

# Introduzione alla programmazione

- Informatique: information automatique
  - Trattamento automatico dell'informazione
- Computer Science
  - Studio dei computer e di come usarli per risolvere problemi

Emanuele Galli – [www.linkedin.com/in/egalli/](https://www.linkedin.com/in/egalli/)

# 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, CPU, Memoria principale (RAM), Memoria di massa (HD, SSD, CD, ...)

# 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

# Computer

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

# Hardware – Software

- Hardware
  - Componenti elettroniche usate nel computer
  - Disco fisso, mouse, ...
- Software
  - Programma
    - Algoritmo scritto usando un linguaggio di programmazione
  - Processo
    - Una istanza di un programma in esecuzione
  - Word processor, editor, browser, ...
- Firmware
  - Programma integrato in componenti elettroniche del computer (ROM, EEPROM, Flash)
    - UEFI / BIOS: avvio del computer
    - Avvio componenti e interfaccia con il computer

# Sistema Operativo

- Insieme di programmi di base
  - Rende disponibile le risorse del computer
    - All'utente finale mediante interfacce
      - **CLI** (Command Line Interface) / **GUI** (Graphic User Interface)
    - Agli applicativi
  - Facilità d'uso vs efficienza
- Gestione delle risorse:
  - Sono presentate per mezzo di astrazioni
    - File System, ...
  - Ne controlla e coordina l'uso da parte dei programmi
- Semplifica la gestione del computer, lo sviluppo e l'uso dei programmi

# Internet

- Estensione di Arpanet
- Rete di comunicazione tra macchine basata su TCP/IP (TCP vs UDP)
- La si può pensare come un grafo
  - I nodi sono periferiche identificate da indirizzo IP
    - DNS: Domain Name System
  - Gli archi sono le connessioni
- Servizi in ascolto su una porta usano protocolli a più alto livello
  - **HTTP** → World Wide Web
  - IMAP, Telnet, FTP, ...

# Problem solving

- Definizione delle **specifiche** del problema
  - Es: calcolo della radice quadrata.
- **Analisi** del problema
  - Diverso da singola istanza di un problema (es: radice quadrata di 25)
  - Quali input sono attesi? Che output va generato?
  - Eliminazione delle possibili ambiguità
- Progettazione di un **algoritmo** che lo risolva
- Implementazione della soluzione
  - con un particolare linguaggio di programmazione
- Esecuzione del programma con un dato input → output (GIGO)

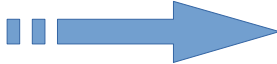




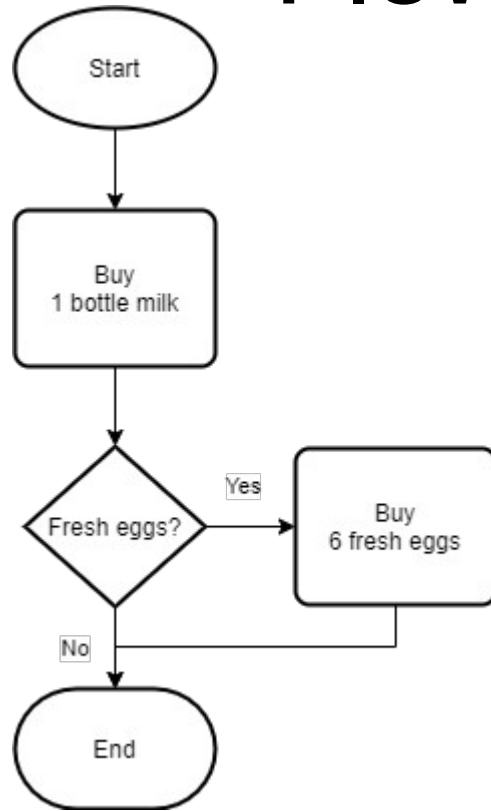
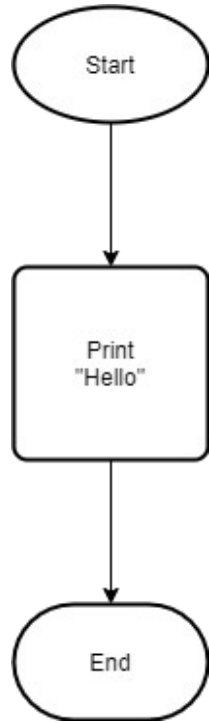
# Algoritmo

