

Il corso, in breve

- Introduzione alla programmazione
- Java SE
- Un database relazionale: MySQL (o Oracle DB, ...)
 - SQL standard + estensioni procedurali, integrazione Java – SQL via JDBC
- Tecnologie Web: HTML, CSS, JavaScript
- Java EE: Web, JPA (Hibernate)
- Progettazione e sviluppo di una WebApp

Emanuele Galli – www.linkedin.com/in/egalli/

Download & Install

- Come editor per Windows conviene usare qualcosa di più evoluto di notepad, ad es: <https://notepad-plus-plus.org/>
- Java SE **11** (LTS) JDK – preferita Oracle (occorre registrarsi), tra le JDK alternative, OpenJDK
<https://www.oracle.com/java/technologies/javase-jdk11-downloads.html> – <https://adoptopenjdk.net/> (o altra, ma sempre **11**)
- Apache Maven 3: <https://maven.apache.org/download.cgi> (di solito il “Binary zip archive”)
- Git client 2: <https://git-scm.com/>
- Eclipse IDE for **Enterprise Java Developers**: <https://www.eclipse.org/downloads/packages/> (zip)
- Un DBMS relazionale
 - MySQL 8: <https://dev.mysql.com/downloads/mysql/>
 - Oracle DB 18c XE: <https://www.oracle.com/database/technologies/xe-downloads.html>
- Node.JS 14 LTS: <https://nodejs.org/en/download/>
- Visual Studio Code: <https://code.visualstudio.com/>
- Browser <https://www.mozilla.org/it/firefox/new/> o https://www.google.com/intl/it_it/chrome/
- Apache Tomcat 9: <https://tomcat.apache.org/download-90.cgi> (Core – [Win] zip)
- Può essere utile anche avere Postman: <https://www.postman.com/downloads/>

Le basi dell'informatica

- Informatica (*information automatique*) → trattamento automatico dell'informazione
- Computer Science: studio dei computer e di come usarli per risolvere problemi
- 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)
- Tra le altre operazioni
 - XOR (disgiunzione esclusiva)

| A | B | AND | OR | XOR |
|---|---|-----|----|-----|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

| 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

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

Internet

- Evoluzione 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

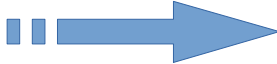
- Deve il suo nome al matematico persiano Al-Khwarizmi (~800)
- Sequenza di istruzioni che fornisce il risultato di un certo problema
 - Ordinata, esecuzione sequenziale (con ripetizioni)
 - Operazioni ben definite ed effettivamente eseguibili
 - Completabile in tempo finito (e ragionevole)
- È corretto 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



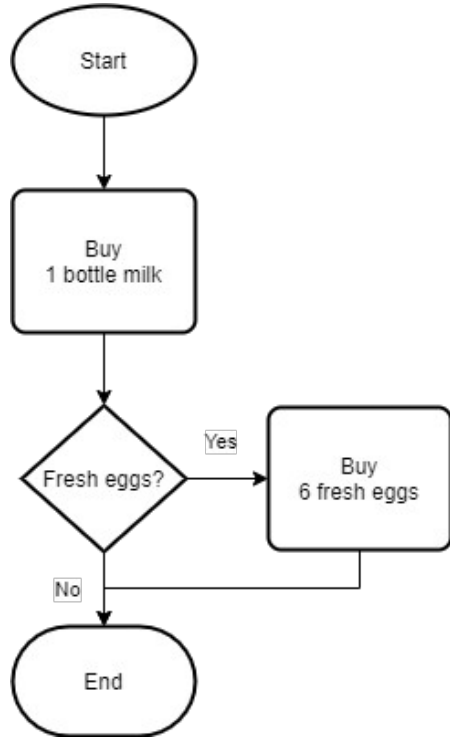
Istruzioni

- Operazioni di base fornite da un linguaggio di programmazione
 - In senso stretto, sono le operazioni messe a disposizione da un processore
- Operazioni fondamentali
 - Assegnamento, somma, prodotto, ...
- Decisioni: si esegue un blocco di istruzioni solo se una data condizione è vera
- Iterazioni: si esegue un blocco di istruzioni finché una condizione è vera
 - Con controllo di terminazione prima o dopo ogni iterazione
 - Con indicazione del numero di volte da iterare
- Salto: abbandono della normale sequenza di esecuzione
 - Può essere incondizionato o subordinato ad una data condizione

