

# Corso Web MVC

## Introduzione

Emanuele Galli

[www.linkedin.com/in/egalli/](http://www.linkedin.com/in/egalli/)

# Informatica

- Informatique: information automatique
  - Trattamento automatico dell'informazione
- Computer Science
  - Studio dei computer e come usarli per risolvere problemi in maniera corretta ed efficiente

# Computer

- Processa informazioni
- Accetta input
- Genera output
- Programmabile
- Non è limitato a uno specifico tipo di problemi

# Hardware, Software, Firmware

- Hardware

- Componenti elettroniche usate nel computer
- Disco fisso, mouse, ...

tutte le parti fisiche del pc.  
Per processare informazioni  
lo unisco al software.

- Software

- Programma
  - Algoritmo scritto usando un linguaggio di programmazione
  - Codice utilizzabile dall'hardware
- Processo
  - Programma in esecuzione
- Word processor, editor, browser, ...

finché non lo avvio,  
rimane un programma.

- Firmware

- Programma integrato in componenti elettroniche del computer (ROM, EEPROM)
  - UEFI / BIOS: avvio del computer
  - Avvio e interfaccia tra componenti e computer

Read only (ROM): non lo  
posso toccare. Il resto, è  
modificabile.

entra subito in  
funzione quando  
accendo il computer

è un codice come il  
software, non facile da  
modificare. Dna della  
macchina, difficilmente  
attaccato da virus.

programma fisso, fa da  
collante tra hardware e  
software. Se dico  
programma intendo  
software

il mouse o un lettore dvd  
hanno il loro firmware.

# Sistema Operativo

è il primo software che entra in funzione dopo il bios.

- Insieme di programmi di base

il programma chiede dati al sistema operativo, che li recupera dal disco.

- Rende disponibile le risorse del computer

- All'utente finale mediante interfacce

- CLI (Command Line Interface) / GUI (Graphic User Interface)

- Agli applicativi

deve essere facile da usare per gli umani ma anche per gli applicativi

CLI: interfaccia carattere, scrivo i comandi dal prompt dei comandi con schermo nero (chiamata anche shell)

windows vs unix (linux e mac), hanno comandi diversi

- Facilità d'uso vs efficienza

- Gestione delle risorse:

- Sono presentate per mezzo di astrazioni

- File System

insieme di cartelle e file. Il file è l'unità minima di memorizzazione su memoria di massa e ha un nome. Il folder o directory (cartella) è qualcosa che può contenere file. File system sono tutte le icone che troviamo nel pc.

astrazione ad es. spostare il file da una cartella a un'altra per facilitare l'accesso. il file system è basato sul sistema albero, identificato con C:/ che è la radice ovvero disco fisso, da cui vedo i folder. I nodi sono quindi le cartelle.

- Ne controlla e coordina l'uso da parte dei programmi

- Semplifica la gestione del computer, lo sviluppo e l'uso dei programmi

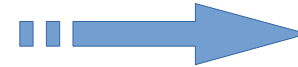
# Problem solving

il programmatore risolve i problemi. Cosa volevi ottenere? (input) E invece cos'hai ottenuto? (output) sono le specifiche.

- Definire chiaramente le **specifiche** del problema
  - Es: calcolo della radice quadrata. Input? Output?
  - Vanno eliminate le possibili ambiguità

stringa: un insieme di caratteri, altrimenti è un array

- Trovare un **algoritmo** che lo risolva



Floating point: numero decimale. Nel senso che non è un numero fisso, il punto può essere spostato: da 7.3 a .73

- Implementare correttamente la soluzione con un linguaggio di programmazione
- Eseguire il programma con l'input corretto, in modo da ottenere l'output corretto

Gigo: garbage in-garbage out. Bisogna gestire correttamente l'algoritmo con un linguaggio opportuno, per trasformarlo in un programma eseguibile dalla macchina.

