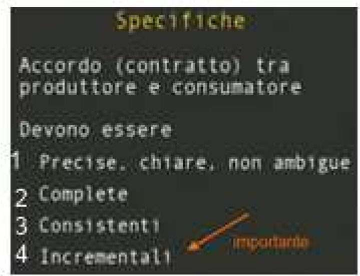
Allora possiamo anche dire che la relazione generale specifiche-requisiti è:



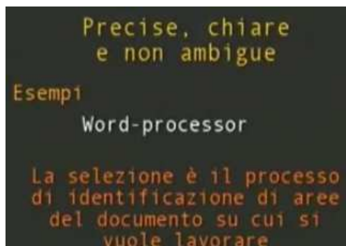
All'utenete interessa CHE IL VARCO SI APRA!	Il progettista si occuperà solo delle specifiche.

-

Specifiche



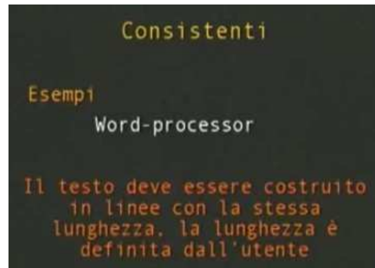
1. Precise e chiare, cioè Dettagliate, senza tralasciare elementi importanti
   Esempio di ambiguità



Ambigua: selezione in modo conscutivo, non mi chiarisce la forma dell'area da utilizzare, non definisce alcune possibilità ex. CTRL, SHIFT

2. Completezza: definire tutti i termini (glossario) (interna), inoltre documentare i requisiti necessari (esterna)
3. Consistente: garantisce di essere soddisfatta in modo univoco

Esempio inconsistente: le parole sono indivisibili? Se è più lunga della riga cosa faccio???? Potrei avere una soluzione corretta ma non soddisfa l'utente



4. Incrementale: avere più bozze sempre più dettagliate. Aggiornare le specifiche in modo incrementale

Stili di specifica

- Differenti tecniche possibili per descrivere le specifiche

Prima caratterizzazione

- **Informali**: usano tipicamente linguaggi naturali
- **Semi formali**: usano notazioni grafiche per cui la semantica non è sempre precisamente definita
- **Formali**: attraverso modelli matematici
-
- Non c'è il meglio e il peggio: dipende cosa si vuole descrivere.
-
-
- EX Esempio di notazione informale/semiformale (un sensore su un ascensore):
-

• Se un sensore non funziona, il sistema dovrebbe fermare l'ascensore in funzione ai piani precedenti o successivi

• L'ascensore deve fermarsi se il sensore del piano selezionato rileva il passaggio dell'ascensore

If Ev_ascensorepiano(x) and Direction=Down and ProssimaRichiesta(y) and x<y then Rallentare Motore;

...

Sensori — Ev_ascensorepiano(x) → Fermare& Partire

If Ev_ascensorepiano(x) and ProssimaRichiesta(y) and x=y then Rallentare Motore;

...

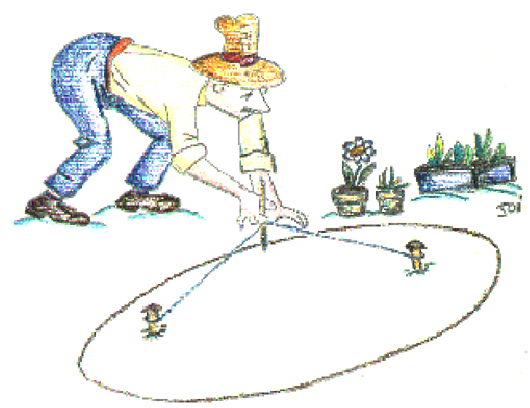
Rif. Esempio Ascensore

-
-

Seconda caratterizzazione

- **Operazionali** : descrizione con una macchina astratta
- **Descrittive** : definisce la proprietà

- Una figura geometrica: Ellisse come la descrive il giardiniere (operazionale) :
- “il metodo consiste nel piantare nel terreno i due pioli. Ai due pioli si lega la cordicella in modo tale che la parte libera risulti più lunga della distanza fra i pioli, ed uguale all'asse maggiore dell'ellisse che si vuole ottenere. Con il punteruolo si tende la funicella e lo si fa scorrere sul terreno badando che i due lati della funicella risultino sempre tesi. La traccia che ne deriva sarà costituita da punti la cui somma delle distanze dai due pioli è costante e coincide con la parte libera della funicella.” (wikipedia)

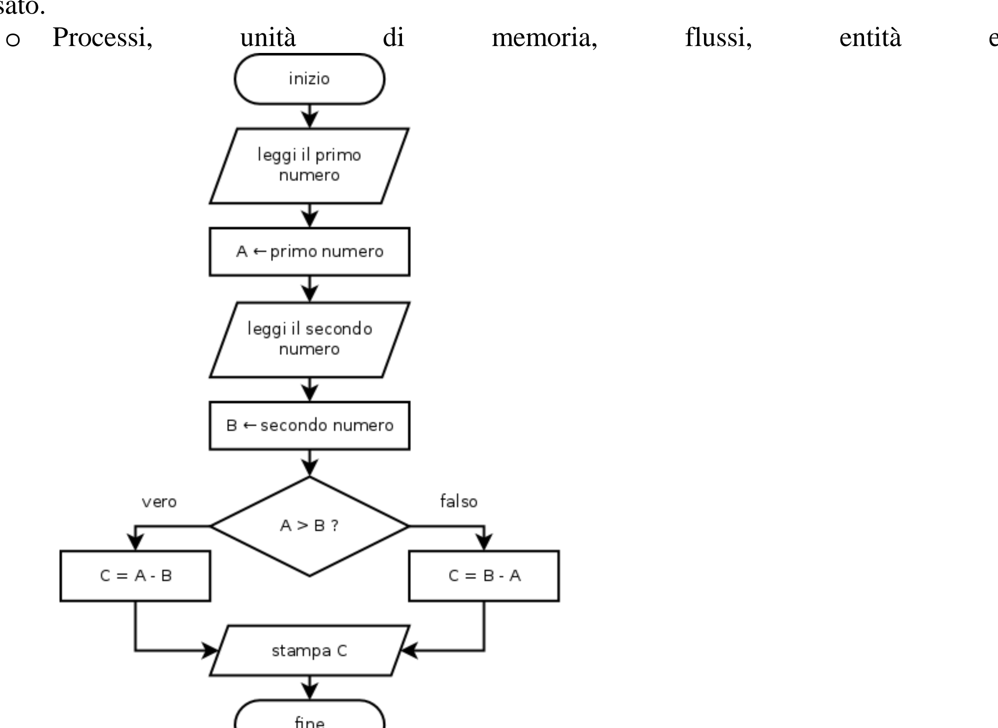


- oppure con una proprietà geometrica (descrittiva)
- “Per definizione l'ellisse è il luogo dei punti la cui somma delle distanze da due punti fissi detti fuochi è costante, cioè (F1-X1 + F2-X1) = (F1-X2 + F2 -X2) = K (costante)” $ax^2+by^2+c=0$.

Linguaggi di specifica

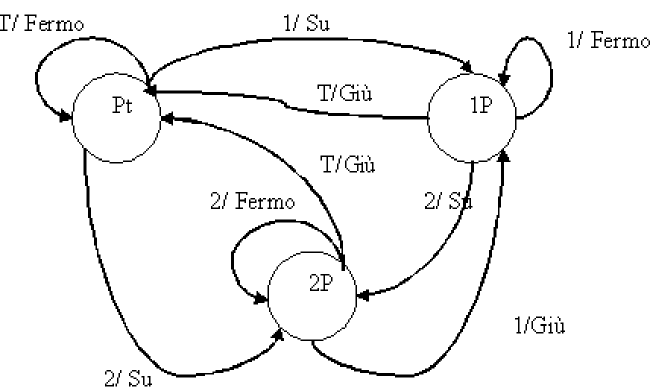
Molti sono i linguaggi di specifica usate. Qui ne mostriamo, come esempio, alcune.

1. **Diagrammi di flusso dei dati** : operazinale e semiformale . Notazione grafica. Hanno avuto enorme successo in passato.
   - Processi, unità di memoria, flussi, entità esterne connesse



2. **Macchine a stati finiti (automi)** : modello di specifica operazionale. Insieme di operazioni. Modello formale (isieme finito di stati (Q), insiseme ingressi I, elementi di transizione (trasforma stato/ingresso in stato)) e grafico.

-

-
- 

3. **Specifiche algebriche**: dominio e operazioni sul dominio

-
-

```
Sintassi: stringa
sorts
  String, Char, Nat, Bool;
operations
  new: () → String;
  add: String, Char → String;
  append: String, String → String;
  length: String → Nat;
  isEmpty: String → Bool;
  equal: String, String → Bool;
```

sorts (dominio) + operazioni (operations)

La fase di progettazione (come)

Progetto= Come fare *(e non cosa fare).*

La Progettazione (detta anche la fase di Design) 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

Durante la fase di progettazione si delineano solamente gli aspetti di alto livello, generali e piuttosto astratti della programmazione: la fase si sviluppo (coding – trasformazione in codice) sarà successiva e inizierà al completamento della fase di progettazione.

Il progetto deve:

- soddisfare i requisiti espliciti ed impliciti, basandosi sulle specifiche
- deve essere una guida leggibile e comprensibile a chi effettuerà la codifica
- deve dare un quadro completo del software per la sua implementazione, contemplando *dati, funzioni e interfacce*
- possedere un'architettura modulare con modelli riconoscibili

Le tecniche di progettazione si basano su tecniche per tradurre il modello in progetto

Regole Empiriche di Progetto

- 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
- 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
- Un software ben concegnato dovrebbe poter reagire a condizioni inusuali e arrestarsi, se necessario, in modo regolato
- Il progetto *NON E' la stesura del codice* (e viceversa!): Il livello di astrazione deve mantenersi più elevato
- 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
- 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.
    In pratica 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
E' una delle fasi più importanti e difficoltose.
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 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.

Metodologie per la progettazione di sistemi

(cfr. http://studenti.di3.units.it/Sistemi%20Informativi%20I/Slide%20Progettazione%20%28e%29.pdf)

Qualità di un progetto

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

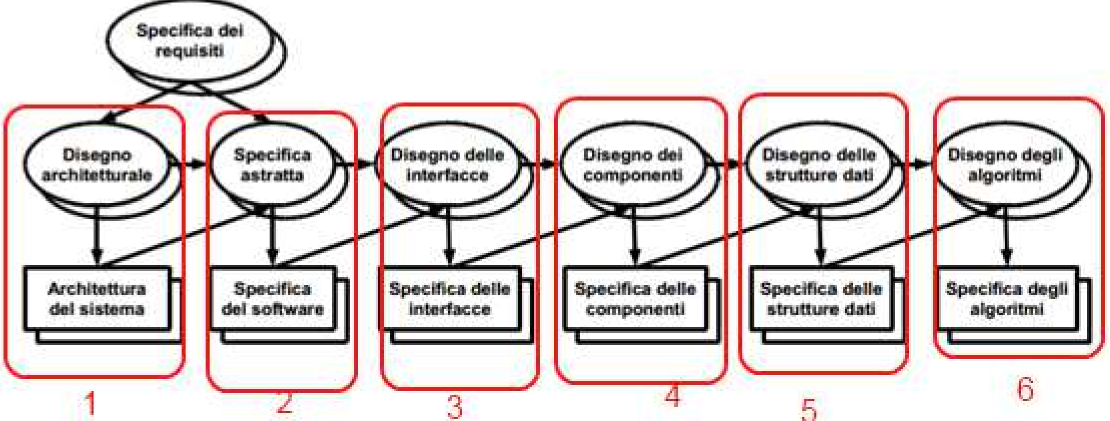
La Specifica del Progetto

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

Le fasi di progettazione

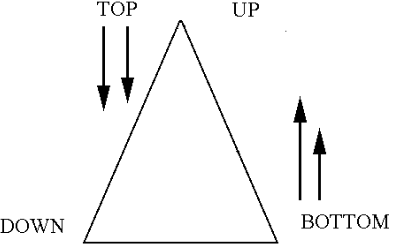
-



1. Disegno Architetturale
   a. Identificare e documentare i sottosistemi e le loro relazioni
2. Specifica astratta
   a. Specifica i servizi forniti da ciascun sottosistema, e i vincoli sotto cui il sottosistema deve operare. La specifica delle interfacce deve essere non ambigua dato che deve consentire di definire i sottosistemi ignorando come operano al loro interno
3. Disegno delle interfacce
   a. Descrive l'interfaccia dei sottosistemi verso altri sottosistemi.
4. Disegno dei componenti
   a. Allocare i servizi ai diversi componenti e definire le interfacce di questi componenti
5. Disegno delle strutture dati
   a. Specificare come sono fatte le strutture dati
6. Disegno degli algoritmi
   a. Specificare gli algoritmi utilizzati

Progettazione top-down e bottom-up

La progettazione di un sistema software e lo sviluppo successivo può essere affrontata in due modi sostanzialmente opposti.
Nel modello di progettazione top-down si parte da una visione generale del sistema e si scende , man mano, verso parti più piccole e dettagliate che, a loro volta, possono essere ulteriormente dettagliate fino alla semplificazione finale (divide et impera). In alto c'è il "problema da risolvere" e più in basso i "sottoproblemi" che lo compongono.



Questo è opposto della progettazione bottom-up, nella quale "parti individuali del sistema sono specificate in dettaglio, e poi connesse tra loro in modo da formare componenti più grandi, a loro volta interconnesse fino a realizzare un sistema completo".

