Ministero dell'Economia e delle Finanze

Ministero del Lavoro e delle Politiche Sociali

PEGASO
SOCIETÀ COOPERATIVA SOCIALE

Modulo didattico: Manutenzione e diagnostica del PC

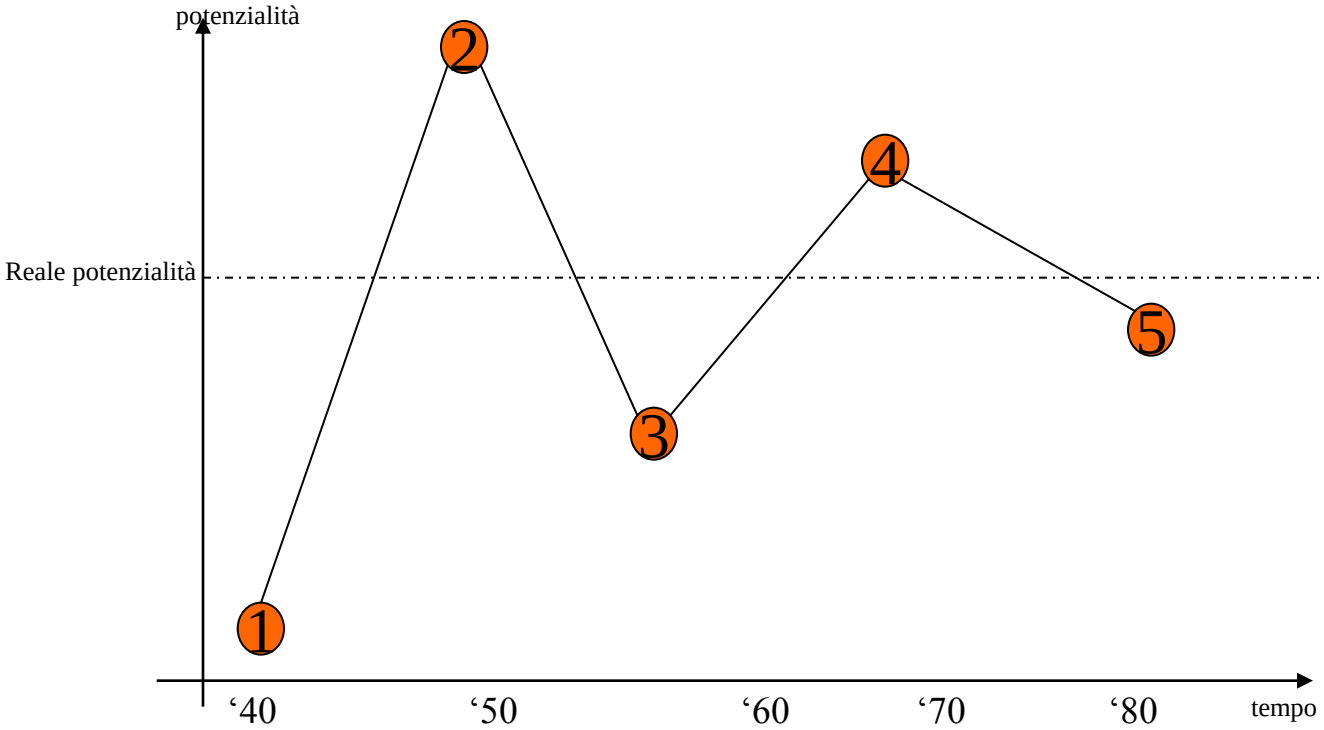
Premessa sui linguaggi

- Il linguaggio di programmazione si può vedere come un ponte tra diverse realtà:
- Il sistema reale (Nessuna assunzione di buon comportamento)
- Il sistema di calcolo (comportamento deterministico)
- Obbiettivo: avvicinare i due sistemi.
- Strumento: astrazione.