# Algoritmo

paragonabile ad una ricetta: una serie di passi chiari che mi permettono di risolvere un problema (scrivendo il mio programma) dalla condizione di partenza.

il *sorting* è un algoritmo

- Sequenza di istruzioni che garantisce di dare il risultato di un certo problema
  - Ordinata, esecuzione sequenziale (con ripetizioni)
  - Operazioni ben definite ed effettivamente eseguibili
  - Completabile in tempo finito
- Definito in linguaggio umano ma artificiale
  - Non può contenere ambiguità
  - Deve essere traducibile in un linguaggio comprensibile dalla macchina

# Le basi dell'informatica

- Matematica

- L'algebra di George Boole ~1850

- Notazione binaria

- La macchina di Alan Turing ~1930

- Risposta all'Entscheidungsproblem (problema della decisione) posto da David Hilbert
    - Linguaggi di programmazione Turing-completi

- Ingegneria

- La macchina di John von Neumann ~1940

- Descrizione dell'architettura tuttora usata nei computer: Input, Output, Memoria, CPU

Kiss: keep it simple, stupid! Bisogna ridurre tutto all'operazione più semplice, poi si mette insieme. Boole introduce la notazione binaria lavorando solo con i valori 0,1 (true/false). è la base.



Turing ha dimostrato che non è possibile avere una macchina che decide cos'è giusto o sbagliato, ma nel frattempo ha avuto l'intuizione che è poi il nostro computer. I linguaggi di programmazione permettono di scrivere codici compatibili con la macchina di Turing, in quel caso sono giusti.

Linguaggio Turing completo: ha la possibilità di gestire un input, elaborarlo e dare un output a partire dalle istruzioni. Nel suo lavoro, la macchina deve poter prendere delle decisioni quindi ha bisogno di un IF (istruzione condizionale) e un ELSE; un'altra funzione dev'essere il poter fare LOOP mettendo for/while più la necessità di avere blocchi di istruzioni (block, cosa voglio fare sui dati) e variabili (dove metto i dati). Quando il dato è in memoria ci posso lavorare.

Idea ingegneristica, a partire da Turing. Ha messo in pratica la teoria matematica: lo schema parte dall'input, per passare dai CPU e generare un output. Tutto si basa sull'algebra di Boole.

dati: scariche elettriche che viaggiano nei circuiti, convertiti in zeri e uni per generare output.

Web MVC



# Algebra Booleana

- Due valori
  - false (0)
  - true (1)
- Tre operazioni fondamentali
  - AND (congiunzione)
  - OR (disgiunzione inclusiva)
  - NOT (negazione)

A	B	AND	OR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

A	NOT
0	1
1	0

# Linguaggi di programmazione

- Linguaggio macchina
  - È il linguaggio naturale di un dato computer
  - Ogni hardware può averne uno suo specifico
  - Istruzioni e dati sono espressi con sequenze di 0 e 1
  - Estremamente difficili per l'uso umano
- Linguaggi Assembly
  - Si usano abbreviazioni in inglese per le istruzioni
  - Più comprensibile agli umani, incomprensibile alle macchine
  - Appositi programmi (assembler) li convertono in linguaggio macchina

ogni macchina ha il suo linguaggio macchina. il cuore della macchina è il CPU, e in sua funzione cambia il linguaggio.

Da quel linguaggio si passa all'Assembly, più comprensibile, ma ancora di basso livello. es: comando add \_ per sommare, invece che una sequenza di 0 e 1. Il primo programma che serve è quello che traduce l'assembly in linguaggio macchina. passaggio: scrivo le istruzioni in assembly, salvo il file, lo butto nell'assembler che lo trasferisce alla cpu in linguaggio macchina e mi ridarà il risultato.

*eseguitibile* è una sequenza di 0 e 1

è il concetto logico con cui esprimo al programma il dato su cui voglio lavorare. A seconda di come definisco le variabili, definisco i linguaggi. Le architetture lavorano sempre a 64 bit, cioè i registri hanno questa dimensione e ciò descrive anche l'ampiezza del BUS. Un bit è l'unità minima di memorizzazione. *8 bit è un byte (il numero quindi va da 0 a 255)*. Userò come base il 2, perché i miei valori sono solo 2 e come esponente i numeri entro i quali posso lavorare (esponenziali). In un 64 bit infatti farò 2 alla 64esima -1 perché devo contare anche lo zero. In questo modo avrò tutte le combinazioni possibili (00.01.1-0.1-1 e via dicendo)

# Variabile

Se ho un numero negativo con un 64 bit, metto all'inizio della stringa il numero 1 invece dello 0 e in base il massimo sarà 2 alla 63esima.

la variabile contiene l'informazione, cioè il valore.