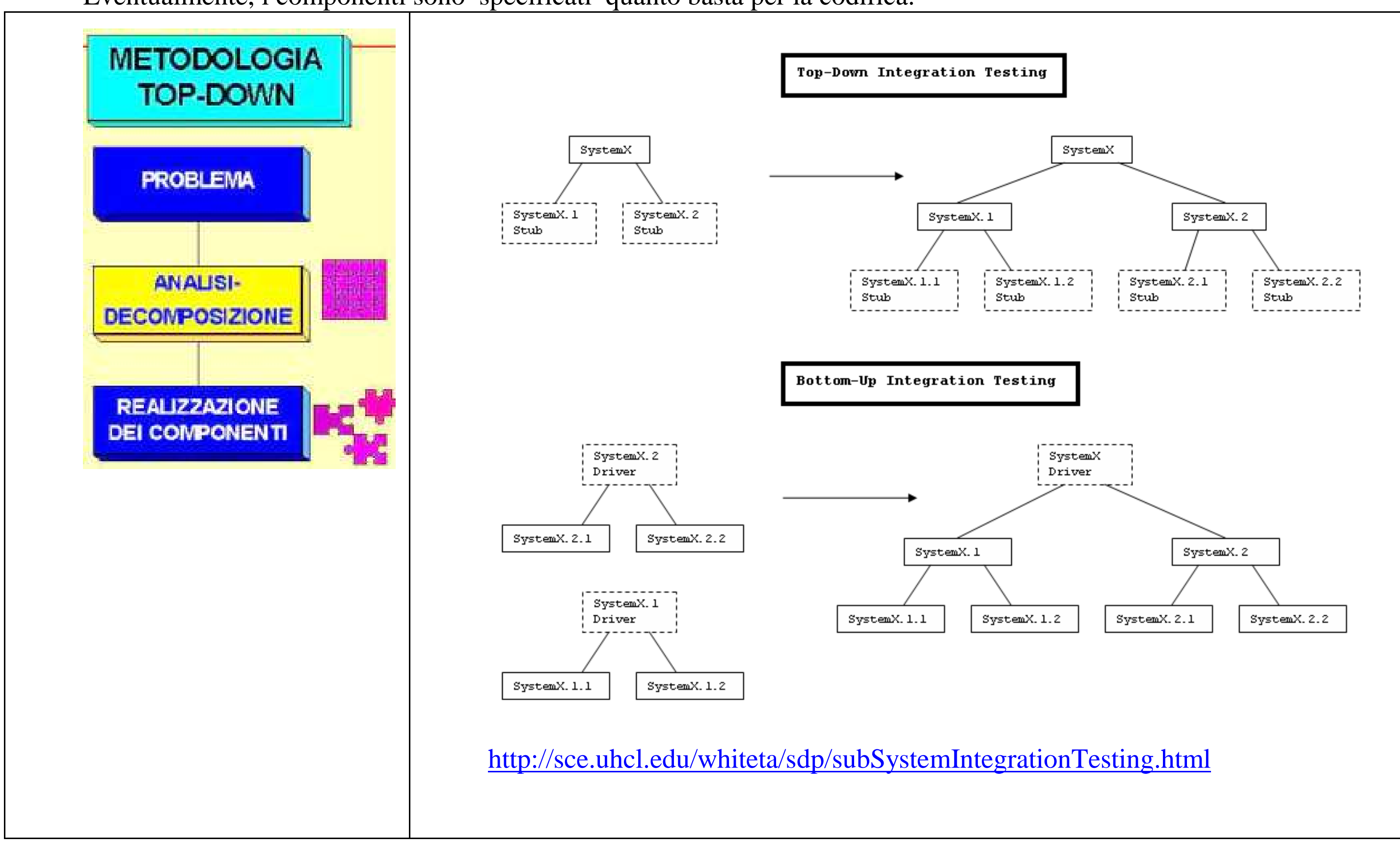
Top down in maglieria

La tecnica per fare maglioni top-down, prevede una laorazione dal collo fino al bordo inferiore, ossia dall'alto verso il basso, senza cuciture. Con questo metodo si può provare il capo in lavorazione man mano che si produce.

e in psicologia
"L'elaborazione bottom-up e top-down sono i sistemi utilizzati dalla nostra mente in determinati ambiti. L'elaborazione bottom-up (detta anche elaborazione guidata dai dati) avviene quando ad esempio entrando nella tua aula ti balza all'occhio un particolare colore rosso. Partendo dalla semplice immagine che colpisce la tua retina questa informazione risale tutti i "livelli cognitivi" che potrebbero riguardare ad esempio un certo ricordo che tu hai di quel particolare colore, le emozioni che ti suscita e via dicendo... Si chiama appunto bottom-up perchè dai primi e basilari livelli percettivi l'informazione risale ai vertici degli schemi cognitivi.
L'elaborazione top-down (elaborazione guidata dai concetti) avviene quando tu, entrando nella tua aula sai di dover cercare un amico che ti aspetta, quindi attivi tutti i circuiti adibiti alla percezione del tuo amico e inibisci tutti gli altri, quindi da uno schema complesso arriverai a percepire la semplice immagine del tuo amico. "http://it.answers.yahoo.com/question/index?qid=20121106003108AAkFkD9

Top-down

Il problema viene partizionato ricorsivamente in sottoproblemi fino a che si identificano dei sottoproblemi trattabili.
La programmazione top-down è uno stile di programmazione, fondamento dei tradizionali linguaggi procedurali, nel quale la progettazione inizia specificando parti complesse e suddividendole successivamente in parti più piccole. Eventualmente, i componenti sono specificati quanto basta per la codifica.

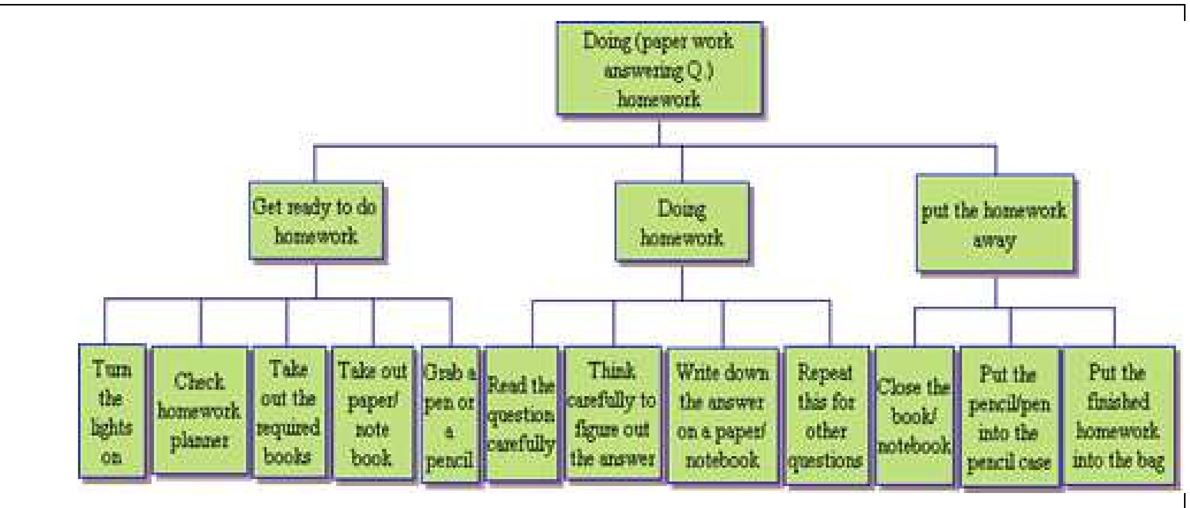


http://sce.uhcl.edu/whiteta/sdp/subSystemIntegrationTesting.html

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.

La tecnica per la scrittura di un programma mediante l'utilizzo dei metodi top-down indica di scrivere una procedura principale che indica dei nomi per le principali funzioni di cui avrà bisogno. In seguito, il gruppo di programmazione esaminerà i requisiti di ognuna di queste funzioni ed il processo verrà ripetuto. Queste sotto-procedure a comparto

eseguiranno eventualmente azioni così semplici che porteranno ad una codifica semplice e concisa. Quando tutte le varie sotto-procedure sono state codificate, il programma è realizzato.



programmazione top-down

Vantaggi:

- Il gruppo di programmazione resta focalizzato sull'obiettivo.
- Ognuno conosce il proprio compito.
- Nel momento in cui parte la programmazione, non vi sono più domande.
- Il codice è semplice da seguire, dato che è scritto in maniera metodica e con uno scopo preciso.

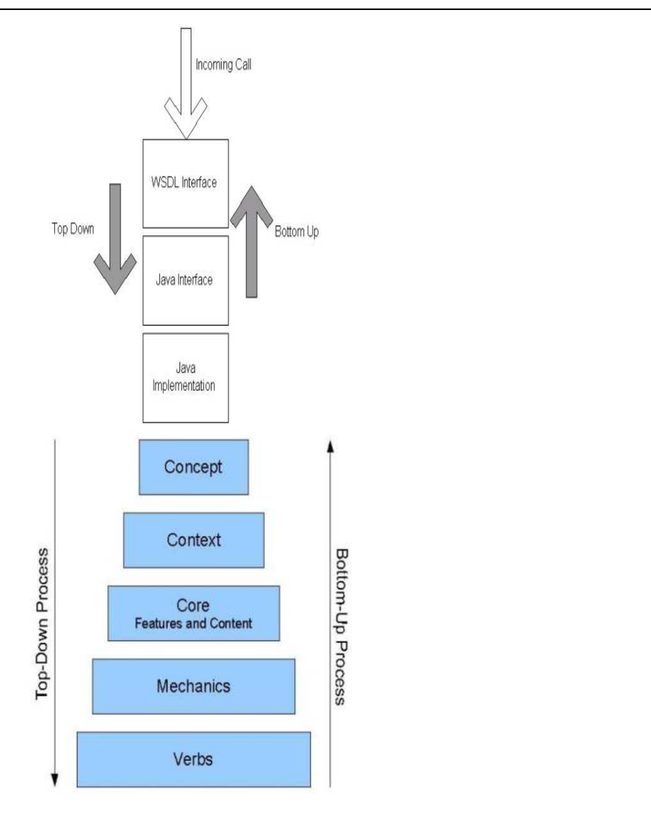
Svantaggi:

- La programmazione top-down può complicare la fase di test, dato che non esisterà un eseguibile finché non si arriverà quasi alla fine del progetto.
- Tutte le decisioni dipendono dall'avvio del progetto ed alcune decisioni non possono essere fatte sulla base del dettaglio delle specifiche.

Bottom-up

La progettazione Bottom-up parte dal basso e va verso l'alto. Sui generano o riutilizzano funzioni elementari e li si combinano in compiti più complessi risalendo fino alla risoluzione del problema. Come per la progettazione di un sistema che utilizza il lego, si creano mattoncini elementari, pioccoli pezzi di programma, che sono facili da progettare, scrivere e testare. Li si collegano tra loro producendo sistemi più complessi, fino ad arrivare all'obiettivo. Ciò comporta spesso la produzione e l'accesso a "librerie" che svolgono compiti comuni, aumentando man mano il repertorio degli strumenti a disposizione e, di conseguenza, il livello di produzione.

”Building blocks are an example of bottom-up design because the parts are first created and then assembled without regard to how the parts will work in the assembly” (Wikipedia)



Top-down vs bottom-up

(Alcune parti di questo capitolo sono tratte dal libro Perl Design Patterns Book.)

I moderni approcci alla progettazione software tipicamente **combinano sia la tecnica top-down che quella bottom-up**. Sebbene la comprensione del sistema completo è tipicamente considerata necessaria per una buona progettazione che conduce teoricamente ad un approccio top-down, la maggior parte dei progetti software cercano di fare uso di codice già esistente ad alcuni livelli. I moduli pre-esistenti danno alla progettazione una tendenza bottom-up.
Alcuni approcci di progettazione operano progettando un sistema parzialmente funzionale che viene completamente codificato, poi questo sistema viene quindi espanso fino a soddisfare tutti i requisiti di progetto.

L'approccio **top-down enfatizza la pianificazione ed una completa comprensione del sistema**. È ovvio che nessuna codifica può iniziare finché non si è raggiunto almeno un sufficiente livello di dettaglio nella progettazione di una parte significante del sistema. Questo, comunque, ritarda la fase di test delle ultime unità funzionali di un sistema finché una parte significativa della progettazione non è stata completata.

L'approccio **bottom-up enfatizza la codifica e la fase di test precoce**, che può iniziare appena il primo modulo è stato specificato. Questo approccio, comunque, induce il rischio che i moduli possano essere codificati senza avere una chiara idea di come dovranno essere connessi ad altre parti del sistema, e quel tipo di link potrebbe non essere facile. La riusabilità del codice è uno dei principali benefici dell'approccio bottom-up. La programmazione bottom-up agevola il test di unità, ma finché il sistema non si unisce non può essere testato nella sua interezza, e ciò causa spesso complicazioni verso la fine del progetto: *"Individualmente ci siamo, insieme falliamo."*

La progettazione **top-down** è stata sostenuta negli anni settanta. Il successo ingegneristico e gestionale di questo progetto condusse alla crescita dell'approccio top-down tramite IBM ed il resto dell'industria informatica. I metodi top-down erano i preferiti nell'ingegneria del software negli anni ottanta.

Il Divide et Impera e il top-down …

Il Divide et Impera è la speranza di risolvere un problema irrisolvibile scomponendolo in un insieme di sottoproblemi egualmente irrisolvibili. Per esempio, come fare entrare cinque elefanti in una Volkswagen? Facendo accomodare i primi due sul sedile anteriore e sistemando gli altri tre in quello posteriore. Il principio del Divide et Impera ha come esatto contrario quello dell'Orda Mongola (*).

Il Divide et Impera ha molto a che vedere con l'approccio Top-Down: di fronte ad un compito complesso si individuano dei sottosistemi e se ne fissano le specifiche, che vengono poi passate ai gruppi di lavoro che li devono progettare e costruire.
"si dà cioè per scontato che, se i sottosistemi soddisfanno le loro specifiche e sono connessi tra loro nel modo previsto, allora anche il sistema globale soddisferà le sue specifiche. Quanto più complicato è il sistema, tanto meno probabile è che ciò accada". (si veda anche la legge di Murphy (****).

(*) Orda Mongola - E' la speranza di risolvere un problema complesso con la sola forza del numero. Si basa sull'idea che, se un muratore costruisce una casa in cento giorni, cento muratori possono farlo in un giorno; se un aereo attraversa l'Atlantico in quattro ore, due aerei possono farlo in due ore; se a una donna per mettere al mondo un figlio occorrono 9 mesi, a nove donne... E' l'esatto contrario del principio del Divide et Impera.
L'applicazione del principio dell'Orda Mongola passa attraverso le leggi diMealy (***) e dei Mille Programmatori (**)

(**) Mille Programmatori (Legge dei) - Se assegnate mille programmatori a un progetto senza aver ben definito il disegno del sistema, otterrete un sistema di almeno mille moduli - anche se non ne dovrebbe contare più di cento.

(***) Mealy (Legge di) - Se un progetto ritarda, aggiungere una nuova persona al gruppo di lavoro consuma più risorse di quante ne produca.

(****) Murphy (Legge di) - Se qualcosa può andare storto, non mancherà di farlo.
Di tutte le "leggi" è probabilmente quella che ha valore più universale: questo concentrato di pessimismo è stato enunciato nel 1949 da Ed Murphy, ingegnere aeronautico e capitano della US Air Force

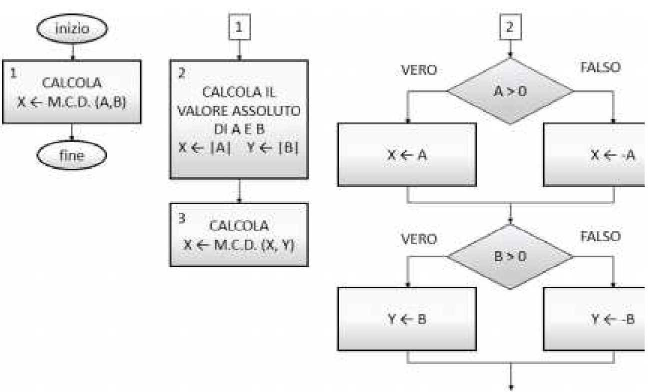
Tratto da “Il Dizionarietto del Diavolo del DP” - di Emilio C. Porcelli

La Programmazione top-down /bottom-up

La tecnica per la scrittura di un programma mediante l'utilizzo dei metodi top-down prevede di scrivere una procedura principale che indica dei nomi per le principali funzioni di cui avrà bisogno. In seguito, il gruppo di programmazione esaminerà i requisiti di ognuna di queste funzioni ed il processo verrà ripetuto. Queste sotto-procedure a comparto eseguiranno eventualmente azioni così semplici che porteranno ad una codifica semplice e concisa. Quando tutte le varie sotto-procedure sono state codificate, il programma è realizzato.