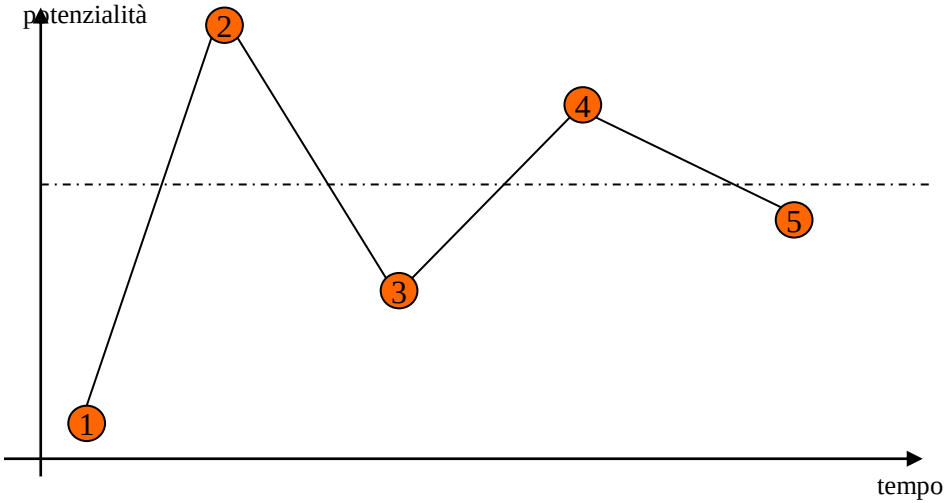
Evoluzione dei linguaggi (cont.)

Carl Adam Petri ha proposto un sistema di riferimento per descrivere un computer, indicando le varie immagini che il calcolatore ha avuto nel corso del tempo caratterizzando il livello di potenzialità in funzione del tempo.

Il computer nasce alla fine della 2^ guerra mondiale, la sua evoluzione non è stata lineare ma a sbalzi

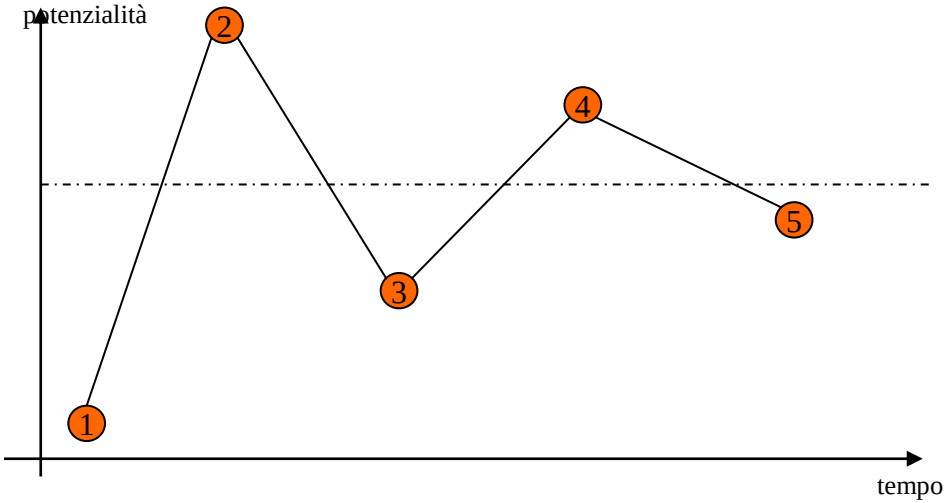


Evoluzione dei linguaggi (cont.)



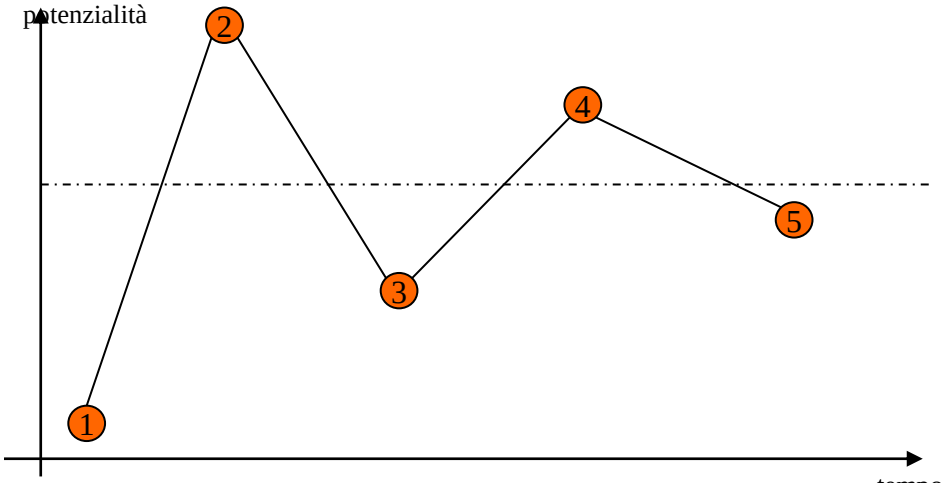
1 Anni ‘40: utilizzato per calcoli balistici,
- veloce macchina calcolatrice;
- nascono i linguaggi: linguaggio macchina, assembler;
- programmi sequenziali;
- sottovalutazione delle potenzialità.