se vedo il segno = è un'assegnazione: metti il valore di destra in quello di sinistra, es. `c=a+b`

- **Locazione di memoria** associata a un nome, contiene un valore
- Costante: non può essere modificata dopo la sua inizializzazione
- Una singola locazione di memoria può essere associata a diverse variabili (alias)
- Supporto a tipi di variabili da linguaggi di:
  - “basso livello” → legati all'architettura della macchina
  - “alto livello” → tipi complessi
  - script → runtime

non abbiamo controllo sulla variabile nello script.

a basso livello: se ho 1 1 1 vuol dire 7 (perché faccio 2 alla 0, che fa 1, poi 2 alla 1 che fa 2 e 2 alla 2 che fa 4: totale 7). il vantaggio è che è un linguaggio aderente alla macchina.  
ad alto livello: si introduce la *semantica*, quali sono le variabili? es. `int=a` vuol dire che è un intero, quindi un numero. es. `2 char b=x`, `char` sta per carattere. In ogni linguaggio di programmazione c'è una tabella alla quale far riferimento quando trovo il numero anticipato dalla variabile `char`, ad esempio il 25 nella tabella sta per lo spazio bianco. UTF-8 è la tabella più usata in html, in Java la UTF-16 (8 e 16 bit)

CPU: central process unit, che ora si chiama Core, la parte del computer che fa tutti i conti. Riceve gli ordini e le esegue: per farlo, devo avere accesso ai dati, che si trovano in memoria (RAM: random access memory) quindi l'accesso è diretto, tramite un BUS che lo sposta nel REGISTRO. Più ram abbiamo e più si possono caricare programmi pesanti. La cache è memoria molto vicina alla cpu e quindi raggiungibile più velocemente, ma è anche più costosa. La cpu quando fa le sue operazioni lavora sui registri, parti su cui devo caricare i miei dati. Tutto torna nella RAM, che però tiene i dati momentaneamente, se non c'è corrente spariscono: se voglio che rimangano, li metto su un disco fisso o su una chiavetta, cioè una memoria di massa.

è una variabile che contiene più valori dello stesso tipo, e si indica con una S finale ad es. `var temperatures= [12.1,18.7..]` mi racchiude tutte le temperature possibili in anni di osservazione, invece che scrivere triloni di variabili. Se lavoro a 64 bit metto in ogni cella da 64 bit gli elementi (12.1 in una cella, e così via). L'array lavora su matrici. Se voglio accedere ad un dato specifico, ne devo indicare la posizione in un indice su `temperatures` (in questo caso) che metterò in parentesi quadra, ad esempio se mi serve il 18.7 individuo la posizione e scrivo `temperatures [1]` perché si parte sempre da 0.

# Array

diverso da *stringa*, vedi pag.6

- Struttura dati comune a molti linguaggi di programmazione
- Basata sul concetto matematico di vettore, nel senso di matrice monodimensionale
- Collezione di elementi (dello stesso tipo) identificati da un indice
  - Il primo elemento ha indice 0 in alcuni linguaggi, 1 in altri (e anche n in altri ancora)
- Gli elementi sono allocati in un blocco contiguo di memoria, il che permette accesso immediato via indice ai suoi elementi

# Linguaggi di alto livello

- Molto più comprensibili degli assembly
- Termini inglesi e notazioni matematiche
- Possono essere espressi in forma
  - **imperativa**: si indica cosa deve fare la macchina
  - **dichiarativa**: si indica quale risultato si vuole ottenere
- A seconda di come avviene l'esecuzione si parla di linguaggi
  - **compilati**: conversione del codice in linguaggio macchina, ottenendo un programma eseguibile
  - **interpretati**: il codice viene eseguito da appositi programmi

imperativo: una sequenza di ordini dati alla cpu.  
dichiarativo: html è un linguaggio dichiarativo, non di programmazione.  
dico cosa voglio ottenere, non importa il *come*

il compilatore, converte il codice comprensibile dagli umani (CODICE SORGENTE, source code) in una sequenza di 0 e 1 comprensibile dalla macchina (cpu e sistema operativo), es. windows a 64 bit. Quello che ne esce sarà un file.exe, eseguibile, che posso avviare.

il programma x.js invece (java script) va direttamente sul browser, che fa da interprete, ed è quindi visibile a tutti, che sia windows, linux, mac ecc. Infatti in questo caso non ho bisogno di un compilatore né di avviare niente. Il codice è quindi

# Istruzioni

- Operazioni **sequenziali**

- Chiedono al computer di eseguire un compito ben definito, poi si passa all'operazione successiva

i valori booleani servono a decidere come operare.  
Nella variabile c metti a per b. sempre in c metti il valore di c - 2.  
Faccio una serie di operazioni, è quindi sequenziale. La cpu quindi le esegue una dietro l'altra, nell'ordine da me stabilito.