MCD approccio Top Down

Scomponiamo il problema man mano che lo affrontiamo
- Al primo livello di scomposizione abbiamo la coincidenza cc problema stesso
- Al secondo abbiamo due sotto-problemi distinti



3
VERO
FALSO
X<>Y
4 RIDUCE IL VALORE DI X E Y IN MODA TALE CHE M.C.D. (A,B)=M.C.D.(X,Y)

4
VERO
FALSO
X>Y
X ← X-Y
Y ← Y-X

http://homes.di.unimi.it/anisetti/Teaching/lezione%205.pdf

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

- 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 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

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

Il principio dell'information hiding

– il modulo non rende nota al mondo esterno una decisione ma la funzionalità che realizzaInformation 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 rilevare 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

Qual è la migliore strategia per lo 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

"... si può affermare che, nella costruzione di un nuovo algoritmo, è dominante il processo top-down, mentre nell'adattamento (a scopi diversi) di un programma già scritto, assume una maggiore importanza il metodo bottom-up." (N.Wirth)

Strategia di progetto orientata alle funzioni

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. I moduli vengono quindi a coincidere con le decisioni
- Questo approccio garantisce una coesione di tipo funzionale e, se la decomposizione è fatta bene, determina spesso un basso accoppiamento

•

La Struttura dei Dati

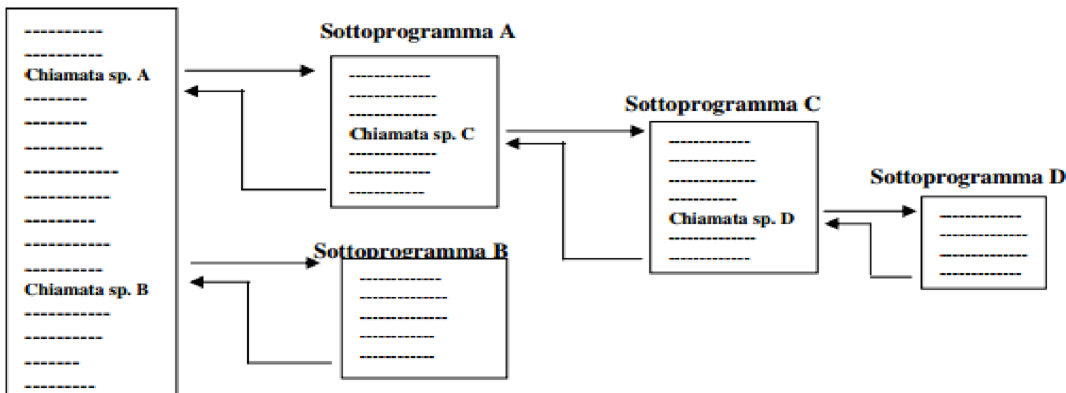
- definisce l'organizzazione, i metodi di accesso, il grado di associatività e le alternative di elaborazione per le informazioni. Organizzazione e complessità dipendono dall'inventiva del progettista e dalla natura del problema.
- Esistono diverse strutture di base che possono essere combinate
- elemento 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



- Descrive l'organizzazione gerarchica dei moduli di un programma. Un esempio classico è la struttura ad albero con Call e Return:

Un esempio di programmazione Top Down (funzionale)

Come esempio si applicherà il procedimento descritto alla risoluzione del seguente problema (http://ennebi.solira.org/c-lang/pag29.html):
In una località geografica sono state rilevate ogni 2 ore le temperature di una giornata. Si vuole conoscere la temperatura media, l'escursione termica e lo scostamento medio dalla media.
Si tratta di scrivere un programma che richiede alcune elaborazioni statistiche su una serie di valori. Si ricorda che la media aritmetica di una serie *n* di valori è data dal rapporto fra la somma dei valori della serie e il numero *n* stesso. L'escursione termica è in pratica il campo di variazione cioè la differenza fra il valore massimo e il valore minimo della serie di valori. Lo scostamento medio è la media dei valori assoluti degli scostamenti dalla media aritmetica, dove lo scostamento è la differenza fra il valore considerato della serie e la media aritmetica.
La prima stesura del programma potrebbe essere la seguente:

```
Inizio
  Acquisizione temperature rilevate
  Calcolo media e ricerca massimo e minimo
  Calcolo escursione termica
  Calcolo scostamento medio
  Comunicazione risultati
Fine
```

In questa prima approssimazione si sono evidenziati i risultati intermedi da conseguire affinché il problema possa essere risolto. Non si parla di istruzioni eseguibili ma di *stati di avanzamento* del processo di elaborazione: per il momento non c'è niente di preciso ma il problema proposto è stato ricondotto a 5 problemi ognuno dei quali si occupa di una determinata elaborazione. Viene messa in evidenza la sequenza delle operazioni da effettuare: l'escursione termica si può, per esempio, calcolare solo dopo la ricerca del massimo e del minimo.
Si noti che ad ogni fase di lavoro è assegnato un compito specifico ed è quindi più facile la ricerca di un eventuale sottoprogramma errato: se lo scostamento medio è errato e la media risulta corretta è chiaro che, con molta probabilità, l'errore è localizzato nel penultimo sottoprogramma.
Il primo sottoprogramma possiamo già tradurlo in istruzioni eseguibili. È opportuno tenere presente che a questo livello il problema da risolvere riguarda solamente l'acquisizione delle temperature rilevate. Il resto del programma, a questo livello, non esiste.

```
Acquisizione temperature
```

```
Inizio
  Per indice da 0 a 11
    Ricevi temperatura rilevata
  Fine-per
Fine
```

Passando al dettaglio del secondo sottoprogramma lo si può pensare composto da una fase di inizializzazione dell'accumulatore della somma dei termini della serie e delle variabili che conterranno il massimo ed il minimo della serie stessa. La seconda fase è il calcolo vero e proprio. Anche qui il problema è limitato solo alla parte specificata.

```
Calcolo media e ricerca massimo e minimo
Inizio
  Inizializza Somma Valori
  Considera primo elemento serie come Massimo e Minimo
  Aggiorna Somma e cerca Massimo e Minimo
  Calcola Media
Fine
```

```
Aggiorna Somma e cerca Massimo, Minimo
Inizio
  Per indice da 1 a 11
    Aggiorna Somma con elemento considerato
    Se elemento<Minimo
      Sostituisci elemento a Minimo
    Altrimenti
      Se elemento>Massimo
        Sostituisci elemento a Massimo
      Fine-se
    Fine-se
  Fine-per
Fine
```

Il terzo sottoprogramma è immediato:

```
Calcolo escursione termica
Inizio
  Escursione = Massimo – Minimo
Fine
Il quarto sottoprogramma, come il secondo, prevede una inizializzazione:
Calcolo scostamento medio
Inizio
  Azzera Somma scostamenti
  Aggiorna Somma
  Calcola Media scostamenti
Fine

Aggiorna Somma scostamenti
Inizio
  Per indice da 0 a 11
    Se elemento>Media aritmetica
      Scostamento = elemento-Media
    Altrimenti
      Scostamento = Media-elemento
    Fine-se
    Aggiorna Somma scostamenti con scostamento
  Fine-per
Fine
```

L'ultimo sottoprogramma è immediato:

```
Comunicazione risultati
Inizio
  Comunica Media
  Comunica Escursione termica
  Comunica Scostamento medio
Fine
```

Il programma a questo punto è interamente svolto. Ogni sottoprogramma ha riguardato un solo aspetto dell'elaborazione: ciò rende la stesura del programma più semplice ed inoltre la manutenzione del programma stesso diventa più semplice. Ogni sottoprogramma diventa più semplice da controllare rispetto al programma nel suo complesso.

Ciò porta ad alcuni indubbi vantaggi:

- Il sottoprogramma è facilmente esportabile
- La manutenzione del programma è semplificata dal fatto che, facendo il sottoprogramma una sola elaborazione e, avendo al suo interno tutto ciò che serve, se c'è un errore nella elaborazione questo è completamente isolato nel sottoprogramma stesso e, quindi, più facilmente rintracciabile.
- Qualora si avesse necessità di modificare una parte del programma, ciò può avvenire facilmente: basta sostituire solamente il sottoprogramma che esegue l'elaborazione da modificare. Il resto del programma non viene interessato dalla modifica effettuata.
- L'utilizzo di sottoprogrammi già pronti per la costruzione di un nuovo programma porta ad una metodologia di sviluppo dei programmi che viene comunemente chiamata ***bottom-up*** poiché rappresenta un modo di procedere opposto a quello descritto fino ad ora. Si parte da sottoprogrammi già esistenti che vengono assemblati assieme a nuovi per costruire la nuova elaborazione.
-

ATTENZIONE

Non sempre la strategia di progettazione orientata alle funzioni è la migliore, anzi può portare alla definizione di moduli fortemente accoppiati

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

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 è costituito da:

- uno stato cioè una rappresentazione interna nascosta agli altri oggetti (information hiding)
- 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.

In particolare:

- 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 he comunicano per mezzo di interfacce astratte

Differenze e analogie tra approccio funzionale e ad oggetti

<table>
<tr><td>approccio funzionale</td><td>Approccio orientato agli oggetti</td></tr>
<tr><td>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</td><td>Un oggetto è noto all'esterno come entità attive e la chiamata di un modulo è vista come la richiesta ad un oggetto di eseguire una sua funzionalità</td></tr>
<tr><td>Buona coesione</td><td>Buona coesione</td></tr>
<tr><td>coesione intorno alle funzionalità</td><td>La progettazione ad oggetti porta a coesione attorno ad entità</td></tr>
<tr><td colspan="2">La migliore strategia di progettazione?</td></tr>
<tr><td colspan="2">Si possono vedere come due strategie complementari piuttosto che alternative

Nella progettazione possiamo individuare fasi temporalmente consequenziali:

• (funzionalità) Inizialmente il sistema da realizzare si presenta logicamente costruito da un insieme di sottosistemi che realizzano delle funzionalità
• (oggetti) Ogni sottosistema può essere modellato come un oggetto al quale si associano le relative funzionalità: è opportuno usare una strategia orientata agli oggetti
• (funzionalità) 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
• (oggetti) 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</td></tr>
</table>

L’interfaccia utente

“Utente - Di lui si dice spesso che non sa quello che vuole. Sarà anche vero, ma una cosa è certa: l'utente sa benissimo quello che non vuole.”

L’interfaccia utente agisce come interfaccia tra utente e sistema e ha due responsabilità fondamentali:

- Mostrare le informazioni agli utenti
- Inoltrare le informazioni in input e avviare le elaborazioni
-

Occorre ricordare che :

- Le persone hanno memoria a breve termine limitata;
- Le persone possono sbagliare;
- Le persone differiscono notevolmente tra loro per abilità fisiche e attitudini.

Regole d'oro dell'Interfaccia Utente

Esistono 3 regole d’oro che devono guidare nella progettazione dell’interfaccia utente (Mandel, 1997)
1) Lasciare che il controllo sia nelle mani dell’utente
2) Limitare la necessità per l’utente di fare ricorso alla propria memoria
3) Utilizzare un’interfaccia uniforme per tutta l’applicazione

Queste 3 regole generali si traducono in un insieme di principi che è bene rispettare quando si definisce un’iterfaccia utente

- Quindi:
- L’nterfaccia deve utilizzare termini e concetti che siano familiari agli utenti, piuttosto che ai programmatori.
- Il sistema deve essere in grado di ripristinarsi facilmente in seguito ad un errore dell’utente.
- Aiuti, guide in linea, suggerimenti devono aiutare l’utente nell’utilizzo migliore dell’interfaccia.
- L’interfaccia deve potersi presentare in maniera differenziata rispetto alle esigenze e alle preferenze dei diversi utenti
-

Vari tipi:

- riga di comando
- menu
- form
- toolbox
- finestre di dialogo con pulsanti

Regola fondamentale (Douglas Adams)

Un errore comune che molti commettono quando tentano di progettare qualcosa di completamente fool proof” è quello di sottovalutare l'ingegnosità degli utenti

Controllo nelle mani dell'utente

- Definire la modalità di interazione in modo da non costringere
  - l’utente ad azioni inutili o indesiderate es. prevedere la possibiità di modifica del testo anche in fase di correzione ortografica
- Offrire sempre un’interazione flessibile
  - l’input deve essere possibile attraverso più canali (tastiera, menù)
- Ogni azione deve poter essere interrotta o annullata

- Nascondere all'utente i dettagli tecnici

- Prevedere modalità d'uso abbreviate (macro o short-cut) per utenti esperti
- Usare metafore che permettano la manipolazione diretta degli oggetti visibili

Processo di Progettazione dell'Interfaccia

Il processo che porta al progetto di interfaccia segue preferibilmente un andamento spiraliforme (analogo al modello di sviluppo a spirale) basato su 4 fasi

- Analisi e modellazione degli utenti, dell'ambiente e delle operazioni
- Progetto dell'interfaccia
- Implementazione dell'interfaccia
- Validazione dell'interfaccia

NOTA

"L'interazione uomo-macchina: linee guida e standard

L'interfaccia uomo-macchina è uno degli elementi caratterizzanti un sistema informatico. Infatti, ogni applicazione può essere classificata anche sulla base della modalità con cui si presenta all'utente e, viceversa, grazie alla quale l'utente può interagire con l'applicazione. Questo elemento, solitamente indicato con il nome **interfaccia utente**, racchiude quell'insieme di funzionalità di comunicazione in cui l'utente dell'applicazione può essere fonte o destinatario del messaggio. Queste funzionalità rappresentano la parte visibile di un applicativo e permettono l'inserimento dei dati di input e la visualizzazione dei dati di output. Sebbene, storicamente, le prime applicazioni per elaboratori fossero di tipo *batch*, quindi più legate all'analisi dei dati, attualmente le **applicazioni interattive**, in cui l'utente e l'applicazione dialogano costantemente, a costituire gran parte delle soluzioni.