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

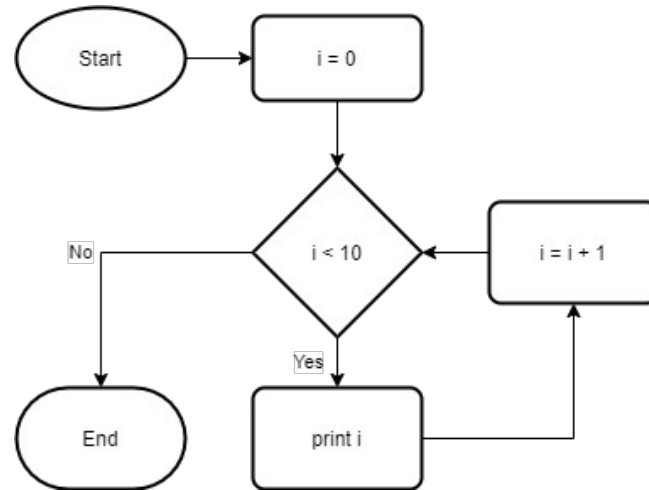
Flow chart



EG64-2104

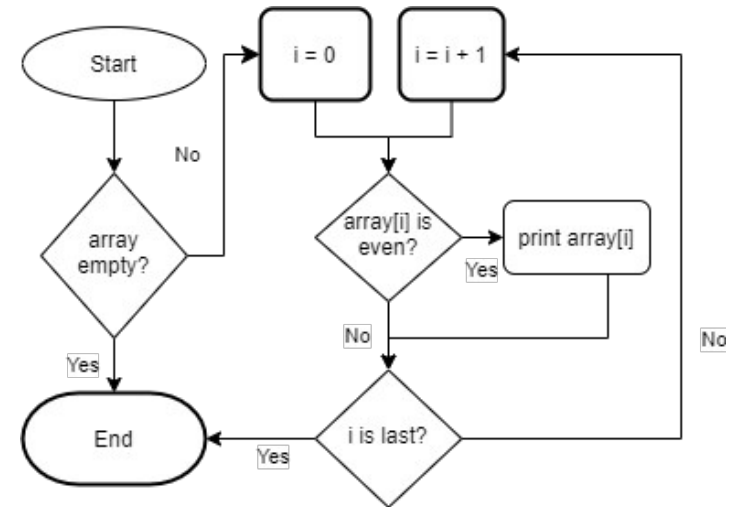


Print numbers in [1..10)