- Deve il suo nome al matematico persiano Al-Khwarizmi (~800)
- Sequenza di istruzioni che fornisce il risultato di un certo problema
  - Ordinata, esecuzione sequenziale (con ripetizioni)
  - Operazioni ben definite ed effettivamente eseguibili
  - Completabile in tempo finito (e ragionevole)
- È corretto solo se genera il risultato atteso per ogni possibile input
- Definito in linguaggio umano ma **artificiale**
  - Non deve contenere ambiguità
  - Deve essere traducibile in un linguaggio comprensibile dalla macchina

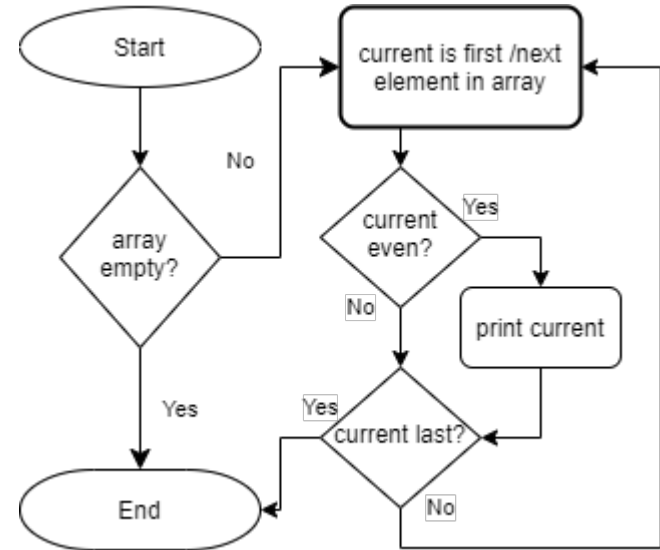
# Flow chart vs Pseudo codice

- Diagrammi a blocchi – flow chart
  - L'algoritmo viene rappresentato con un grafo orientato dove i nodi sono le istruzioni
  - Nell'implementazione più basica:
    - Inizio e fine con ellissi
    - Rettangoli per le operazioni sequenziali (o blocchi)
    - Esagoni o rombi per condizioni
  - Un tool: draw.io <https://www.diagrams.net/>
    - <https://github.com/jgraph/drawio-desktop/releases/>
- 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

# Flow chart



Print even numbers in an array



# Pseudo codice

```
print "Hello"
```

```
buy 1 bottle milk
```

```
if fresh eggs:  
    buy 6 fresh eggs
```

```
// print even numbers in an array
```

```
for each element in array:  
    if current element is even:  
        print current element
```

# Linguaggi di programmazione

- Linguaggio macchina
  - È il linguaggio proprio 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 macchina
  - Più comprensibile agli umani, incomprensibile alle macchine
  - Appositi programmi (assembler) li convertono in linguaggio macchina

# Linguaggi di alto livello

- Molto più comprensibili degli assembly
- Termini inglesi e notazioni matematiche
- Possono usare uno (o più) dei seguenti paradigmi:
  - **imperativo**: cosa deve fare la macchina (Von Neumann), un passo alla volta
    - programmazione strutturata → procedurale / orientata agli oggetti
  - **dichiarativo**: quale risultato si vuole ottenere
    - funzionale
- A seconda di come esegue il programma si parla di linguaggi
  - **compilati**: da codice sorgente a programma eseguibile via compilatore
  - **interpretati**: il codice sorgente viene eseguito dall'interprete

# Programmazione Strutturata

- *Goto statement considered harmful*, Edsger Dijkstra, 1968
- Teorema di Böhm-Jacopini, 1966
  - Ogni algoritmo può essere definito usando esclusivamente
    - Sequenze (**blocchi**) di istruzioni
    - **Decisioni** tra alternative di esecuzione: scelta condizionata dell'istruzione da eseguire
    - **Iterazioni** / cicli di esecuzione: ripetizione condizionata di un blocco di istruzioni
      - attenzione ai loop infiniti!
- Un linguaggio di programmazione è Turing completo se gestisce
  - Istruzioni “semplici” – input, output, assegnamento, ...
  - Istruzioni definite da Böhm-Jacopini

# Programmazione Procedurale

- Il problema viene diviso in blocchi (procedure)
- Ogni procedura
  - Ha un compito ben definito
  - Agisce come se fosse un sottoprogramma (subroutine)
  - Può essere riutilizzata in diversi programmi
- Le procedure interagiscono tra loro
  - Passandosi dati (parametri, valore di ritorno)
  - Operando su dati condivisi