Indipendentemente dalla tecnologia e dalle modalità di interazione, spesso accade che il progettista di un'applicazione informatica sviluppi una propria idea del profilo utente e dei requisiti dell'applicazione in sviluppo senza mai avere incontrato gli utilizzatori finali. In questi casi il modello concettuale dell'applicazione sviluppato dal progettista (che corrisponde a ciò che egli immagina dei bisogni dell'utente e alle soluzioni che implementa per permettere all'utilizzatore finale di raggiungere nel modo più efficace i propri obiettivi) può non coincidere con il modello che si è creato l'utente (il modello mentale dell'utente corrisponde all'idea che egli sviluppa del funzionamento dell'applicazione, delle operazioni eseguibili su di essa, dei comandi che vengono resi disponibili dall'interfaccia, delle azioni necessarie per fornire il proprio input e dei tipi di output che ci si deve aspettare). Quando la distanza tra i due modelli mentali à ampia, il livello complessivo di usabilità dell'applicazione informatica può essere molto basso e l'utente può andare incontro a problemi di comprensione nell'utilizzo dell'applicazione stessa.
Così come i moduli software anche le interfacce grafiche devono essere sottoposte a opportune **fasi di test** in grado di verificare non solo il loro funzionamento, ma anche l'effettiva aderenza tra i requisiti utente e l'interfaccia creata. In questo ambito sono spesso utilizzati i **test case** quali ipotetici scenari di riferimento che simulano il comportamento dell'utente finale. Rispetto al test di un'interfaccia programmatica, il test di un'interfaccia utente risulta molto oneroso in termini di definizione dei test case, visto il numero delle possibili funzionalità solitamente offerte da un applicativo. Inoltre,il grado di libertà dell'utente è molto elevato ed è difficile riuscire a coprire il grande numero di azioni che possono essere compiute sui controlli che compongono l'interfaccia. Modelli basati su macchine a stati possono divenire intrattabili visto il numero di stati nei quali un utente si può trovare. Sono pertanto stati proposti alcuni modelli che cercano di ridurre questa complessità lasciando comunque una buona copertura delle possibili situazioni.
Nella fase di valutazione della bontà di un'interfaccia è di particolare importanza la valutazione dell'**usabilità** , definita come la misura in cui una tecnologia può essere usata da particolari utenti per raggiungere certi obiettivi con efficacia, efficienza e soddisfazione, in uno specifico contesto d'uso. Da questa definizione risulta evidente che ogni volta che si parla di usabilità ci si riferisce sempre al **contesto** di utilizzo del prodotto, evidenziato sia dalla particolare classe di utenti, sia dal tipo di attività svolta con l'ausilio del prodotto stesso. Il concetto di usabilità si è sviluppato all'interno dell'ergonomia tradizionale ma, fin dal suo sorgere ha avuto forti rapporti con l'ergonomia cognitiva e, più in particolare, con gli studi volti a migliorare l'utilizzo dei prodotti a base informatica, soprattutto il software. Concettualmente l'usabilità di un prodotto, in particolare software, misura la distanza cognitiva tra il **design model** (il modello di funzionamento del prodotto che l'utente si costruisce e che regola la sua interazione con il prodotto): quanto più i due modelli sono vicini, tanto meno l'usabilità è un problema.
In questa prospettiva assume rilevanza cruciale l'interfaccia del prodotto a base informatica : essa deve rendere immediatamente conto delle possibilità, dei limiti e delle modalità di funzionamento del sistema, evidenziando la relazione tra le azioni che l'utente può compiere (manipolare oggetti, spostarli, ecc…) e i risultati che può ottenere, tralasciando le operazioni e le computazioni che consentono quelle azioni. Va infine sottolineato che il concetto di usabilità di un sistema è soprattutto pratico: l'analisi deve fornire linee guida operative per la progettazione. Infatti, al centro dell'usabilità vi è la consapevolezza che ogni alternativa di progettazione deve essere valutata il più presto possibile con gli utenti potenziali del prodotto. L'obiettivo delle valutazioni è assicurare che i prodotti a base software

siano caratterizzati da: tempi di apprendimento dei contenuti, rapida esecuzione dei compiti, basso tasso di errore, facilità di ricordare le istruzioni base, alta soddisfazione dell'utente. Per progettare interfacce caratterizzate da un buon livello di usabilità sono state sviluppate linee guida e standard differenti. L'ISO 9241 ha definite l'usabilità come "the extent to which a product can be used by specified users to achieve specified goals with **effectiveness**, **efficienty** and **satisfaction** in a specified context of use". In questa definizione **l'efficacia** indica l'accuratezza e la completezza con cui gli utenti possono raggiungere I loro obiettivi, mentre l'**efficienza** riguarda le risorse necessarie per farlo e la **soddisfazione** si riferisce al comfort e al grado di accettabilità dell'esperienza d'uso.

Nielsen, di recente, ha definito cinque principi fondamentali di usabilità:

1. **Apprendibilità**: la facilità di apprendimento delle funzionalità e del comportamento dell'applicazione
2. **Efficienza**: il livello di produttività ottenibile dopo che l'utente ha appreso il funzionamento del sistema
3. **Memorizzabilità**: la facilità di ricordare le funzionalità del sistema in modo che l'utente casuale possa ritornarvi dopo un periodo di non uso senza aver bisogno di un nuovo periodo di apprendimento
4. **Riduzione degli errori**: la capacità del sistema di prevenire gli errori e di aiutare l'utente a non compierne; se l'utente sbaglia ugualmente, deve essere messo in grado di rimediare in modo facile e veloce
5. **Soddisfazione dell'utente**: la misura in cui l'utente trova il sistema piacevole da usare.

Lo stesso Nielsen ha anche proposto una valutazione euristica del sistema. Essa consente di analizzare le specifiche di un sistema in riferimento a delle euristiche , o linee guida, ossia i principi consolidati di buona progettazione e ergonomia. La valutazione euristica consiste in un giudizio da parte degli esperti valutatori sull'aderenza o meno dell'interfaccia alle linee guida di usabilità. Le linee guida, basandosi sulle caratteristiche cognitive e comportamentali degli individui durante l'interazione con il sistema informativo, definiscono e descrivono le proprietà di un sistema che possa essere definito usabile. **Un prodotto è usabile quando consente di raggiungere, in poco tempo e con buona tolleranza agli errori, gli scopi per i quali gli utenti lo utilizzano.**

La valutazione euristica è un metodo detto **discount**, in quanto consente di comprendere in tempi rapidi i principali problemi di usabilità nell'interazione con il sistema (per esempio: assenza o non correttezza dei feedback sulle azioni, comprensibilità del linguaggio utilizzato, efficacia dei supporti alla navigazione). Tali risultati possono rappresentare un primo screening delle criticità di interazione dei siti. La valutazione euristica è un metodo ispettivo, ovvero viene applicato da esperti di usabilità. Altri metodi prevedono invece il coinvolgimento attivo degli utenti (per esempio, **cognitive walkthrough**) e valutazione basata su scenari); tra essi il più noto è il test di usabilità.

Il **test di usabilità** consente di valutare nel dettaglio le performance e le reazioni dell'utente nei diversi stadi di sviluppo del sistema tramite l'uso dei prototipi e fornisce indicazioni per le modifiche da apportare nelle fasi successive di design-redesign. Per condurre un test di usabilità bisogna compiere una serie di passi fondamentali, come indicato di seguito:

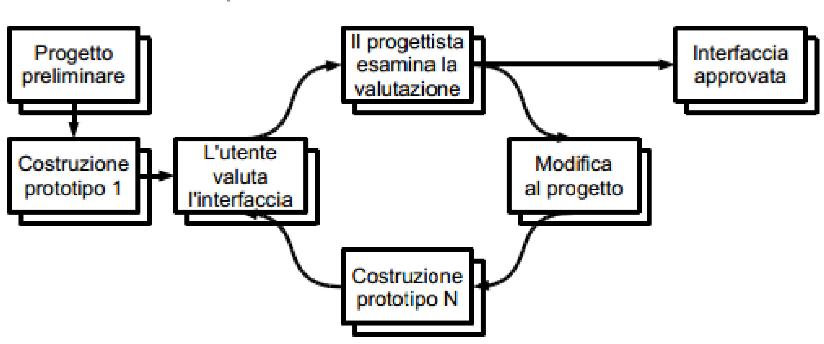
1. Definire gli obiettivi del test (generici o specifici)
2. Definire il campione dei soggetti che partecipano al test. Il numero di valutatori influenza il numero dei problemi di usabilità individuabili
3. Selezionare test case rappresentativi delle attività che gli utenti svolgeranno sul sistema
4. Decidere i criteri di valutazione dell'usabilità (qualitativi e quantitativi)
5. Predisporre l'ambiente di testing e verificare che il prototipo supporti i test case selezionati
6. Eseguire il test, documentandone lo svolgimento (registrazione audio o video, appunti scritti, log delle interazioni)
7. Analizzare i risultati

La definizione dei test case è una fase cruciale e anche difficile visto il numero di possibili situazioni. Alcuni approcci per questa definizione si basano sui modelli legati al mondo dell'intelligenza artificiale. Il planning, per esempio, è più goal-oriented e permette, una volta definito lo stato finale, di ricostruire i cammini che portano a quello stato, eventualmente definendo anche il punto iniziale. In alternativa vi sono approcci basati su **event-flow**, sulla definizione cioè di test case sulla base della sequenza di azioni che devono essere compiute."

Valutazione del Progetto

La valutazione passa per un test di utilizzo dell'interfaccia da parte di un utente

- Informale: un utente usa il sistema e fa un report
- Formale: gruppi di utenti utilizzano il sistema e vengono utilizzati metodi statisitici per valutare le reazioni



"NO PROBLEM è il metodo che ha fatto la fortuna di molte aziende medio-piccole del nordest."

SCHEMA PER LA SOLUZIONE DI OGNI PROBLEMA

FUNSIA?
SI
NO
NO STA TOCARE
GHETO TOCA'
SI
BRAO MONA !
NO
POE DAR PROBLEMI?
SI
POITO DARGHE LA COLPA A QUALCHEDUNI?
NO
'VANTI COSI'
LO SA QUAL'CUNI?
NO
SI
TE SI' CIAVA'
NO
SI
NESUN PROBLEMA

http://eliasmengwee.wordpress.com/category/linee-sottili/page/2/

Soluzioni per la progettazione

UML

UML è un linguaggio di modellazione, cioè una notazione anche grafica usata per esprimere le caratteristiche di un progetto, nell'ambito della programmazione ad oggetti.

Esso cerca di fornire una soluzione completa alla progettazione di un sistema, sia da un punto di vista **statico** che **dinamico**.

- UML è il successore di una serie di metodologie di analisi e progettazione orientate agli oggetti.
- Il processo proposto da UML consiste in una serie di "consigli" riguardanti i passi da intraprendere per produrre il progetto stesso.
- Il modello UML tiene conto dei diversi aspetti funzionali, temporali e dei dati.
- Il modello presenta varie ***"viste"*** che descrivono i diversi comportamenti della realtà. Ogni vista è rappresentata con un *diagramma* grafico e testuale insieme

Per un approfondimento: http://www.analisi-disegno.com/uml/introuml.pdf

Tipi di diagrammi UML

Esistono due grandi famiglie di diagrammi che descrivono aspetti diversi del sistena:

- aspetti statici
  - casi d'uso
  - diagramma di classi
- aspetti dinamici
  - diagrammi di sequenza
  - diagrammi di stato
  - diagrammi di attività
  - Diagrammi dei Componenti e di Deployment
-

un esempio
Viste diverse del Duomo di Milano

Diagrammi relativi agli aspetti statici di un sistema

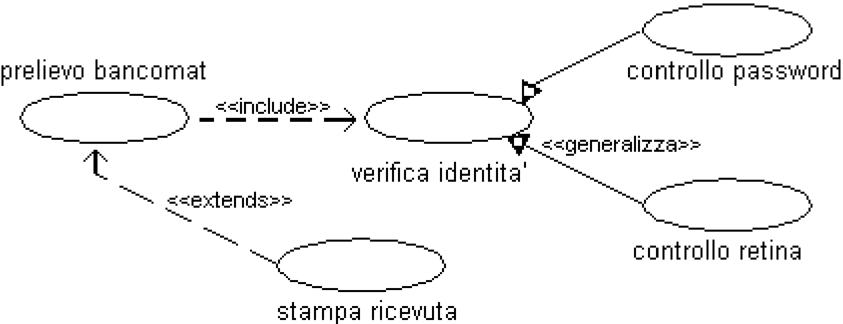
Diagrammi dei **Casi d'uso**

- Un caso d'uso è un insieme di azioni legate da un obiettivo comune per l'utente. I diagrammi di casi d'uso descrivono ad alto livello l'interazione tra il sistema e uno o più attori che richiedono un servizio. Ad esempio:

Esempio: Acquisto di un prodotto

- Descrive le modalità di utilizzo del sistema, come è visto dall'utente (attore).

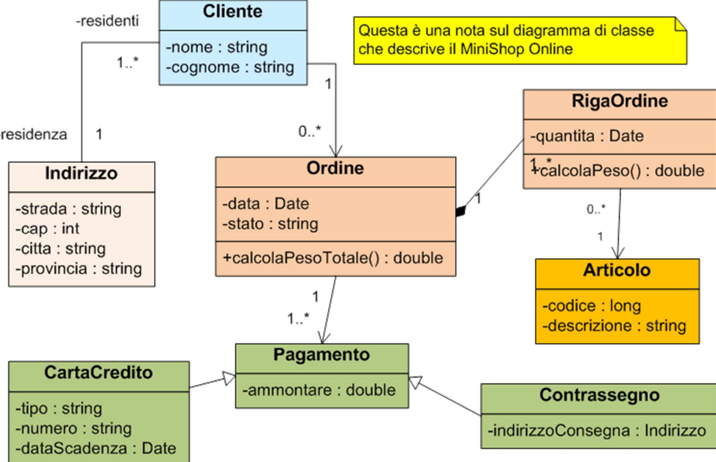
Si rapopresenta come un grafo che mostra le interazioni tra un utente e l'aspetto di un sistema. Risulta particolarmente utile durante la fase di specifica dei requisiti.



- http://www.ba.infn.it/~regano/tesi/tracker_model_file/UML.html

Diagrammi di Classe, degli Oggetti

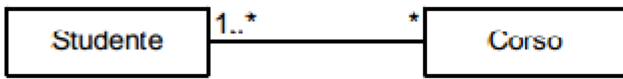
- Descrive la scomposizione in classi e le relazioni tra esse. Inoltre mostra una rappresentazione delle istanze delle classi, cioè gli oggetti



-

Partendo dalle specifiche del sistema si ricavano le classi che lo possono caratterizzare, descrivendoil tipo degli oggetti che compongono il
sistema e le **relazioni** statiche esistono tra loro.