Evoluzione dei linguaggi (cont.)



2 Anni ‘50 : si suppone che sia una sorta di super intelligenza, in grado di replicare il comportamento e pensiero umano;
- nascono i linguaggi: Lisp, Prolog;
- programmi funzionali;
- sopravalutazione delle potenzialità.

Evoluzione dei linguaggi (cont.)

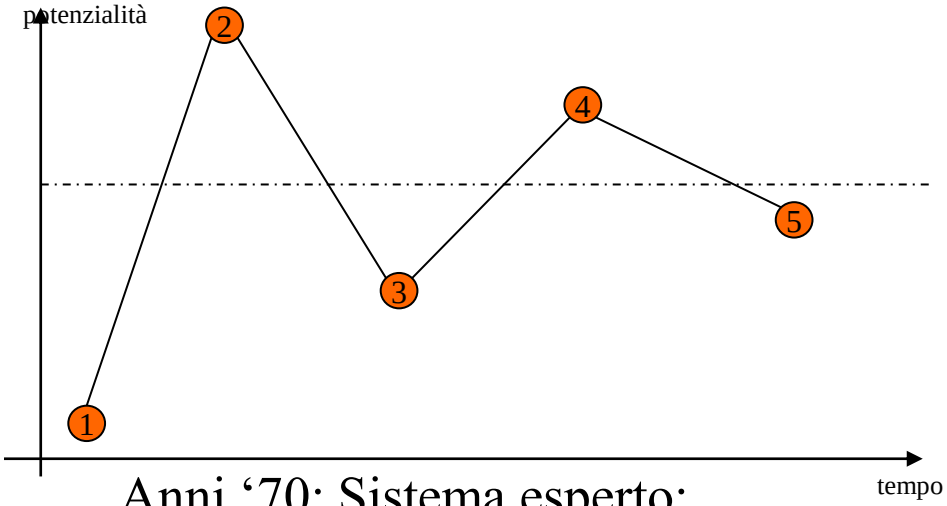


Anni ‘60 : strumento versatile,

3

- capace di trattare problemi molto diversi
- HLL (High Level Language): Fortrant, COBOL;
- VHLL (Very High Level Language): Pascal, ADA, C
- programmi procedurali;
- sottovalutazione delle potenzialità;
- spostamento dall’essere legato al sistema di calcolo al sistema reale.

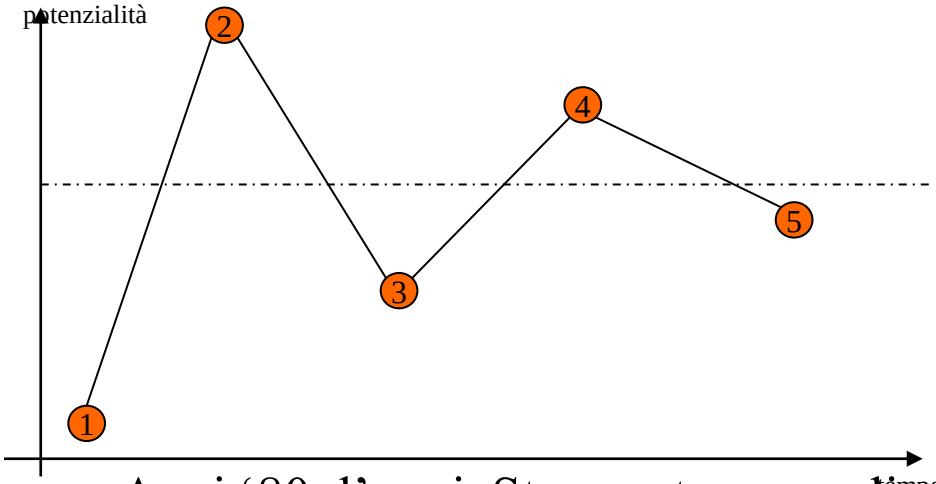
Evoluzione dei linguaggi (cont.)



4

Anni ‘70: Sistema esperto;
- Partner della comunicazione;
- Immagine esagerata in quanto i calcolatori trasmettono solo dati, mentre l’essere umano trasmette altro (con i gesti, con la voce, i consigli).

Evoluzione dei linguaggi (cont.)