# Programmazione Orientata agli Oggetti

- Al centro sono i dati e la loro interazione
- Definizione della struttura degli oggetti (classe)
  - Dati (proprietà) e altri dettagli interni di un oggetto
    - Le proprietà determinano lo **stato** corrente dell'oggetto
  - Funzionalità accessibili esternamente (metodi)
    - I metodi richiamabili su un oggetto rappresentano il suo **comportamento** / interfaccia
- Un programma è un insieme di oggetti
  - che interagiscono tra loro per mezzo dei metodi
- È un paradigma che permette un naturale incapsulamento dei dati

# Programmazione Funzionale

- Uso di funzioni nel senso matematico del termine (“pure”)
  - Non hanno uno stato e operano su valori immutabili – e dunque sono facilmente componibili e thread-safe non avendo effetti collaterali
  - Il flusso di esecuzione è determinato dall’invocazione di funzioni a partire da collezione di dati
  - È comune l’uso di chiamate ricorsive
- Le funzioni sono valori a tutti gli effetti, si può
  - passarle come parametro
  - ottenerle come risultato dall’invocazione di una funzione
- Facilita lo sviluppo di applicazioni che prevedono l’esecuzione in parallelo

# Variabile

- Locazione di memoria associata a un nome, contiene un valore
  - È buona norma (quando non è un obbligo) inizializzare le variabili contestualmente alla dichiarazione (operazione detta **definizione**) con una espressione “right hand side”
    - contenuto di un'altra variabile
    - risultato dell'esecuzione di una espressione (*statement*)
    - un valore letterale
- 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

# Strutture dati

- **Array** / vettore → concetto matematico, matrice monodimensionale
  - Elementi (omogenei) identificati da un indice
  - Allocati in un blocco contiguo di memoria, accesso diretto via indice ai suoi elementi
  - Il primo elemento ha indice 0 (C/C++, Java, Python, ...), 1 (MATLAB, R, Julia, ...), ...
- **Struttura** / record → elementi (eterogenei) identificati da un nome
- **Lista** → nodi collegati (semplice o doppia), ha una testa (e una coda)
- **Albero** → nodi collegati, ha una radice, ogni nodo può avere n figli
  - BST: Binary Sorted Tree
- **Hash table** → il valore è acceduto tramite il suo “hash”
- **Grafo** → simile all'albero, ma
  - non ha radice, gli archi possono avere un peso ed essere orientati

# Funzione / Procedura / Metodo

- **Blocco di codice** identificato da
  - Un nome
  - Una lista di parametri (input)
  - Il tipo del valore ritornato (output)
- Una procedura è una funzione che non ritorna alcun risultato
- Si può 'invocare' (o 'chiamare') una funzione da altre parti del codice
  - I valori passati come parametri sono detti 'argomenti'
- OOP: le *funzioni* sono 'libere', i *metodi* sono relativi a classi / oggetti

# UML

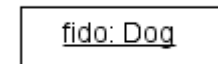
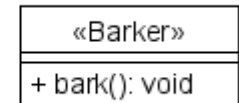
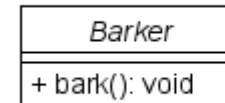
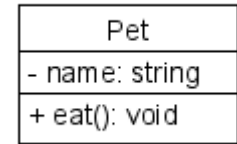
- Unified Modeling Language @ Rational ~ metà 1990
  - Grady Booch, James Rumbaugh e Ivar Jacobson
- Standard OMG dal 1997, ISO dal 2005
- Riferito alla programmazione Object Oriented
  - ma indipendente dal linguaggio di implementazione
- Non completamente formale, basato su diagrammi
- Usato per identificare in un sistema
  - le entità che lo compongono, loro caratteristiche e relazioni

# Diagrammi

- Modellano il sistema secondo diversi punti di vista
  - Strutturali: caratteristiche statiche del sistema
  - Comportamentali: interazioni tra componenti nel sistema
  - Architetture: descrizione della struttura del sistema
- Class, Interface, Object
- Use Case: comportamento del sistema in uno specifico caso
- State: cambiamento dello stato di un oggetto nel corso della sua vita
- Sequence: interazione tra gli oggetti
- Activity: sequenza di attività all'interno di un Use Case
- Tra gli altri
  - Communication: cooperazione tra elementi per obiettivo comune
  - Component: struttura di sistemi complessi
  - Deployment: configurazione del sistema a runtime