- Operazioni **condizionali**

- Si valuta una condizione, il risultato determina quale operazione seguente verrà eseguita

se ( c>0) allora C=c+2, altrimenti c=diverso da 0. Il primo controllo è un valore booleano perché può essere solo vero o falso che c>0, quindi a seconda del risultato ho una operazione seguente. IF ... THEN, ELSE...

- Operazioni **iterative**

- Richiede di ripetere un blocco di operazioni finché non si verifica una certa condizione – se ciò non accade: loop infinito

Es: se voglio moltiplicare i dati di un array (es 5,3,1,7) ottenendo il loro doppio scriverò FOR (each element in a) el= el per 2 quindi non mi fermo fino a che non ho ottenuto la mia condizione --> LOOP

I linguaggi di programmazione devono supportare: variabili, condizionali, loop, blocco che sono fondamentali. I programmi si differenziano per questo, per quello che riescono a supportare.

# Flow chart vs Pseudo codice

mi raccontano come deve essere fatto un programma, me lo consegna il cliente.

- Diagrammi a blocchi – flow chart
  - L'algoritmo viene rappresentato con un grafo orientato dove i nodi sono le istruzioni
  - Inizio e fine con ellissi
  - Rettangoli per le operazioni sequenziali (o blocchi)
  - Esagoni o rombi per condizioni
- Pseudo codice
  - L'algoritmo viene descritto usando l'approssimazione un linguaggio ad alto livello, si trascurano i dettagli, ci si focalizza sulla logica da implementare

è fatto come gli schemi riassuntivi, con diversi passaggi, tutto inquadrato, con due ellissi una a inizio e una a fine.

in modo da capirsi senza usare il linguaggio specifico, altrimenti il cliente non avrebbe bisogno del programmatore se già conoscesse il linguaggio.

# Complessità degli algoritmi

- “O grande”, limite superiore della funzione asintotica

"O grande" : voglio una funzione che sia sempre superiore alla mia funzione, e possibilmente vi si avvicina sempre più. Voglio vedere quanti millisecondi (tempo) o quanta memoria occupa (spazio) il mio algoritmo, almeno approssimativamente (=costo). Tipicamente le due variabili sono legate, aumenta uno diminuisce l'altro e viceversa.

- Costante  $O(1)$  if

- Logaritmica  $O(\log n)$  raddoppio o divido lavorando sull'albero, quindi essendo esponenziale la complessità è logaritmica

- Lineare  $O(n)$  quando c'è for ... di n

- Linearitmico  $O(n \log n)$  sorting

- Quadratica  $O(n^2)$  – Polinomiale  $O(n^c)$  un for dentro un altro

- Esponenziale  $O(c^n)$

- Fattoriale  $O(n!)$

- Tempo e spazio

- Caso migliore, peggiore, medio

il più interessante è il caso medio. Sapere il caso migliore non mi interessa, perché è una cosa che si verifica una tantum. Se possibile, evitiamo il caso peggiore.



# Algoritmi di ordinamento

- Applicazione di una relazione d'ordine a una lista di dati
  - Naturale → crescente (alfabetico, numerico)
- Utile per migliorare
  - l'efficienza di altri algoritmi
  - La leggibilità (per gli umani) dei dati
- Complessità temporale
  - $O(n^2)$ : algoritmi naive
  - $O(n \log n)$ : dimostrato ottimale per algoritmi basati su confronto
  - $O(n)$ : casi o uso di tecniche particolari

normalmente non riusciamo a scendere  
sotto a questo logaritmo

# Ingegneria del software

- Approccio sistematico alla creazione del software scopo
  - Struttura, documentazione, milestones, comunicazione e interazione tra partecipanti scadenza
- Analisi dei requisiti
  - Formalizzazione dell'idea di partenza, analisi costi e usabilità del prodotto atteso l'unità più grande dell'algoritmo
- Progettazione
  - Struttura complessiva del codice, definizione architetturale
  - Progetto di dettaglio, più vicino alla codifica ma usando pseudo codice o flow chart MVC: modern view controller, un modo con cui noi strutturiamo il codice. devo capire se ho bisogno di macchine, licenze, software che ci servono...
- Sviluppo tradurre le specifiche in codice
  - Scrittura effettiva del codice, e verifica del suo funzionamento via **unit test** preparano il materiale per i programmatori
- Manutenzione
  - Modifica dei requisiti esistenti, bug fixing