La principale relazione statica è l'**Associazione**.
Una associazione rappresenta una relazione tra le istanze di due classi
Es: Uno studente frequenta più corsi, un corso frequentato da più studenti

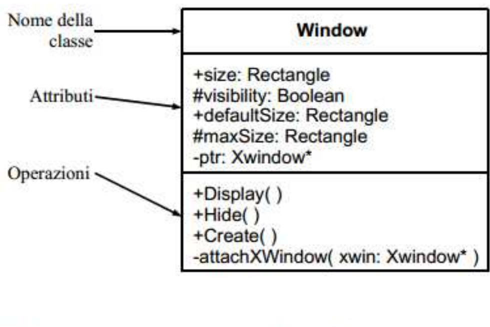


Es: Un Ordine deve venire da un singolo Cliente, e che un Cliente può fare più Ordini nello stesso tempo

I diagrammi delle classi mostrano anche gli **attributi e le operazioni** di una classe, e lerestrizioni che si applicano al modo con cui gli oggetti sono collegati tra loro.
Concetto intuitivo: l'attributo "nome" della classe. Cliente indica che i clienti hanno un nome
Gli attributi individuano lo stato interno degli oggetti di una data classe

Le operazioni rappresentano le computazioni che una classe sa come effettuare. Possono riportare all'esterno informazioni sullo stato degli Oggetti. Possono modificare lo stato degli oggetti

Una classe UML è rappresentata da un rettangolo all'interno del quale sono presenti nome, attributi e comportamenti/operazioni (metodi)

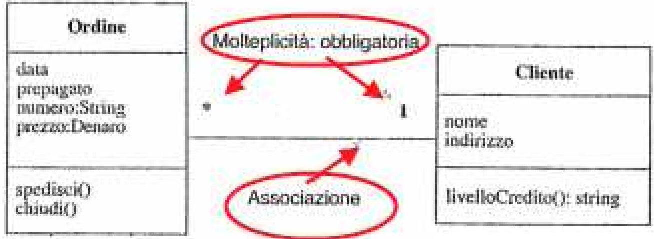


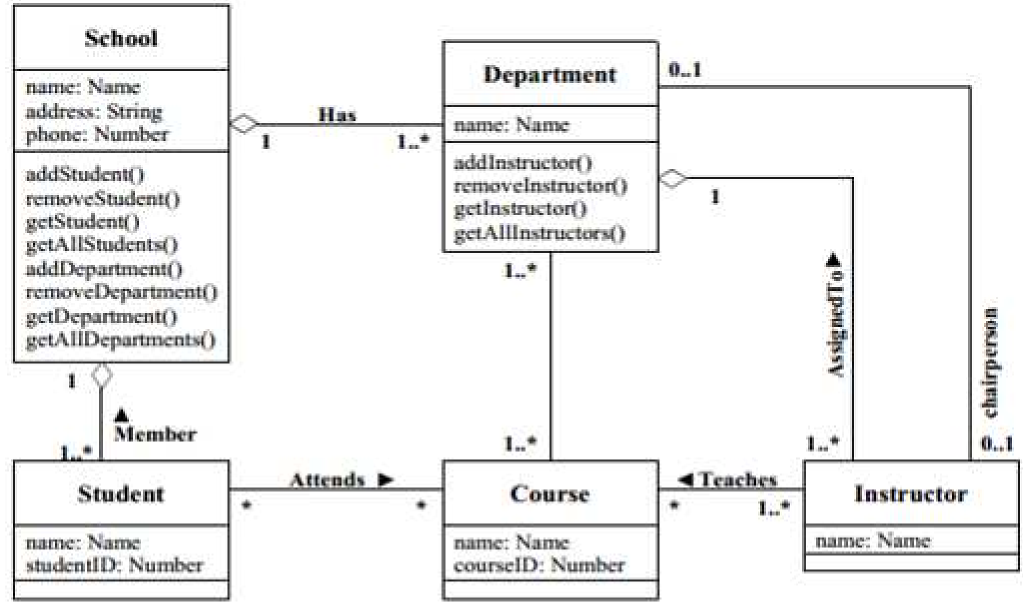
Nota i simboli +-# indicano la visibilità, cioè se possono essere usati, o meno, da oggetti di altre classi.

Molteplicità delle Associazioni

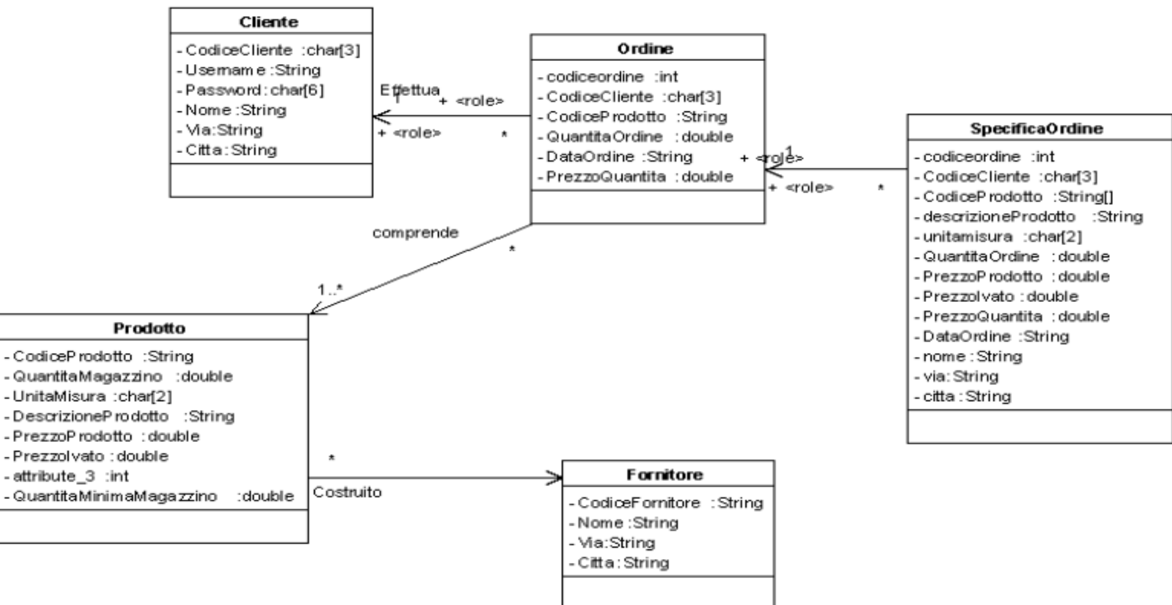
Gli estremi di una associazione possono essere etichettati con il nome del ruolo. Se non è indicato, il ruolo è il nome della classe

- Il capo di una associazione ha una molteplicità. Indica quanti oggetti possono prendere parte alla relazione
- La molteplicità si può anche indicare con un intervallo m..n
- L'asterisco * rappresenta l'intervallo da 0 a infinito
- Nella pratica, le molteplicità più usate sono 1, * e 0..1





Moreno Marzolla Ingegneria del Software 2



http://www.giuliodestri.it/doc/ingsw/D04_GuidaEsame.pdf

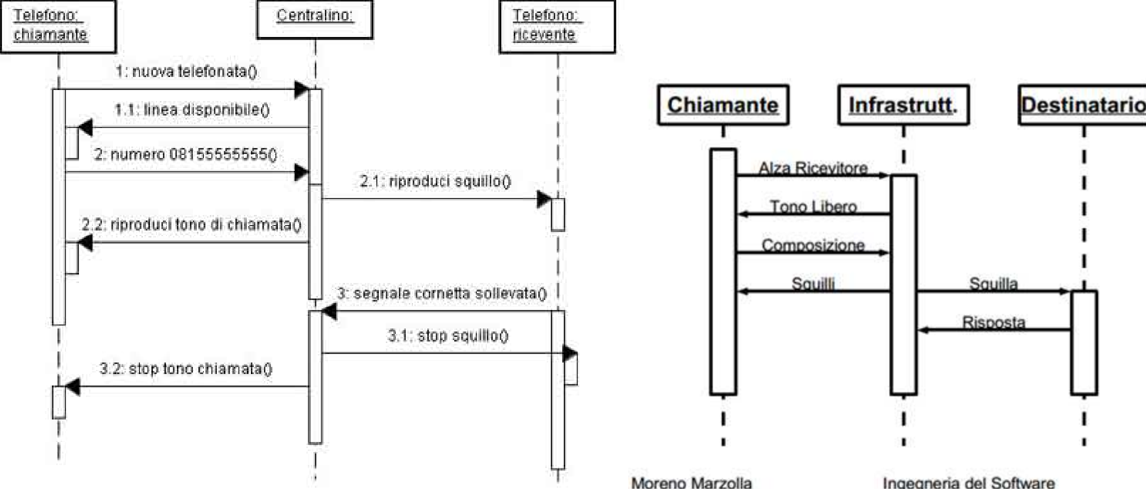
Diagrammi relativi agli aspetti dinamici di un sistema

Fino a qui abbiamo visto i diagrammi che rappresentano la struttura statica di una Architettura Software
Occorre però anche caratterizzarne il comportamento dinamico

Diagrammi di Sequenza

- Mostra l'ordinamento temporale dei messaggi scambiati tra gli elemrnti del sistema.

I Diagrammi di Sequenza sono utilizzati per mostrare quali comunicazioni vengono effettuate tra oggetti.



Diagrammi di Stato

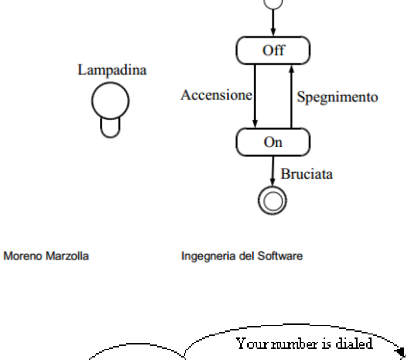
- Mostra gli stati che il sistema può assumere durante il suo ciclo di vita.

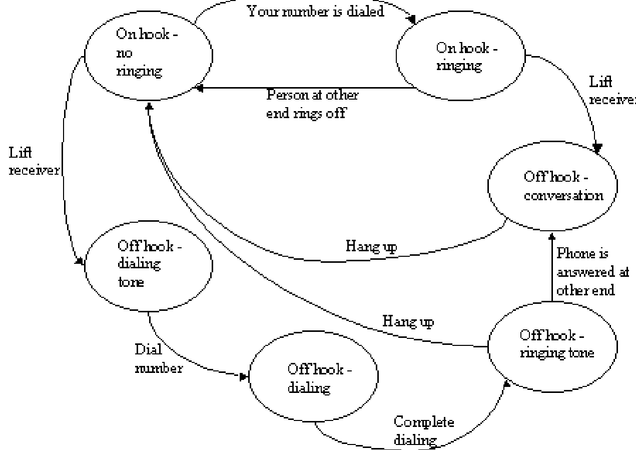
Descrivono l'evoluzione del sistema

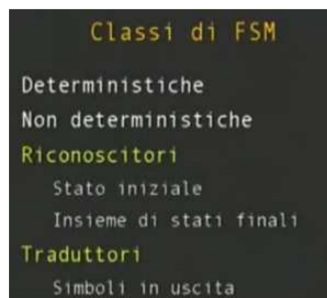
Sono costituiti da una serie di stati che descrivono delle attività, tra i quali ci si muove attraverso delle azioni

Le azioni sono associate a cambiamenti di stato (transizioni), quindi sono considerate processi rapidi e non interrompibili

Le attività sono associate a stati, possono prendere un lasso di tempo più lungo e possono essere interrotte da un evento







Transizioni e attività

La sintassi generale di una etichetta associata ad una transizione è:

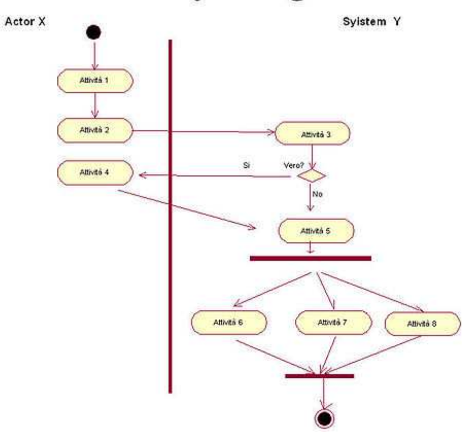
<Evento> [<Condizione>] / <Azione> “Esegui L'azione se si verifica l'evento, e la condizione è vera”

La sintassi di una attività è più semplice: **do/ <nome attività>**

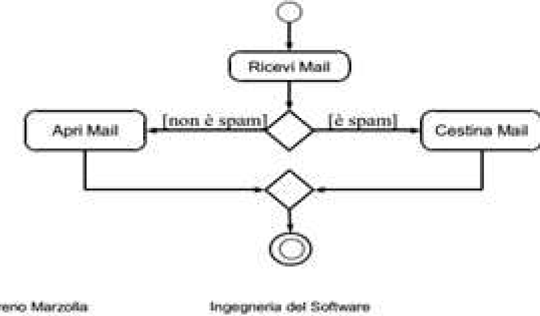
Diagrammi di Attività

- Mostra le elaborazioni che internamente avvengono nel sistema in funzione dei suoi cambi di stato

Activity Diagram



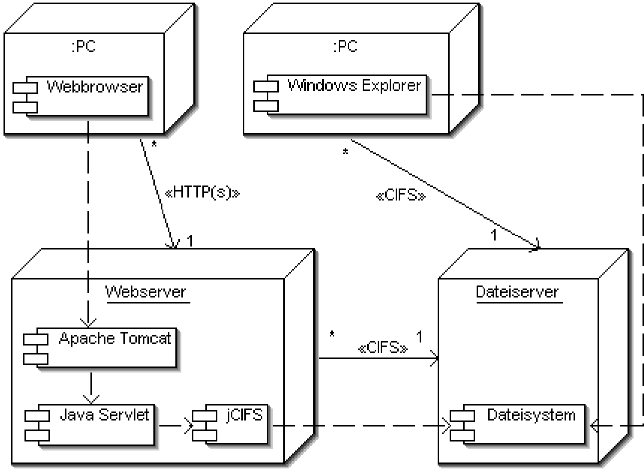
Descrivono la sequenza delle attività e supportano un comportamento sia condizionale che parallelo
Due costrutti fondamentali: Branch (diramazione) Merge (giunzione)



http://www.simple-groupware.de/cms/ext/files/WebDisk/uml_deployment.gif

Diagrammi dei Componenti e di Deployment

- Mostra come sono organizzate le componenti fisiche del sistema (file, librerie, eseguibili, moduli…)



Collaudo del software

Collaudo del software (test) =Garantire che il sistema software sia privo di errori e difetti

Principi fondamentali

- Nelle prime fasi del processo di sviluppo software si passa da una visione astratta ad una implementazione concreta (implementazione)
- A questo punto occorre iniziare una serie di casi di prova destinati a "demolire" il software realizzato
- Il **collaudo** è l'unico passo del processo software che si può considerare (dal punto di vista psicologico) **distruttivo** anziché costruttivo
- Il collaudo deve infondere un senso di colpa? I collaudi sono distruttivi? *Ovviamente no*!
-

Nel collaudo ci domandiamo due cose:

- Scoprire i difetti presenti nel sistema (**verifica**)
  - Stiamo costruendo il software in modo corretto?"
  - Il software deve essere conforme alle specifiche; cioè, deve comportarsi esattamente come era previsto
  - Are we making the right product?