# Class

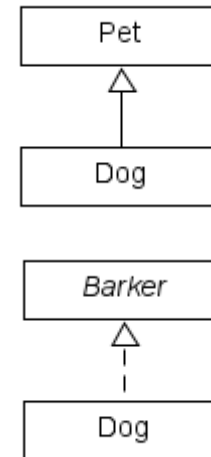
- Una classe è rappresentata da un rettangolo
  - Normalmente diviso in tre parti (+ 1)
  - **Nome**, può essere prefissato dal nome del package (separatore ::)
  - **Proprietà**, visibilità, nome: tipo = inizializzazione
  - **Metodi**, visibilità, nome(parametri): return type
  - Note aggiuntive
- Un modo per indicare la visibilità dei membri di classe
  - **+** pubblico, **~** package, **#** protetto, **-** privato
- Hanno una rappresentazione simile a quella della classe
  - **Interfaccia**, nome in corsivo (oppure <<InterfaceName>>)
    - Nota: anche i metodi astratti nelle classi sono resi in corsivo
  - **Oggetto**, nomi sottolineati





# Gerarchie

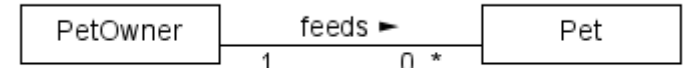
- Indicate con frecce da figlia a madre
- **Generalizzazione**
  - Ereditarietà, “is a”
  - Freccia linea intera con triangolo a punta vuota
- **Realizzazione**
  - Implementazione di interfaccia
  - Freccia tratteggiata con triangolo a punta vuota



# Associazioni

- **Associazione** generica tra classi

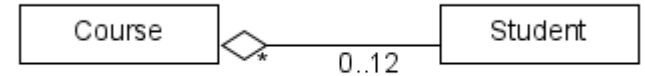
- Una riga connette le classi
- È possibile indicare come note
  - la molteplicità (1..\*) e il comportamento di un oggetto nell'associazione



- Oggetti di una classe che **posseggono** oggetti di un'altra classe

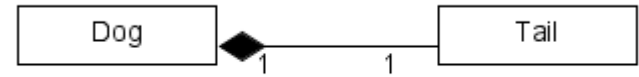
- **Aggregazione**

- Gli aggregati possono esistere indipendentemente
- Freccia con diamante vuoto



- **Composizione**

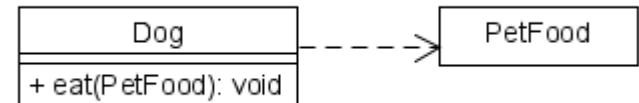
- Dipendenza dall'esistenza del proprietario
- Freccia con diamante pieno