5

Anni ‘80: l’oggi, Strumento generalizzato di comunicazione;
- non più un partner ma uno strumento;
- generalizzato perché trasmette solo dati ma anche suoni, immagini etc.
- nascono i linguaggi ad oggetti, C++, Smaltalk, Eiffel, Java.

Evoluzione dei linguaggi

- Dal 3 al 5 si ha un passaggio dai mainframe ai personal computer, e si può vedere come sono radicalmente cambiati ed evoluti i linguaggi di programmazione.

- 1: Programming in the small
- 3: Programming in the large
- 5: Sistemi embedded

Ambienti di sviluppo

- I linguaggi diventano sempre più astratti;
- Nascono gli ambienti di programmazione integrato che consente di mettere insieme tool+elementi e tool sempre più sofisticati.
- Gli ambienti di sviluppo facilitano anche il lavoro di gruppo (Programming in the large).

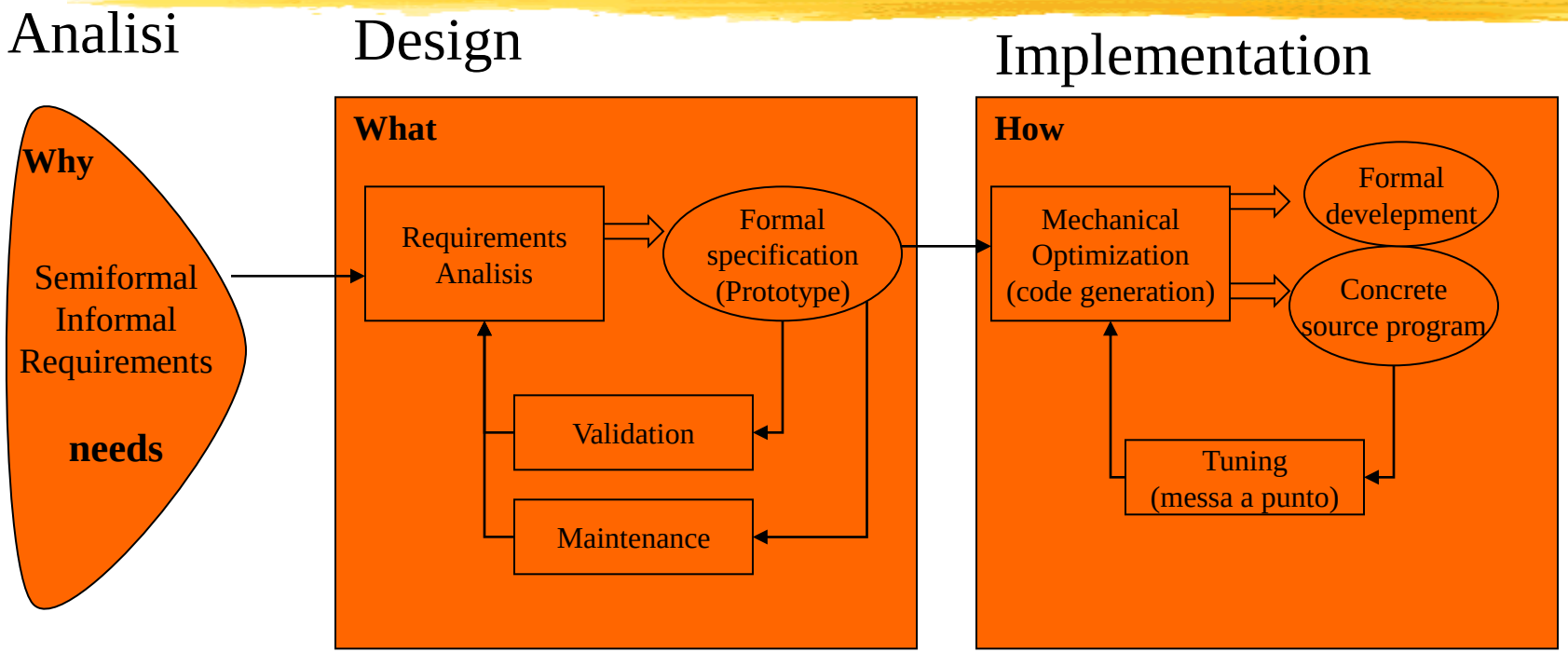
Modularità

- **Incrementabilità:** (nella costruzione) consente di considerare un problema per volta;
- **Riusabilità:** serve per poter riusare porzioni di codice che trovo in tutti i sistemi e che quindi sono già testati.
- **Modificabilità:** posso cambiare una sola componente senza dover ripensare a tutto;
- Modulo = Componente

Step per la creazione del software

- I passaggi per arrivare da un sistema reale al linguaggio sono:
- **Analisi:** carattere informale, compiuta tra progettisti e utenti, serve ad individuare i requisiti fondamentali;
- **Disegno:** svolta dai progettisti, avanzano una soluzione che possono man man raffinare;
- **Implementazione:** traduzione del disegno in uno specifico lunguaggio di programmazione.