Dev Intro

Print even numbers in an array



13

Pseudo codice

```
print "Hello"
```

```
buy 1 bottle milk
```

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

```
// print array elements in position [0..10]
```

```
for i in 0 .. len(array):  
    print array[i]
```

```
// 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, astrazione dalle specifiche macchine
- 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

| Pet |
|---|
| - name: string - <u>count: int</u> |
| + eat(): void + <u>getCount(): int</u> |

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 su 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
- È comodo operare con funzioni anonime
 - Dette anche **funzioni lambda** (nome derivato dal lavoro di Alonzo Church)
- Facilita lo sviluppo di applicazioni che prevedono l’esecuzione in parallelo

Funzione / Procedura / Metodo

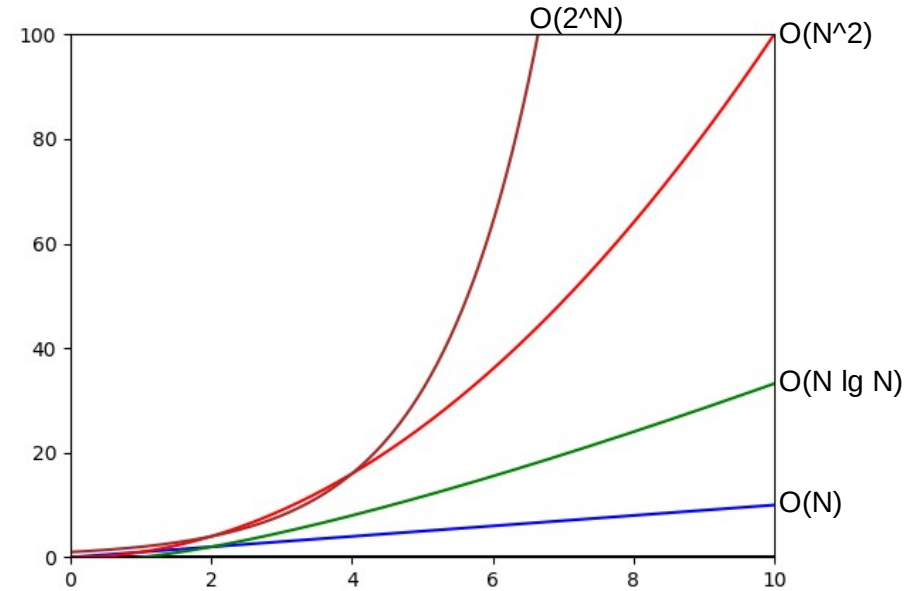
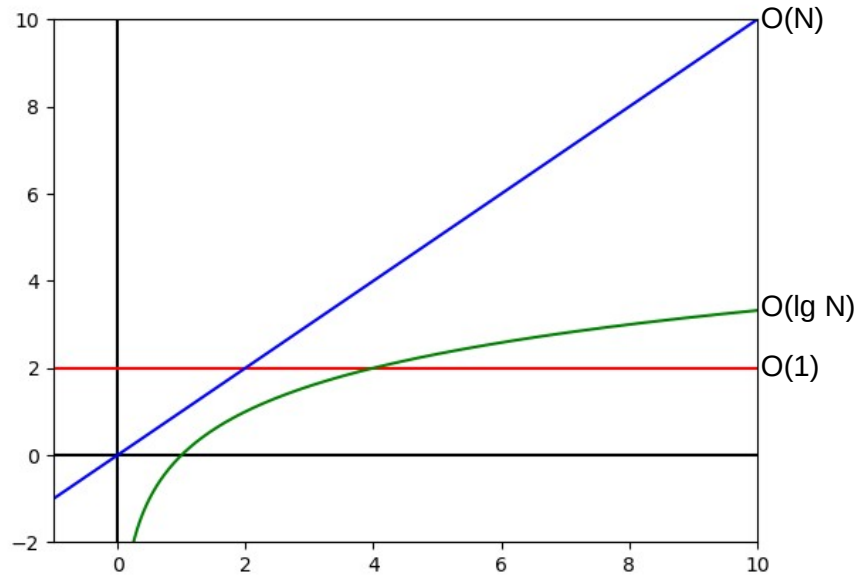
- Nell'informatica, una **funzione** è un blocco di codice identificato da
 - Un nome (non strettamente necessario, vedi lambda)
 - Una lista di parametri (input, può essere vuota)
 - Il tipo del valore ritornato (output, può non ritornare niente)