# Unit Test

- Verificano la correttezza di una singola “unità” di codice
  - Mostrano che i requisiti sono rispettati
- Verifica
  - Casi base (positivi e negativi)
  - Casi limite
- Ci si aspetta che siano
  - Ripetibili: non ci devono essere variazioni nei risultati
  - Semplici: facile comprensione ed esecuzione
  - E che offrano una elevata copertura del codice

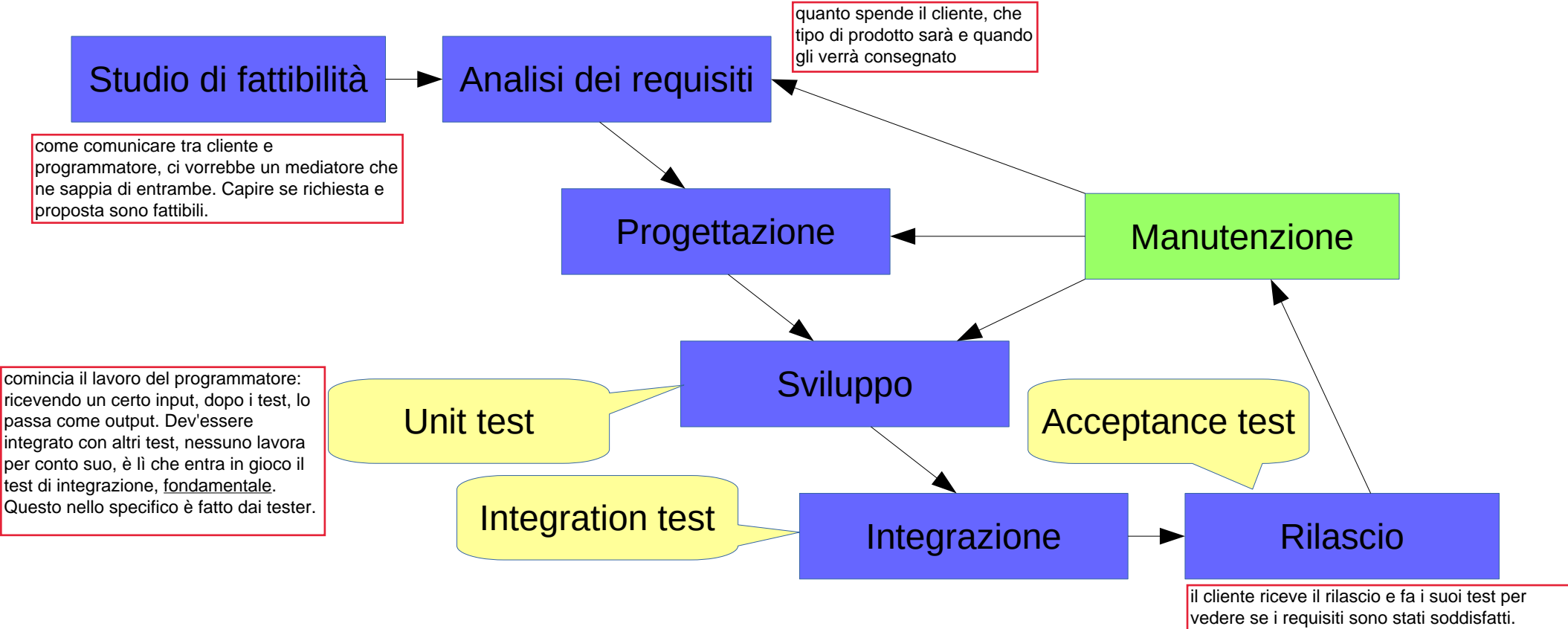
TDD: prima di scrivere la funzione, devo chiedermi cosa potrebbe andare storto, immaginando i casi più strani e scrivendone i test. Devo accertarmi che il mio codice non si comporti in modo strano

tutti i possibili flussi di esecuzione devono essere coperti.