- Verificare se il sistema è o meno utilizzabile in condizioni operative (**validazione**)
  - "Stiamo costruendo il giusto software?" Il software deve essere implementato in modo da soddisfare le attese degli utenti e dei committenti"
  - Are we making the product right?

Definizioni

Errore (umano): incomprensione umana nel tentativo di comprendere o risolvere un problema, o nell'uso di strumenti.

Difetto (fault o bug): manifestazione nel software di un errore umano, e causa del fallimento del sistema nell'eseguire la funzione richiesta.

Malfunzionamento (failure): incapacità del software di comportarsi secondo le aspettative o le specifiche; un malfunzionamento ha una natura dinamica: accade in un certo istante di tempo e può essere osservato solo mediante esecuzione.

Ad esempio:

```
Function RADDOPPIA ( )
....
read (x);
y := x*x;
write (y)
...
```

ERRORE di editing/digitazione

DIFETTO (causato da un errore)=> "*" invece di "+"

MALFUNZIONAMENTO => il valore visualizzato è errato. Possibile MALFUNZIONAMENTO in esecuzione… (può verificarsi o meno: dipende dall'input)

Relazione fra errore, difetto e malfunzionamento:

errore	causa	Difetto	causa	Malfunzionamento
	1..* — 1..*		1..*	

Standard IEEE per il Testing

Standard IEEE 610.12-1990:
(1) The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component.

IEEE Std.729-1983
(2) The process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software item.
http://www.federica.unina.it/ingegneria/ingegneria-software-ii/verifica-convalida-software/

Il testing ha lo scopo di scoprire i difetti di un programma. Un test dei difetti *ha successo* se porta il programma a comportarsi in maniera scorretta (cioè esibisce malfunzionamenti).
Il testing può solo dimostrare la presenza dei difetti, ma non la loro assenza!! (Tesi di Dijkstra)
Serve a:

- Dimostrare al cliente e allo sviluppatore che il software soddisfa i suoi requisiti Si parla di *Test di Convalida*
- Verifica che il software si comporti adeguatamente usando un insieme di casi di test che riflette l'uso previsto
- Scoprire gli errori o difetti del software Si parla di *Test dei Difett*

Tale test *ha successo* se tutto funziona come ci si aspetta

Approccio usato: scoprire la presenza di difetti osservando i malfunzionamenti eseguire un programma al fine di scoprire un errore

- Rivela la presenza di errori, NON la loro assenza
- Un caso di prova è valido se ha un'alta probabilità di scoprire un errore ancora ignoto
- Un test è riuscito se ha scoperto un errore prima ignoto (non il contrario!

Testing e debugging

Testing e debugging sono due attività distinte

- Testing: confermare la presenza di errori
- Debugging: localizzare e correggere tali errori

Strategie Problemi e Limitazioni

Strategie:

Testing Top-down
I test cominciano dalle componenti più astratte, e ci si muove via via verso le componenti di basso livello

Testing Bottom-up
I test cominciano dalle componenti più di basso livello, e ci si muove via via verso le componenti più astratte

Alpha testing:
uso del sistema da parte di utenti reali ma nell'ambiente di produzione e prima della immissione sul mercato.

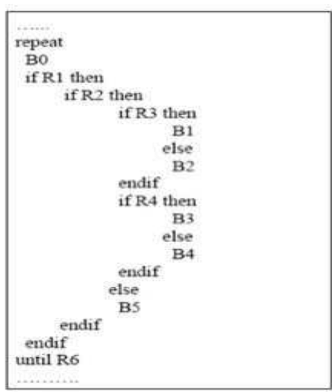
Beta Testing:
installazione ed uso del sistema in ambiente reale prima della immissione sul mercato.
Tipicamente adottati dai produttori di packages per mercato di massa!

Problemi:

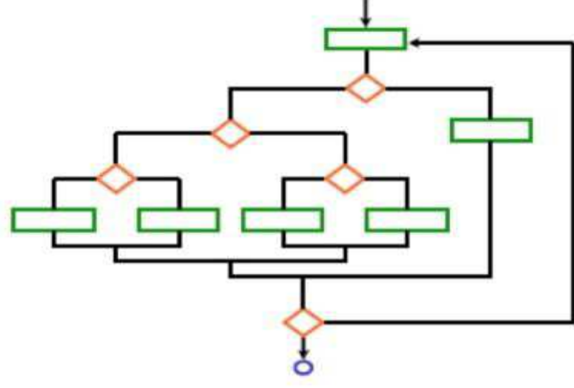
La correttezza di un programma è un problema indecidibile!
non vi è garanzia che se alla n-esima prova un modulo od un sistema abbia risposto correttamente (ovvero non sono stati più riscontrati difetti), altrettanto possa fare alla (n+1)-esima;
Impossibilità di produrre tutte le possibili configurazioni di valori di input (test case) in corrispondenza di tutti i possibili stati interni di un sistema software.

Ulteriori Problemi
In molti campi dell'ingegneria, il testing è semplificato dall'esistenza di proprietà di continuità. Se un ponte resiste ad un carico di 1000 tonnellate, allora resisterà anche a carichi più leggeri. Nel campo del software si ha a che fare con sistemi discreti, per i quali piccole variazioni nei valori d'ingresso possono portare a risultati scorretti. Il testing esaustivo (ideale) è condizione necessaria per poter valutare la correttezza di un programma a partire dal testing.

L'impossibilità del Testing Esaustivo!
Se il ciclo fosse eseguito al più 20 volte, ci possono essere oltre 100.000 miliardi di possibili esecuzioni diverse!!! Se ogni test fosse elaborato in 1 ms, sarebbero necessari 3170 anni!!!



```
......
repeat
  B0
  if R1 then
        if R2 then
              if R3 then
                    B1
                  else
                    B2
              endif
              if R4 then
                    B3
                  else
                    B4
              endif
            else
              B5
        endif
  endif
until R6
..........
```



Esercitazioni

«La brevità consiste nel dire molte cose in poche parole e, se fosse possibile, a far pensare più di quanto si dica»   (Roukhomovsky, 2001)

Fare progetto

(http://www.dmi.unict.it/~nicolosi/LezioniPS200809/Lezione1.pdf http://www.giuliodestri.it/ingsw.shtml)

Obiettivi

Coinvolgervi nello sviluppo di un progetto software in cui
– mettere a frutto le conoscenze che avete acquisito (programmazione, algoritmi, ingegneria del software, …)
– applicare un processo per la gestione e lo sviluppo del progetto analogo a quelli usati nel mondo reale

Come?

Lavorerete in team di 2/3 persone

- un leader
- uno o più analisti
- due o più programmatori

Sarete voi a scegliere il ruolo più adatto alle vostre capacità e alla vostra indole. Mi comunicherete la formazione del vostro gruppo ed eventuali modifiche.

I Leader

• Problem solving:
– determinare i fattori tecnici e organizzativi più importanti
– strutturare sistematicamente una soluzione
– motivare gli sviluppatori
• Identità manageriale:
– assumersi la responsabilità del progetto
– coordinazione del resto del team
• Incentivazione:
– ricompensare lo spirito di iniziativa
– saper rischiare
• Influenza e spirito di gruppo:
– Mantenere il controllo nelle situazioni critiche
– relazionarsi bene coi colleghi

Il team

"Se vuoi migliorare in modo incrementale, sii competitivo; se vuoi migliorare in modo esponenziale sii cooperativo." Autore ignoto

• Staff tecnico auto-organizzato
• Parola d'ordine: cooperazione

Criteri di valutazione
• Qualità del progetto sviluppato
• Coesione del gruppo

RICORDA

Cos'è un progetto software?

Temporaneo & Unico
• Temporaneo: Ogni progetto ha un inizio e una fine ben precisa (obiettivo raggiunto o irraggiungibile)
– non significa che dura poco infatti molti progetti durano diversi anni
• Unico: Ogni progetto riguarda qualcosa che non è mai stata fatta prima ed è pertanto unica
– un progetto differisce da un altro per il tipo di oggetto sviluppato, per gli strumenti usati, per le persone coinvolte o soltanto per le date Progetti software

Il lavoro di ingegnerizzazione del software viene normalmente organizzato in progetti
Sistema software di piccole dimensioni: team di sviluppo unico con tre/quattro sviluppatori
Sistema software di dimensioni grandi: il lavoro viene suddiviso in molti progetti piccoli

Tipi principali di progetto:

• Modifica di software esistente
• Sviluppo di un sistema da zero
• Sistema nuovo costruito con componenti già esistenti

Progetti software

In generale non possiamo produrre software nella manierapiù immediata
1. I progetti sono in genere di dimensioni consistenti
2. Possono richiedere l'applicazione di più strumenti (linguaggi di programmazione, algoritmi)
3. Possono coinvolgere più persone (sviluppatori, committenti e utenti)

Esempi

Sviluppo di:
• Un compilatore, un editor, un sistema operativo
• Un applicativo web
• L'implementazione di un algoritmo di ordinamento
• Un dimostratore automatico
• …Progetti software/ingegneria software

Esempio: programma di elaborazione testi

• Stadio 1: funzioni base relative alla gestione dei file, al trattamento del testo e alla produzione di documenti
• Stadio 2: funzioni più raffinate di trattamento del testo e di produzione dei documenti
• Stadio 3: controlli ortografici e grammaticali
• Stadio 4: funzioni avanzate di impaginazioneIl modello incrementale 3

Progettare con gli EAS

Fasi EAS	Azioni dell'insegnante	Azioni dello studente	Logica didattica
Preparatoria	Assegna compiti Disegna ed espone un framework concettuale Fornisce uno stimolo Dà una consegna	Svolge i compiti assegnati Ascolta, legge e comprende	Problem solving
Operatoria	Definisce i tempi dell'attività Organizza il lavoro individuale e/o di gruppo	Produce e condivide un artefatto	Learning by doing
Ristrutturativa	Valuta gli artefatti Corregge le misconceptions Fissa i concetti	Analizza criticamente gli artefatti Sviluppa riflessione sui processi attivati	Reflective Learning

http://inanimatealice.edu.au/

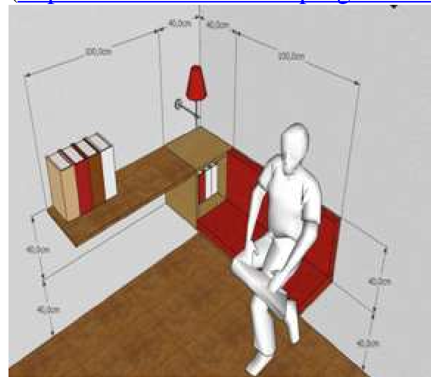
DUPLone Assembler

Costruire l'assemblatore di DUPLone

Esempi di progetti di ordine generale

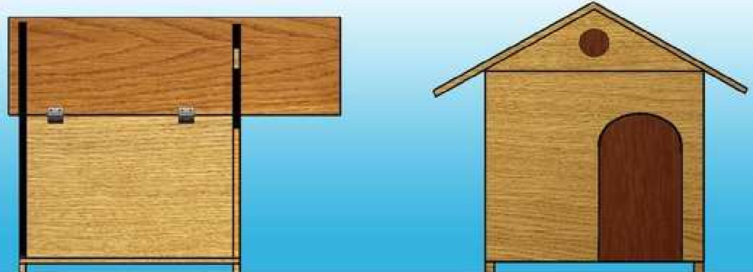
⚠EX Progettare una mensola porta-oggetti

(https://it.formabilio.com/progetto-concorso/nuvola-4414)



⚠EX Progettare la cuccia del vostro cane

(cfr. http://www.insidehomeart.it/tutorial/falegnameria/380-come-realizzare-una-cuccia-in-legno?showall=1&limitstart=)



Alcune domande da porsi prima di iniziare

- Che taglia ha il cane?
- Dove deve trovarsi la cuccia : località geografica, enti atmosferici, esterna, protetta…?
- Materiale?
- Budget§?
- Attrezzi?
- Tempi di realizzazione?
- Controlli automatici?
  - Sensori di temperatura, umidità, timer cibo/acqua?
- Giochi per cani?
- …

⚠EX Progettare la potatura di una siepe

⚠EX Progettare un giardino

Appendice

Classificazione dei sistemi

I sistemi si presentano in molteplici forme e grandezze. Alcuni sono limitati e seguono sempre la stessa traettoria, percorrendo sempre gli stessi stati come ad esempio il semaforo. Altri sono decisamente piu' complessi e non sappiamo come potrebbero evolvere in futuro : un sistema economica, sociale o anche naturale.
Quando però studiamo un sistema possiamo cercare di classificarlo facendolo rientrare in una certa categoria, esattamente come i biologi classificano insetti o piante.
Ad esempio distinguiamo i mammiferi dagli ovipari i vertebrati dagli invertebrati .
Anche per i sistemi possiamo cercare di classificarli in funzione di alcune loro caratteristiche salienti.
I sistemi possono essere classificati in base a :

- La Natura
- Il Tempo
- Proprietà delle variabili
- Proprietà delle relazioni

Se il criterio di classificazione è la **natura** del sistema dobbiamo considerarne l'origine e classificarli quindi in:

- ***Naturali*** *--> esistenti in natura*
- **Artificiali** --> realizzati dall'uomo
- **Misti** --> naturali, ma modificati dall'uomo (es. dighe)

Se il criterio di classificazione è il **tempo** dobbiamo distinguerlo in :

- **Discreto** --> puo' esistere in uno e solo uno di un certo numero di stati separati. Il sistema deve saltare da uno stato all'altro e non svanire in modo continuo. Ad ex. Passa da 1 a 2 , dal rosso al verde, un grafico a punti, etc.
- **Continuo** -->non hanno uno stato ben definito ma si muovono in modo impercettibile da uno stato all'altro. Ad ex. Un' auto che si muove su una strada, un grafico continuo, i numeri reali etc.

Se il criterio di classificazione si basa **sulle proprietà delle variabili** dobbiamo distinguerlo in :

- **Dinamico** --> quando le variabili di stato cambiano valore nel tempo
- **Statico** --> quando le variabili di stato assumono sempre lo stesso valore
- **Chiuso** --> quando non interagisce con l'ambiente esterno , cioè non riceve nulla e non lascia uscire nulla. Ad es. un sistema fisico di termodinamica con lo studio delle variabili di stato di un gas perfetto . Questi sistemi come si puo' ben intuire sono chiusi solo in teoria perché non sono mai perfettamente isolati dal mondo esterno, Tuttavia se questi scambi sono talmente piccoli da essere ignorati possiamo egualmente considerarli chiusi!
- **Aperto** --> quando interagisce con l'ambiente esterno: essi assumono materiale, energia, informazione dall'esterno e possono ritrasmettere tutto cio' all'esterno. I sistemi biologici sono sistemi aperti. Un calcolatore è un sistema aperto. Una rete, Internet sono sistemi aperti. Essi sono molto importanti perchè sono dotati di Ingressi e Uscite e possono reagire agli elementi esterni e possono anche autoregolarsi.