- Oggetti di una classe in **relazione debole** con oggetti di un'altra classe

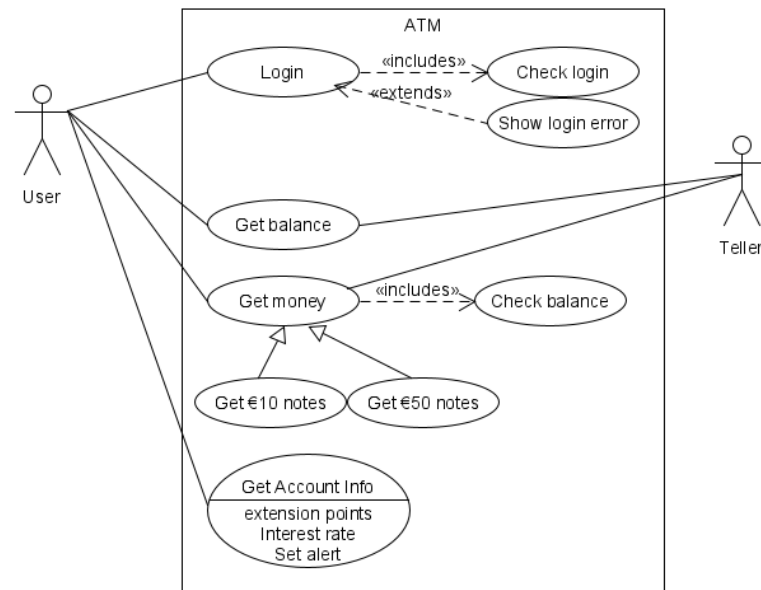
- **Dipendenza**

- Parametro o variabile locale di un metodo
- Freccia tratteggiata



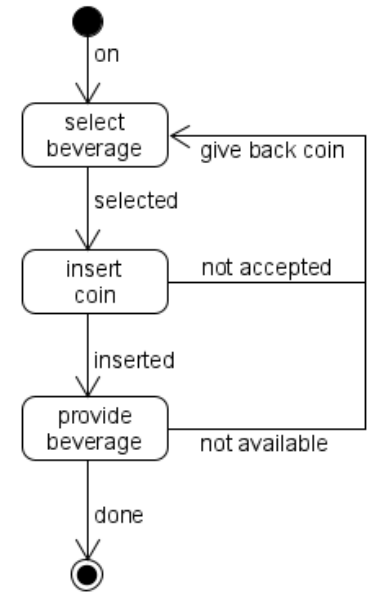
# Use Case

- System: cosa si vuole ottenere (web app, componente software, ...)
- Actor: omettino stilizzato, interagisce col sistema
  - **Primary** Actor: inizia l'azione (a sinistra)
  - **Secondary** Actor: reagisce alle azioni del Primary Actor (a destra)
- Use Case: azione all'interno di System
  - Possibili **Extension Points** per UC complessi
- Relationship: linea tra Actor coinvolti e UC
  - **<<includes>>** su freccia tratteggiata
    - Base UC richiede un included UC
  - **<<extends>>** su freccia tratteggiata
    - Base UC può essere esteso in casi specifici
  - Generalization
    - Gerarchia di UC (o Actor)



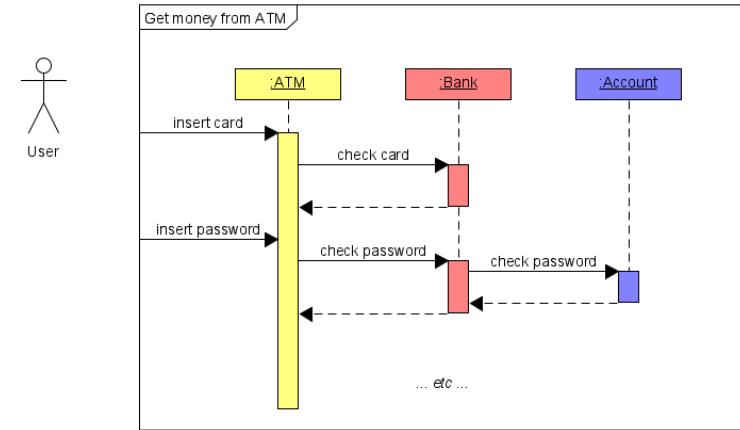
# State (Machine)

- Comportamento di un oggetto in seguito a eventi
- Mostra le transizioni tra stati (freccia)
- Stati
  - Iniziale (pallino nero)
  - Intermedio (rettangolo)
  - Finale (pallino nero dentro un cerchio)



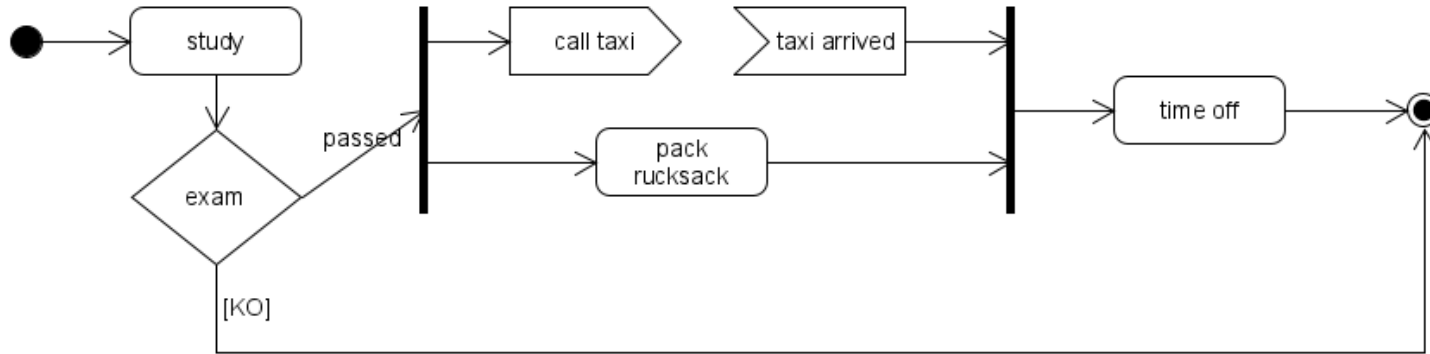
# Sequence

- Interazione tra classi / oggetti nell'ordine in cui avvengono
  - Sequenza di eventi
- Actor
  - Interagisce col sistema
- Message e return message
- Activation box
  - tempo necessario per un task



# Activity

- Flow chart più rigorizzato, state diagram più dettagliato
  - Fork-join per multithreading indicato con una barra
  - Box specifici per send/receive di messaggi asincroni