Per lo sviluppo del codice: waterfall e agile

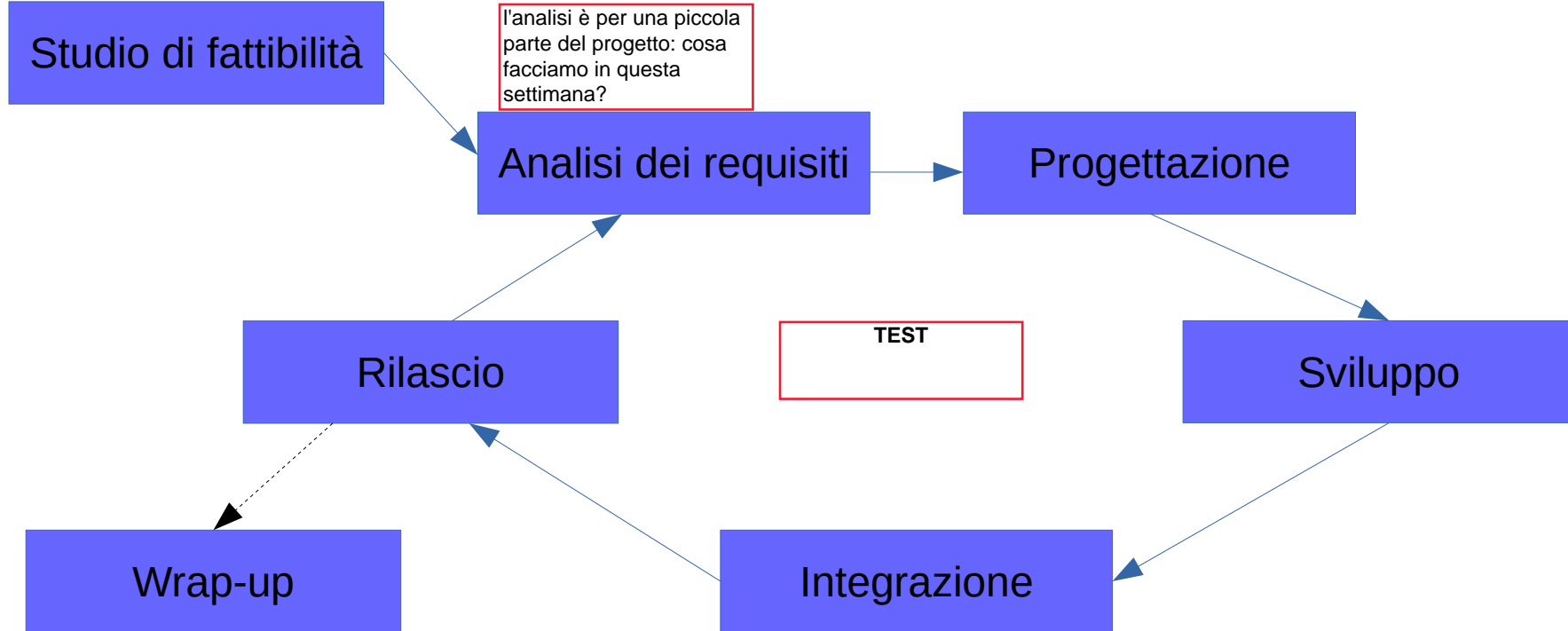
# Modello a cascata (waterfall)

grandi aziende, metodo molto rigoroso



# Modello agile

non piace perché non si riesce a dare una stima di costo e tempo, le start up prediligono questo modello.



l'app è finita, non la tocchiamo più. (solo in teoria però)

# Software Developer

- Front End Developer
  - Pagine web, interazione con l'utente
    - HTML, CSS, JavaScript
    - User Experience (UX)
- Back End Developer
  - Logica applicativa
    - Java, C/C++, Python, JavaScript, SQL, ...
    - JavaEE, Spring, Node, DBMS, ...
- Full Stack Developer
  - Sintesi delle due figure precedenti

quello che il cliente non vede, è a livello della nostra macchina, o meglio, sul cloud