 - Una **procedura** è una funzione che non ritorna alcun 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
- FP: le funzioni sono "cittadine di prima classe" del linguaggio
 - Utilizzabili come lo sono i dati nei confronti di funzioni e variabili

Complessità degli algoritmi

- Caso migliore, peggiore, medio in tempo e spazio
 - In funzione del numero “n” di elementi su cui l'algoritmo opera
- “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



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, tipicamente con base
 - 0 (C/C++, Java, Python, ...) o 1 (MATLAB, R, Julia, ...)
 - Allocati in un blocco contiguo di memoria di dimensione prefissata
 - Accesso diretto via indice ai suoi elementi: $O(1)$
 - Non è possibile lasciare “buchi” in mezzo alla struttura
 - “Spingendo” i buchi “in fondo”, l’eliminazione di un elemento ha complessità media $O(n)$
- **Hash table** → array associativo in cui l’indice dagli elementi è il suo numero “hash”
 - Le operazioni di ricerca, inserimento, eliminazione hanno complessità media $O(1)$
 - Non esiste alcun concetto di ordine
 - Vanno implementate con attenzione (generazione hash number, fattore di carico)

Strutture dati

- Tre strutture basate sul **Nodo** → dato e riferimento a uno o altri nodi
- **Lista** → ha una testa (e di solito una coda)