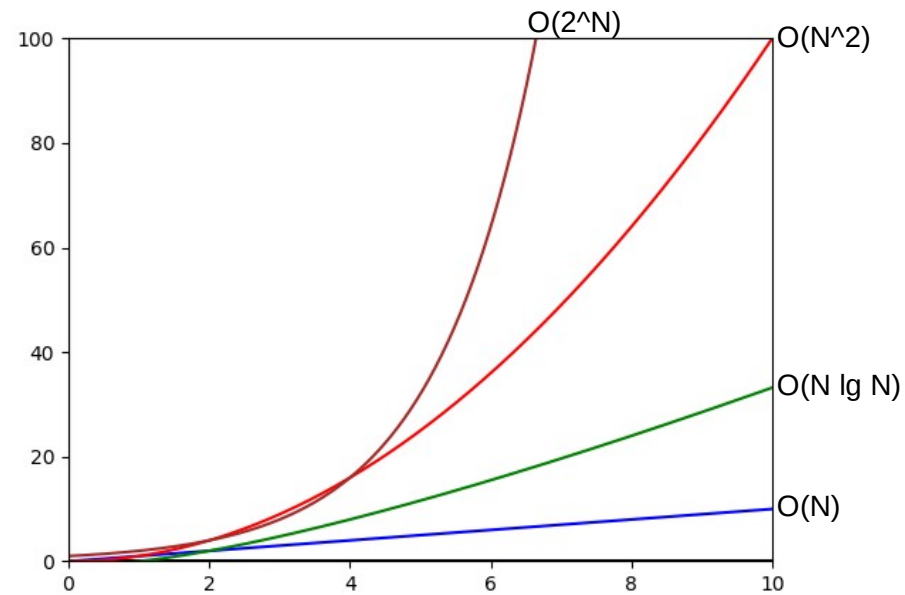
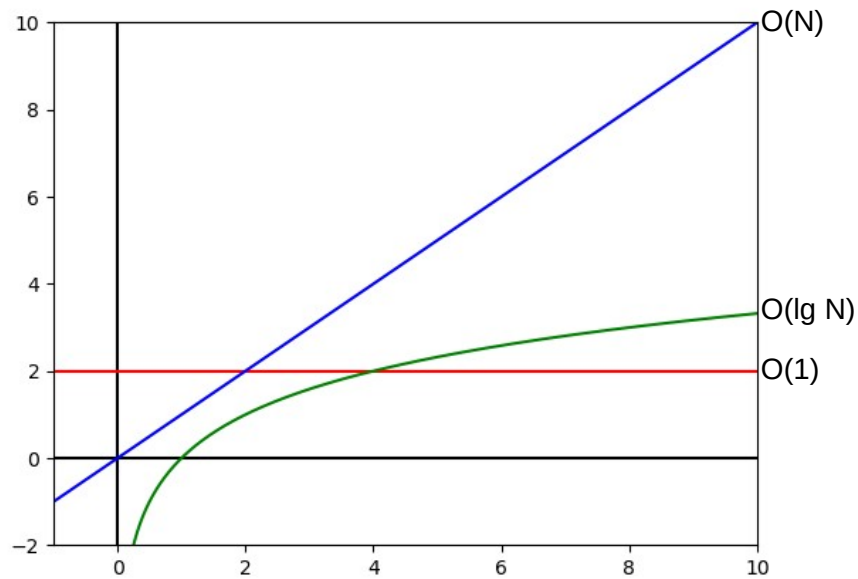


# Complessità degli algoritmi

- Caso migliore, peggiore, medio in tempo e spazio
- “O grande”, limite superiore della funzione asintotica
  - Costante  $O(1)$
  - Logaritmica  $O(\log n)$
  - Lineare  $O(n)$
  - Linearitmico  $O(n \log n)$
  - Quadratica  $O(n^2)$  – Polinomiale  $O(n^c)$
  - Esponenziale  $O(c^n)$
  - Fattoriale  $O(n!)$



# Complessità degli algoritmi





# Algoritmi di ordinamento

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

# Tre algoritmi $O(n^2)$

- Bubble sort
  - Confronta ogni coppia di elementi adiacenti, se non sono in ordine, li si scambia. Termina quando non si trovano elementi fuori ordine
- Selection sort
  - Per ogni posizione si seleziona il valore minimo da quel punto in poi
  - Swap tra elemento corrente e valore minimo
- Insertion sort
  - Ogni elemento viene confrontato agli elementi alla sua sinistra, parzialmente ordinati, scambiandolo fino a trovare il suo posto