Step per la creazione del software



- All'estremità sinistra (needs) c'è il sistema reale dove si parla un linguaggio dichiarativo logico e informale o semi formale;
- All'estremità destra c'è il sistema di calcolo.

Step per la creazione del software

- La fase di **analisi** consente di avere una traccia dei requisiti che si vogliono;
- **Informal requirement/semi formal:** sono i requisiti, le esigenze di chi vuole sviluppare un sistema informatico (Requisiti base);
- Va sempre più affermandosi l'idea che il semi formal deve diventare formal già in fase di analisi, utilizzando un linguaggio dichiarativo magari logico;

Step per la creazione del software

- La fase di **Design** è il punto di inizio della progettazione, facendo delle scelte di tipo hardware e software;
- La letteratura e l'esperienza consiglia di realizzare dei prototipi funzionanti per avere una visione dell'avanzamento del progetto, creando selle simulazioni dei casi reali. Tali prototipi vanno sempre più raffinati in accordo con le specifiche del sistema.
- La simulazione in fase di design permette di anticipare il tuning che altrimenti peserebbe tutto sulla fase di implementazione.
- Il costo di riparazione di un errore è tanto più alto quanto più tardi è rilevato.
- Si ottiene un programma non ottimizzato ma che è una buona guida per il successivo sviluppo.
- **Maintenance:** necessaria se sopraggiungono modifiche