 - Nodi collegati in maniera lineare (semplice o doppia)
 - L'accesso, inserimento, eliminazione $O(1)$, ma solo in testa (e coda)
- **Albero** → ha una radice, ogni nodo può avere n figli
 - BST: Binary Search Tree, è ordinato, ogni nodo può avere al massimo 2 figli
 - Se bilanciato, accesso, inserimento, eliminazione in $O(\log_2 n)$
- **Grafo** → simile all'albero, ma
 - non ha radice, gli archi possono avere un peso ed essere orientati

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 basati sul 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 UML, pseudo codice o flow chart
 - Sviluppo
 - Scrittura effettiva del codice, e verifica del suo funzionamento via **test**
 - Manutenzione
 - Modifica dei requisiti esistenti, bug fixing



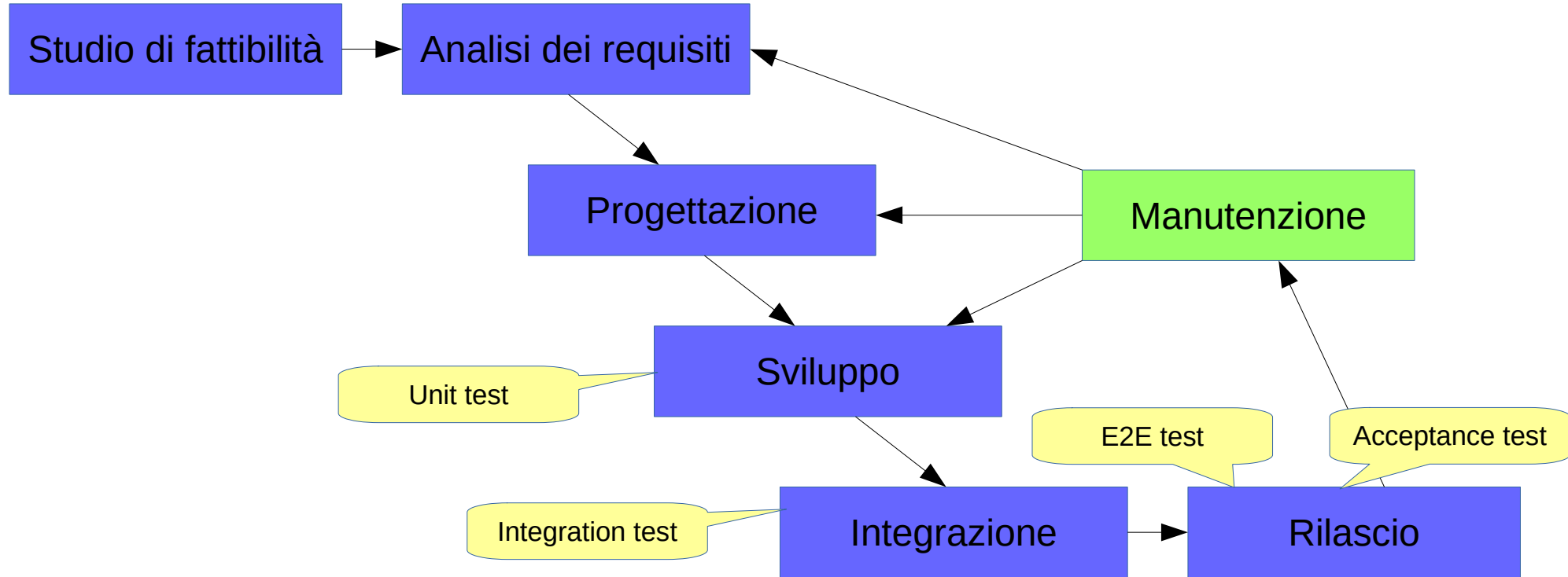
Test

- **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

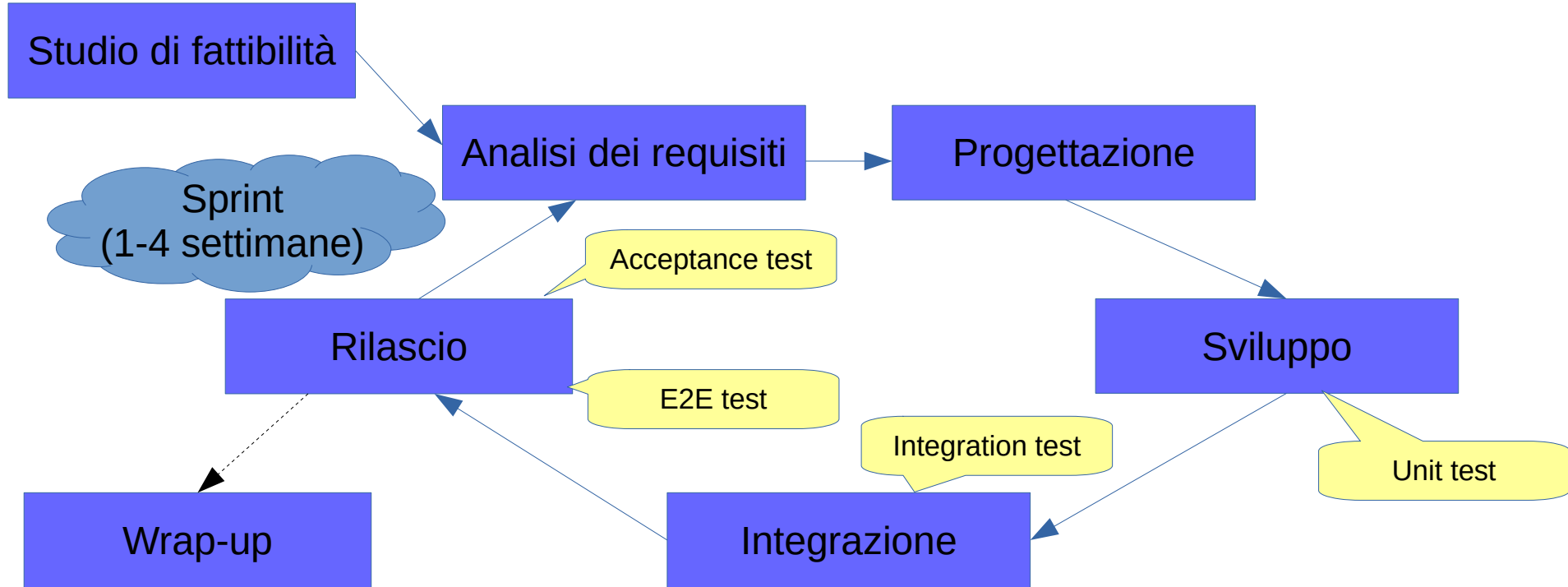
Ciclo di vita del software

- 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

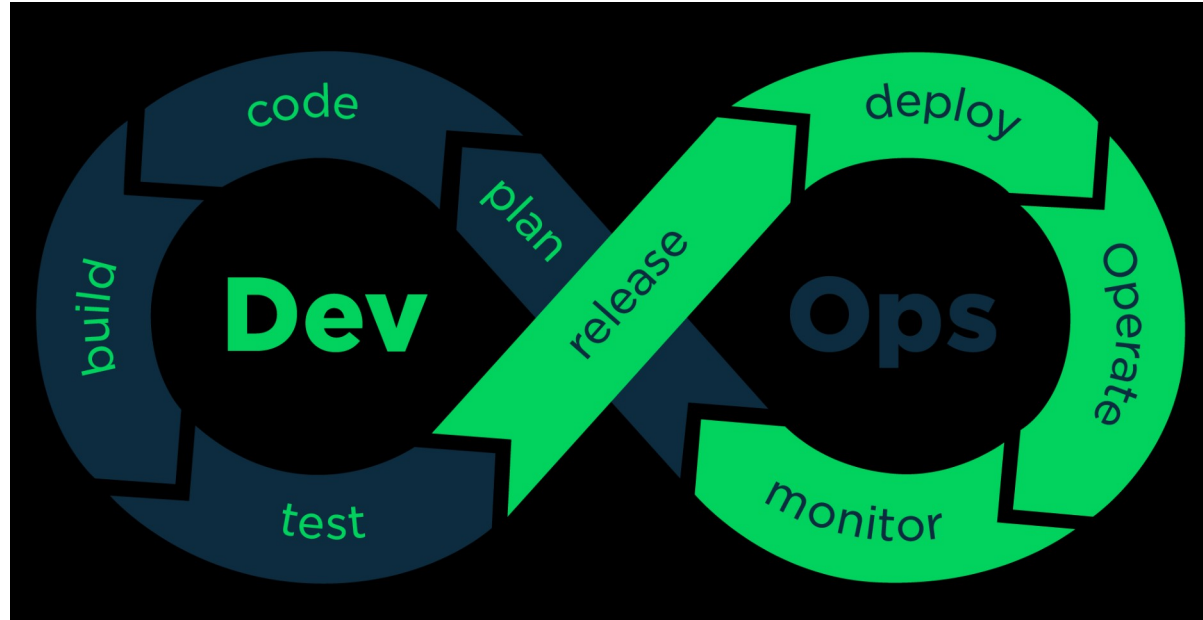
Modello a cascata (waterfall)



Modello agile



Fasi DevOps



Devopedia. 2020. "DevOps." Version 7, January 6. Accessed 2021-02-08. <https://devopedia.org/devops>

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), ...