# Due algoritmi $O(n \lg n)$

- Merge sort (John Von Neumann ~ 1945)
  - Se ci sono meno di due elementi, la sequenza è ordinata
  - Dividi la sequenza in due parti (circa) uguali
    - Applica ricorsivamente l'algoritmo alle due parti
    - Combina le due sottosequenze mantenendo l'ordine
- Quick sort (Tony Hoare ~ 1960)
  - Se ci sono meno di due elementi, la sequenza è ordinata
  - **Partiziona** la sequenza rispetto ad un elemento scelto **a caso** (detto pivot)
    - A sinistra gli elementi minori, a destra gli elementi maggiori
    - Il pivot è nella posizione corretta
  - Applica ricorsivamente l'algoritmo alle due parti

# Ingegneria del software

- Come gestire la complessità di un progetto?
  - Approccio sistematico alla creazione del software
    - Struttura, documentazione, milestones, comunicazione e interazione tra partecipanti
  - Analisi dei requisiti
    - Formalizzazione dell'idea di partenza, analisi costi e usabilità del prodotto atteso
  - Progettazione
    - Struttura complessiva del codice, definizione architetturale
    - Progetto di dettaglio, più vicino alla codifica ma usando pseudo codice o flow chart
  - Sviluppo
    - Scrittura effettiva del codice, e verifica del suo funzionamento via **test**
  - Manutenzione
    - Modifica dei requisiti esistenti, bug fixing



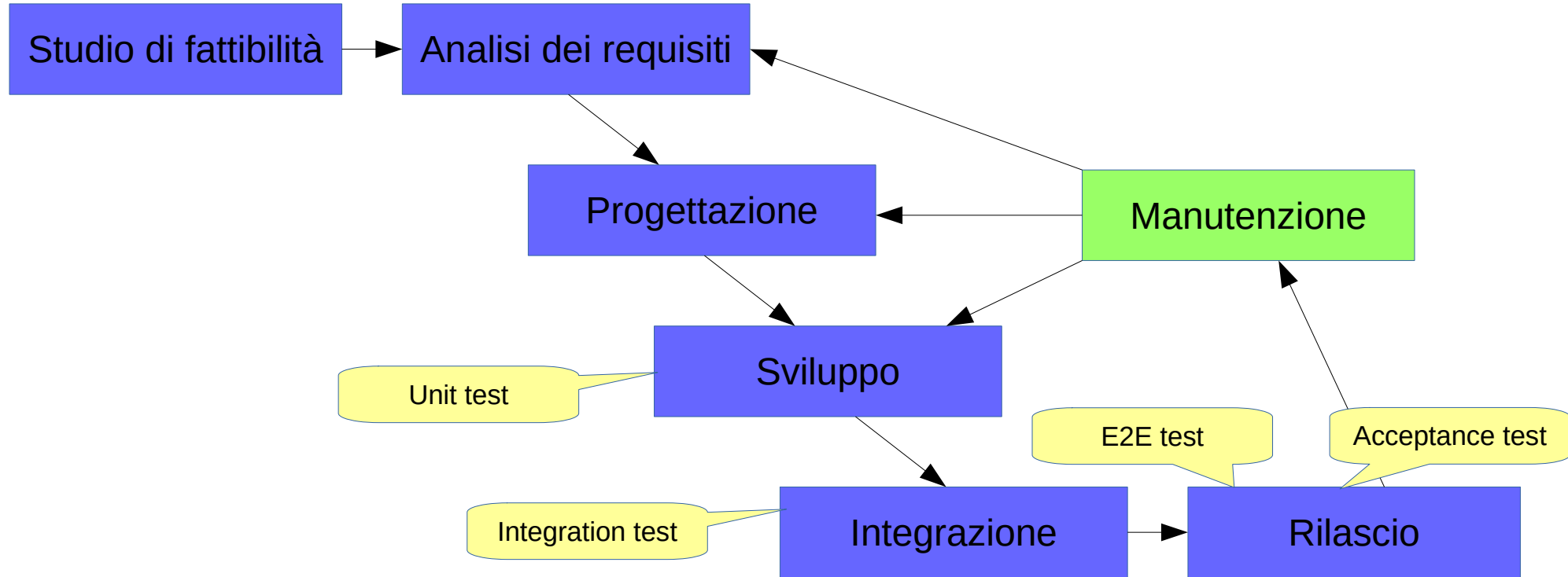
# Test

- **Unit Test:** singola “unità” di codice, isolata dalle altre
  - Mostrano che i requisiti sono rispettati
  - Possono richiedere la simulazione di dipendenze → mock
  - Verifica dei casi base (positivi e negativi) e di casi limite
  - Ci si aspetta che siano ripetibili, semplici e che offrano una elevata copertura del codice
- **Integration Test:** unità + dipendenze
  - Possibile l’uso di mock per focalizzarsi su specifica dipendenza
- **End to end test:** intera applicazione
  - Richiede tipicamente un lungo tempo d’esecuzione
  - Difficile da implementare per applicazioni complesse