Se il criterio di classificazione si basa sulle **proprietà delle relazioni** dobbiamo distinguerlo in :

- **Deterministic**o --> quando ad una certa sollecitazione risponde con una sola e univoca risposta . Si pensi al semaforo in cui la sequenza degli stati è del tutto decisa in anticipo e la traettoria è sempre la medesima nel tempo.
- **Probabilistico** o Stocastico --> quando ad una stessa sollecitazione può rispondere in diversi modi. Pensiamo a una roulette, Facendola ruotare ripetutamente si ottengono risultati apparentemente imprevedibili. Cio' pero' non è del tutto vero . Infatti , come per il lancio dei dadi o di una moneta, siamo in grado di dire qualcosa sulla frequenza con cui appaiono i numeri. Ad esempio se lanciamo una moneta sappiamo che la probabilità che esca testa è del 50%, anche se non possiamo dire cosa uscirà al prossimo lancio.
- **Combinatorio** --> quando l'uscita dipende solo dagli ingressi e non dallo stato interno. I sistemi combinatori sono detti anche "Sistemi senza memoria".
- **Sequenziale** --> quando l'uscita dipende sia dagli ingressi che dallo stato interno

Esistono altre classificazioni, piu' specifiche e dettagliate. Senza entrare in merito a queste classificazioni, riportiamo solo alcuni esempi.

- Sistema trascendentale Secondo il modo di operare e la loro natura fisica (C. Jones):
- Sistema manuale
- Sistema automatico
- Sistema uomo-macchina
- Sistema biologica
- Sistema fisico
- Sistema simbolico

…

secondo un altro schema a livello crescente (K. Boulding):

- Sistema fisico
- Sistema dinamico
- Sistema cibernetico o autoregolato
- Sistema aperto o di automantenimento
- Sistema genetico di gruppo (botanico)
- Sistema animale
- Sistema umano
- Sistema sociale

NOTA: Il modo di pensare sistemistico ha permesso di uscire dai ristretti campi specialistici delle varie scienze e di capire che non sempre i sistemi reali ricadono necessariamente nelle divisioni delle varie scienze.
In particolare i sistemi che riguardano l'interazione uomo-macchina e, nello specifico, i sistemi di elaborazione che interagiscono con l'uomo escono dai confini rispettivamente dalla fisiologia-psicologia e ingegneria per costituire qualcosa di nuovo e diverso.
La cibernetica , cioè la scienza del controllo e della comunicazione, valica i confini della singola disciplina ed entra nel terreno della interdisciplinarietà.

Automi a stati finiti

Gli Automi a Stati Finiti sono usati in informatica da lungo tempo anche in sistemi particolarmente popolari, come il software per i videogiochi e la robotica.

*Una **Macchina a Stati Finiti (FSM) o Automa a Stati Finiti** (o semplicemente una **Macchina a Stati**), descrive il modello di comportamento di un sistema composto da un numero finito di stati. Definisce, inoltre, le transizioni tra questi stati, e le azioni, in modo simile a un grafico di flusso in cui si può controllare la logica di funzionamento.*

L'Automa descrive il sistema in base al solo suo "funzionamento" senza alcun riferimento alla "cosa" (macchina o altro) e al modo con cui è realizzata.

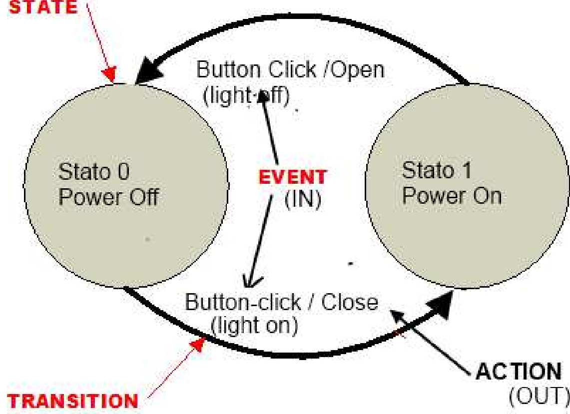
Da un punto di vista formale l'Automa costituisce il **modello di un Sistema Stazionario e Discreto.** In pratica si considerano sistemi Stazionari che dipendono solo dallo stimolo ricevuto (*Ingresso o Evento*) e dallo *Stato Interno,* nel senso che lo stesso stimolo produce lo stesso effetto se lo stato interno è lo stesso, indipendentemente dall'istante in cui viene attivato, e sistemi Discreti con un numero finito di Stati e di valori di uscita.
Il sistema si trova di volta in volta, in stati definiti e stabili che possono evolvere, in seguito ad ingressi che gli arrivano, verso altre situazioni stabili. Tali transizioni sono accompagnate, eventualmente, da *Uscite o attività o Azioni* che corrispondono alle operazioni richieste.

Un automa può essere descritto come un insieme A:

A = { I, U, S, f, g}
dove:
I è l'insieme degli ingressi;
U è l'insieme delle uscite;
S è l'insieme degli stati interni in cui può trovarsi;

f è la funzione che fa passare da uno stato al successivo, $S_{t+1} = f(S_t, i_t)$;
g è la funzione che determina il valore delle uscite, $U_t = g(S_t, i_t)$.

In pratica la descrizione del processo di funzionamento del sistema si traduce in *un Grafo a Stati e Transizioni,* in cui i *Nodi* rappresentano gli *stati* e gli *archi orientati* le *transizioni.* Accanto ad esse ci sono gli *Eventi (Ingressi)* che provocano le *Transizioni* e le eventuali *Uscite (Azioni)* che vengono intraprese..



cfr. http://odetocode.com/Articles/460.aspx

In altre parole una FSM consiste in un sistema per il calcolo delle USCITE e dello STATO FUTURO a partire dagli INGRESSI e dallo STATO PRESENTE:

- Uno **stato** rappresenta una situazione del sistema in quell'istante, ne dà cioè, la "fotografia" in quel momento. Nel diagramma mostrato (grafo orientato o , in gergo, pallogramma), abbiamo una macchina a stati con due stati: Acceso (Power on) e Spento (Power off). La macchina sarà sempre in uno di questi due Stati.
- Un **evento** è uno stimolo esterno. In figura, abbiamo solo un tipo di evento, il clic sul pulsante. La macchina a stati risponderà a questo evento in entrambi gli stati.
- Una **transizione** fa precipitare il sistema in un altro stato (ma potrebbe anche farlo rimanere nello stesso). Una transizione può avvenire solo in risposta a un evento.

Nel diagramma a stati mostrato sopra, l' evento "Button clic", nello stato "Power On" fa precipitare la macchina nello stato "Power off". Lo stesso evento in "Power Off" fa cambiare lo stato macchina in "Power On".

Implicita nel concetto di una transizione di stato è che una certa **azione** (cioè una esecuzione di codice) avrà luogo al momento della transizione. Nel nostro diagramma l'azione apre o chiude il circuito per il transito di energia elettrica.
Gli automi si prestano molto bene nelle soluzioni di problemi informatici perché il diagramma a stati può essere rappresentato secondo una "*tabella (o matrice) di transizione Stati/Eventi/Azioni"* del tipo:

Stato / Evento	**Evento 1**	**Evento 2**
Stato 1	Stato2 / Azione 1	Stato1 / -
Stato 2	Stato2 / Azione 2	Stato1 / Azione 3

Nel caso del nostro diagramma (esempio 0):

Stato / Evento	**"button click"**
Power on	"Power off" / "Open"
Power off	"Power on" / "Close"

Tradurre questa tabella in codice risulta agevole e permette la creazione di un "motore" software capace di elaborare gli eventi e produrre le azioni corrispondenti.

Esempio 1. Il succedersi delle stagioni con gli effetti su un albero:
Stati :

- Inverno
- Primavera,
- Autunno,
- Estate,

con la relativa successione.

Eventi :

- il 21 marzo è l'evento che fa transitare da Inverno a Primavera,
- il 21 giugno da Primavera a Esate e cosi' via…

Azioni:

- L'albero mette i fiori
- L'albero mette i frutti
- L'albero lascia cadere le foglie
- L'albero va "in letargo"

Non è difficile realizzare un grafo e la tabella delle transizioni.

Esempio 2 Gli Automi possono essere usati per descrivere una grande varietà di situazioni come nel seguente esempio in cui l'Automa è usato come riconoscimento e accettatore di sequenze

Automi riconoscitori

(http://terzablst.altervista.org/automiriconoscitori.html) Si tratta di automi particolari, aventi una o più entrate, ma sicuramente un'unica uscita. L'unica uscita è il segnale di riconoscimento che ci indicherà se quello che abbiamo in ingresso è qualcosa che viene riconosciuto oppure non viene riconosciuto.

0 = non riconosciuto
1 = riconosciuto

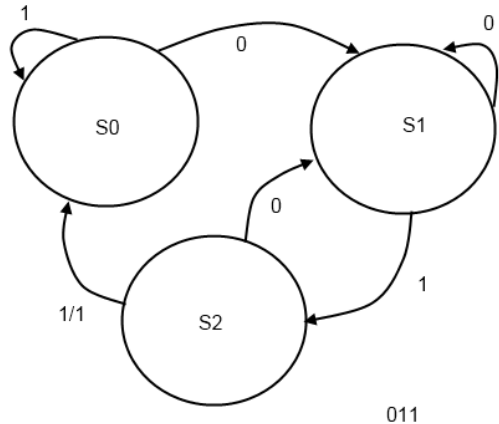
Gli automi riconoscitori vengono utilizzati ad esempio dai software "traduttori". I software "traduttori" sono molto importanti, infatti ci permettono di tradurre un programma scritto in un linguaggio di programmazione, vicino all'uomo, in un linguaggio binario vicino alla macchina.
In ambito informatico però,esistono altri ambiti in cui vengono utilizzati gli automi riconoscitori: pensiamo ad esempio a quando accediamo all'elaboratore con l'inserimento di una password. Pensiamo ai robot in grado di riconoscere comandi forniti dall'ambiente esterno.

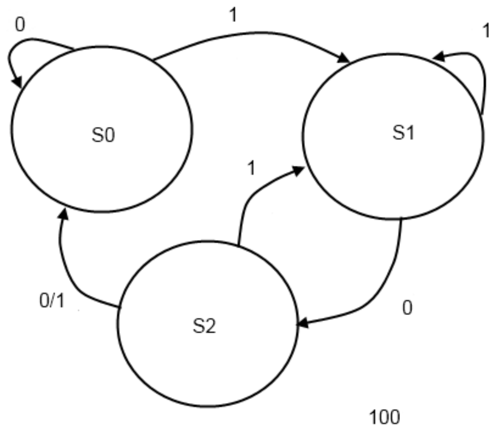
Come lavorano gli automi riconoscitori :

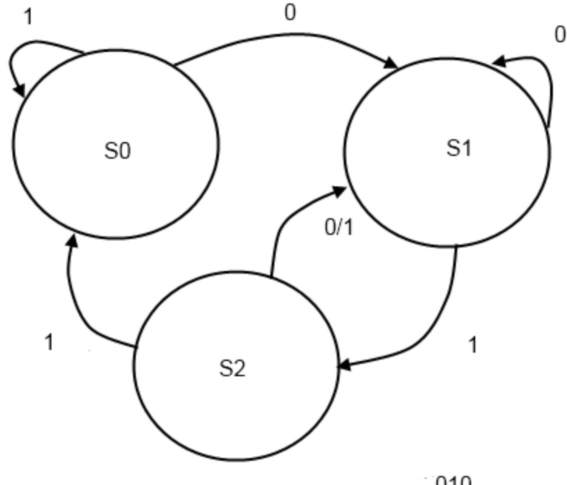
- Innanzitutto ricevono in ingresso un insieme di caratteri (uno alla volta).
- In ingresso possono avere: una stringa o un'infinità di caratteri. Per definizione la stringa (o parola) è un insieme finito di caratteri. Quando parlo di stringhe devo indicare l'alfabeto mediante il quale compongo la parola. Ad esempio aababbab e bbababb sono due stringhe sull'alfabeto composto dalle sole lettere {a,b}.
- Il grafo degli stati ci permette di capire se l'automa riconoscitore sta riconoscendo la stringa su un insieme finito o infinito di simboli numerici e/o alfanumericI.

Esrcizi: http://magistri.altervista.org/SISTEMI/quarta/Esercizi_sugli_Automi.pdf

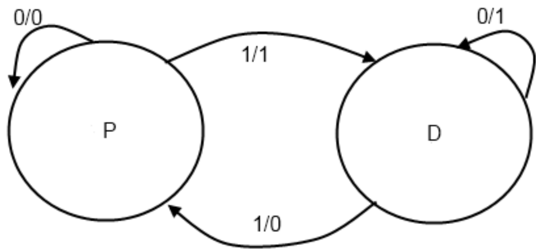
Esempio 2. 1 riconosce la sequenza e, solo in questo caso, emette 1 in uscita







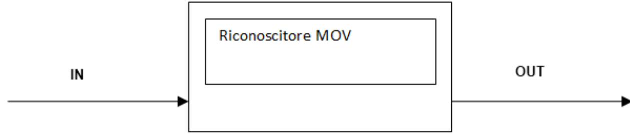
Esempio 2.2 Generatore di parità



ESEMPI0 2.3 :

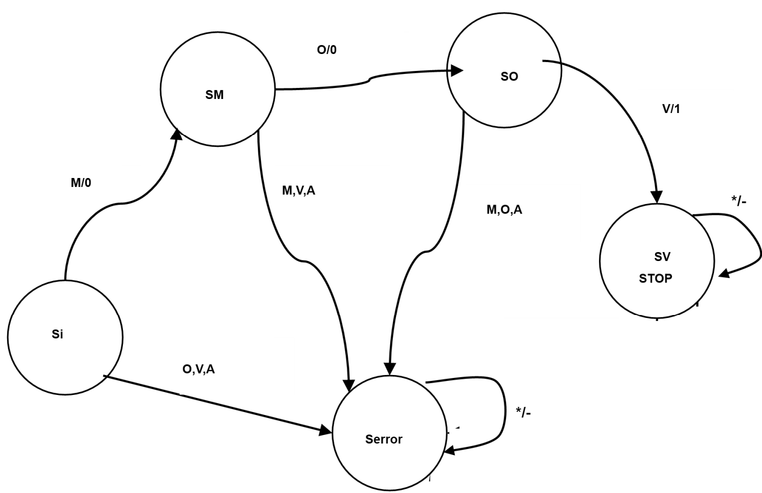
http://terzablst.altervista.org/automiriconoscitori.html

Dato l'alfabeto costituito solo dalle lettere M,O,V,A riconoscere la parola MOV.