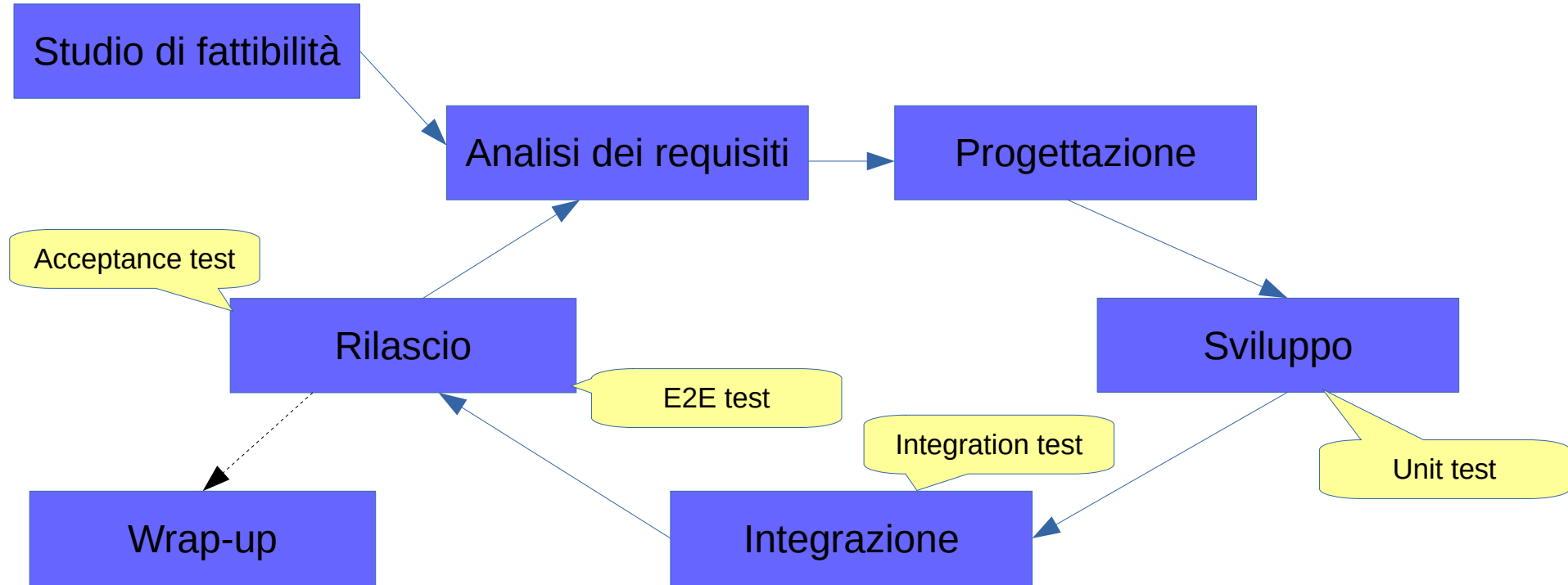
# Ciclo di vita del software

- Programmazione
  - sviluppo, unit test, review, condivisione del code base, merge
- Build
  - Integrazione del code base
- Integration Test
- Packaging
  - Gestione degli artefatti, preparazione del rilascio, End to End Test
- Rilascio
  - Gestione dei cambiamenti, approvazione, automazione del rilascio
- Configurazione
- Monitoring
  - Valutazione delle performance e qualità del prodotto

# Modello a cascata (waterfall)



# Modello agile





# Software Developer

- Front End Developer
  - Pagine web, interazione con l'utente
    - HTML (struttura), CSS (stile), JavaScript (interattività)
    - Framework: Angular, React, Vue, Bootstrap, ...
    - User Experience (UX)
- Back End Developer
  - Logica applicativa, persistenza
    - Java, C/C++, Python, JavaScript, SQL, ...
      - JavaEE, Spring, Node.js, DBMS, ...
- Full Stack Developer
  - Front End + Back End, DevOps (CI / CD), ...