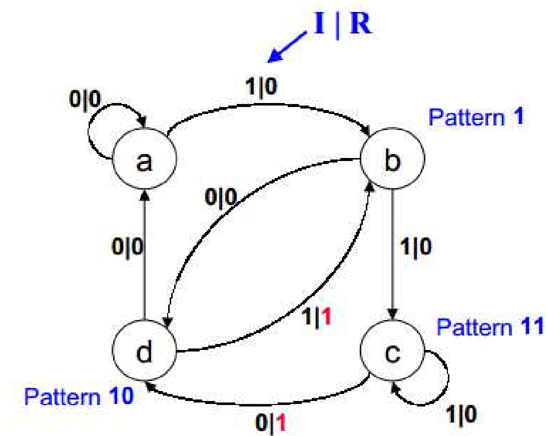


Alfabeto: { M,O,V,A }
OUT 0 = non riconosciuto, 1 = riconosciuto

s\E	M	O	V	A
Si	SM/0	Serr/0	Serr/0	Serr/0
SM	Serr/0	0/So	Serr/0	Serr/0
So	Serr/0	Serr/0	SV/1	Serr/0
Sv	Serr/0	Serr/0	Serr/0	Serr/0
Serr	Serr/0	Serr/0	Serr/0	Serr/0

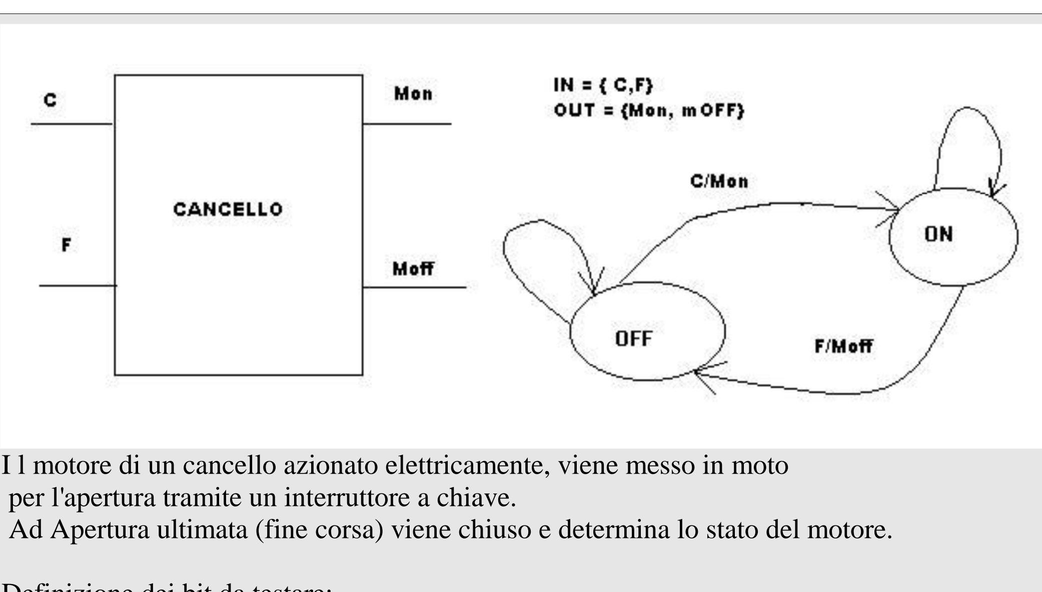


Es 2.4



http://www.dsi.unive.it/~arcb/AA01-02/SLIDES/12_ese_circ_seq2.pdf

Esempio . Reralizzazione di una macchina a stati applicata mediante Assembly 8086

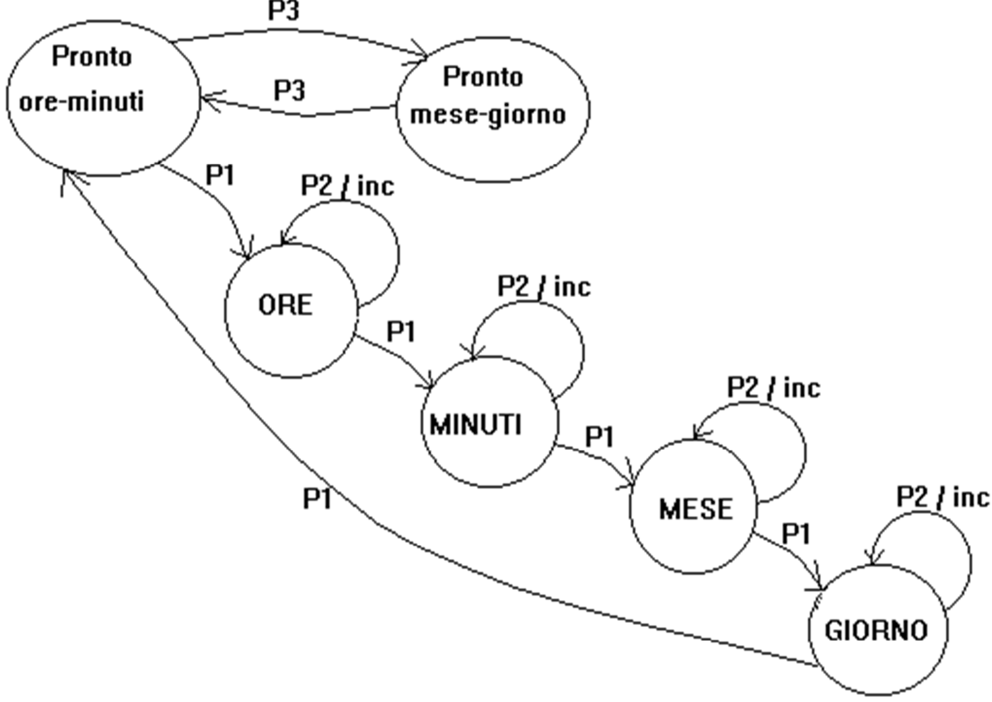



```
;I l motore di un cancello azionato elettricamente, viene messo in moto
; per l'apertura tramite un interruttore a chiave.
; Ad Apertura ultimata (fine corsa) viene chiuso e determina lo stato del motore.

;Definizione dei bit da testare:
; Porta (o varin0):
;       IN0 x x x x x x x x
;                         | |
;                         |  chiave = 1 on, 0 off
;                         fine corsa = 1 on, 0 off
; Porta  (o varout0) :
;       OUT0 x x x x x x x x
;                             |motore = 1 on, 0 off

;VINCOLI OGGETTIVI
;1-una chiave mette in moto il motore del cancello
;2-il motore è in funzione finchè non arriva all'interruttore di fine corsa

;VINCOLI SOGGETTIVI
;1. al lancio del programma il cancello si presuppone che sia chiuso
;2. il programma continua a funzionare in eterno

data segment
    ; add your data here!
    varin0  db 0
    varout0 db 0
ends

;DEFINE
IN0 EQU 00h
OUT0 EQU 01h

stack segment
    dw   128  dup(0)
ends

code segment
start:
; set segment registers:
    mov ax, data
    mov ds, ax
    mov es, ax

MOFF:      ;mov dx,IN0                          ; ingressi e uscite (non  usate in questo esempio)
           ;in al,dx
        mov al,varin0                           ;Controllo sulla porta per la chiave
        and al,00000001B
        jz MOFF
```

```
MON:
        or al, 00000001B                          ;Accensione motore
        mov varout0,al
        ;mov dx,OUT0
        ;out dx,al

NonFine:
        ;mov dx,IN0
        ;in al,dx
mov al,varin0
        and al,00000010B                          ;Aspetta lo switch del finecorsa
        cmp al, 00000010B
        jnz NonFine
        and al,11111110B                          ;spegni
        ;mov dx,OUT0
        ;out dx,al
        mov varout0,al                            ;il motore si spegne e ritorna da capo
        jmp MOFF

    ;------------

    mov ax, 4c00h ; exit to operating system.
    int 21h
ends
end start
```

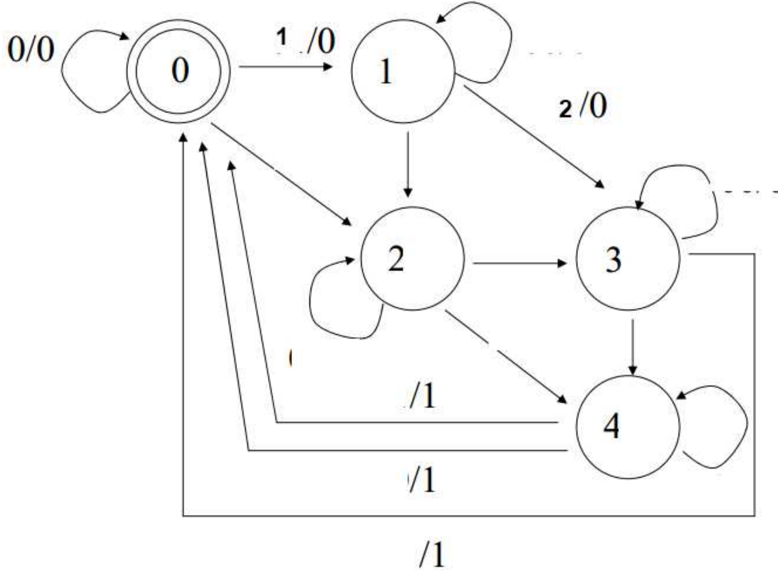
Esempio 4 - Regolazione di un orologio digitale

(http://www.dettori.info/dettori_old/automi.htm) L'orologio è munito di tre pulsanti la cui pressione chiameremo P1, P2, P3 che servono per la regolazione delle ore, minuti, mese, giorno e per il passaggio dalla modalità del display in cui vengono mostrati ore e minuti alla modalità in cui vengono mostrati mese e giorno. Il tasto P1 serve per passare dallo stato in cui il display mostra ore-minuti agli stati di regolazione, il tasto P2 serve per incrementare il valore attualmente presente sul display (inc=incrementa), il tasto P3 è un tasto bistabile tra le modalità del display ore-minuti e mese-giorno. Ecco il grafo:



Esercizi

Esercizio n.0 (parzialmente risolto)
La macchina accetta monete da 1 e 2 euro
I = {nessuna moneta, 1e, 2e}
• Fornisce il prodotto quando viene raggiunto l'importo di 5 euro (uscita a 1)
Convenzione: chiamiamo ogni stato della macchina con l'importo raggiunto



Esercizio n.1 Costruire la macchina a stati, sia in forma di grafo che di tabella di un riconoscitore: Vite,Dado e Vite,Rondella,Dado

Esercizio n.2 Costruire la macchina a stati, sia in forma di grafo che di tabella di un riconoscitore della sequenza 1101 che produce l' uscita 1 solo quando ultimata, altrimenti nessuna uscita.

Esercizio n.3 Costruire la macchina a stati, sia in forma di grafo che di tabella di un riconoscitore della parola 'MIO' che produce l' uscita 1 solo quando trovata, altrimenti nessuna uscita.

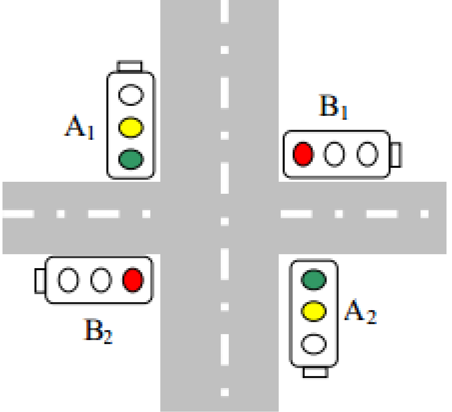
Esercizio n.4 (Generatore di parità) Costruire la macchina a stati, sia in forma di grafo che di tabella di un generatore di parità: data una sequenza di 0 e 1 in ingresso e dal simbolo terminale T, fare in modo che l' uscita finale sia1 o 0, rispettivamente se il numero di 1 rilevati sia dispari o pari.

Esercizio n.5 Costruire la macchina a stati, sia in forma di grafo che di tabella di un distributore di bevanda: l' unica bevanda presente è il latte. La bevanda costa 20 centesimi e la macchina accetta solo 10 o 20 centesimi oltre al tasto "annulla" che funziona solo dopo l' inserimento di 10 centesimi. Al raggiungimento di 20 centesimi la macchina serve il latte automaticamente.

Esercizio n.6 Costruire la macchina a stati, sia in forma di grafo che di tabella di un ascensore a 2 piani: l' ascensore ha i pulsanti Piano Terra (PT) e Piano 1 (P1). Le azioni sono "Sali" e "Scendi".

Esercizio n.7
Supponiamo di avere un sistema che si può trovare in uno stato appartenente ad un insieme finito di stati possibili.

Ex: Immaginiamo un incrocio tra due strade regolate tramite semafori.



Un semaforo può trovarsi in uno dei seguenti stati :
S = {rosso, verde , giallo, lampeggiante}
Supponiamo che il nostro sistema sia dotato di ingressi che condizionano l'evoluzione del sistema stesso. Supponiamo che l'insieme dei possibili valori di ingresso sia anch'esso finito e che il sistema, a determinati istanti di tempo, cambi di stato in funzione degli ingressi e dello stato corrente.

AUTOMI DETERMINISTICI SULL'ALFABETO E={a,b} Costruire gli automi finiti deterministici sull'alfabeto E={a,b} che accettano i seguenti linguaggi: 1. P: insieme delle parole che iniziano per ab e terminano per ba. 2. Q: insieme delle parole che contengono almeno una a e almeno una b. 3. R: insieme delle parole in cui ogni sottostringa aa è immediatamente seguita da almeno una b. 4. S: insieme delle parole palindrome di lunghezza 2. 5. T: insieme delle parole palindrome di lunghezza 4. 6. U: insieme delle parole palindrome.

UOMO, CAPRA, LUPO, CAVOLO Costruire un automa finito deterministico che rappresenta il seguente problema. Un uomo deve traghettare dalla sponda sinistra alla sponda destra di un fiume tre passeggeri: una capra, un lupo e un cavolo. L'uomo ha a disposizione una barca che gli consente di portare dall'altra parte un solo passeggero alla volta. Inoltre, egli non deve lasciare incustoditi sulla stessa sponda del fiume: (a) la capra con il cavolo; (b) il lupo con la capra. Per la costruzione dell'automa etichettare i vari stati con la seguente notazione:
• (C L V U) : Capra, Lupo, caVolo, Uomo sulla sponda sinistra; • (C L V U) : Capra, Lupo, caVolo sulla sponda sinistra; Uomo sulla sponda destra; • ecc. Gli eventi saranno: • u: l'uomo compie la traversata da solo; • uc: l'uomo compie la traversata con la capra; • ul: l'uomo compie la traversata con il lupo; • uv: l'uomo compie la traversata con il cavolo. Una volta costruito l'automa: • indicare lo stato finale (uomo e passeggeri sulla sponda destra); • indicare gli stati proibiti (capra-cavolo oppure lupo-capra incustoditi); • determinare se esiste una parola accettata che possa venir generata con una produzione che non passi per alcuno stato proibito e, in caso positivo, indicare tale produzione.