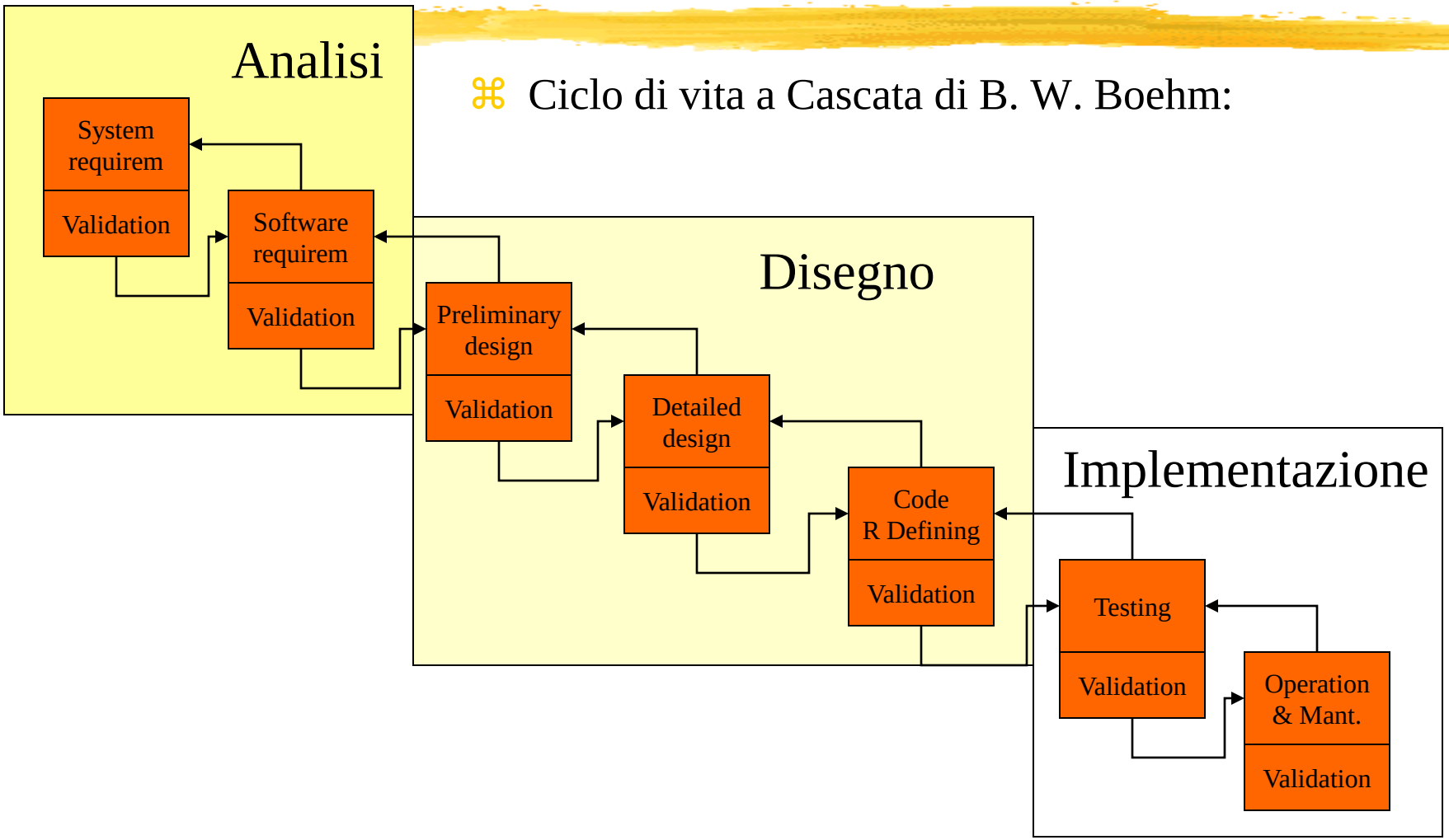
Step per la creazione del software

- La fase di **implementazione** deve consentire di ottimizzare il codice ed svilupparlo in un linguaggio adatto alle varie esigenze;
- **Tuning:** messa a punto quando il codice è stato terminato si passa alla fase di test e di manutenzione. Se in questa fase si ci accorge che si è commesso un errore di analisi sarà molto più difficile e dispendioso modificare il codice.

- Seguendo queste fasi non si diminuisce il tempo di realizzo di un progetto, ma si evitano errori di valutazione che porterebbero a raddoppiare il tempo necessario.

Ciclo di vita del software (cont.)

- Ciclo di vita a Cascata di B. W. Boehm:

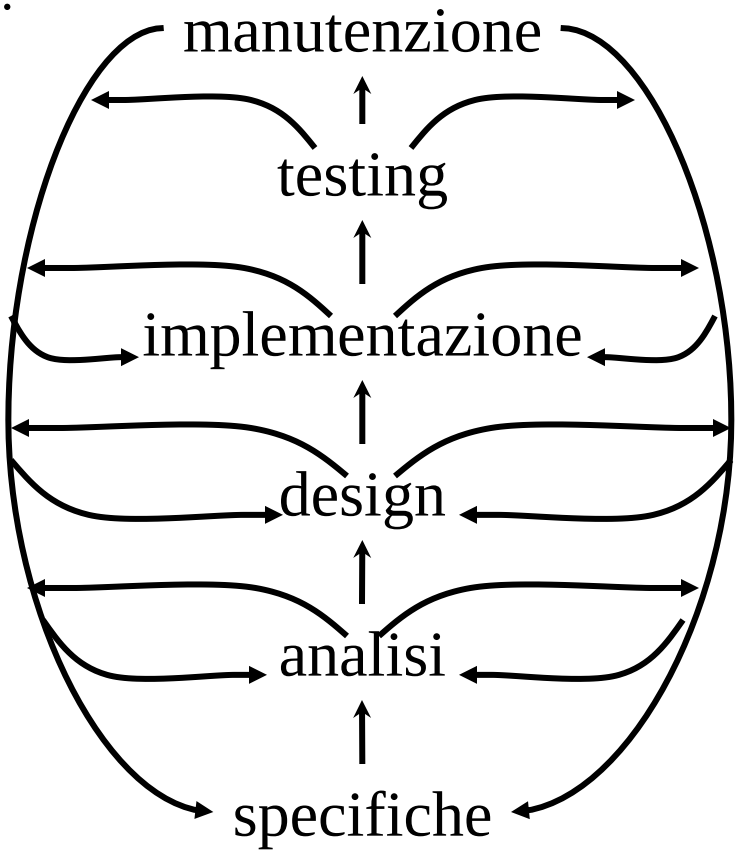


Ciclo a Cascata

- Il ciclo di vita a cascata identifica una serie di attività per arrivare da una fase iniziale di prima identificazione dei requisiti all'attività di software operativo in un ambiente;
- Il merito di Boehm e quello del riconoscimento della necessità di un controllo sistematico all'indietro;
- Ogni volta che si fa un passo si effettua una verifica.

Ciclo di vita del software (cont.)

- Ciclo di vita a fontana:

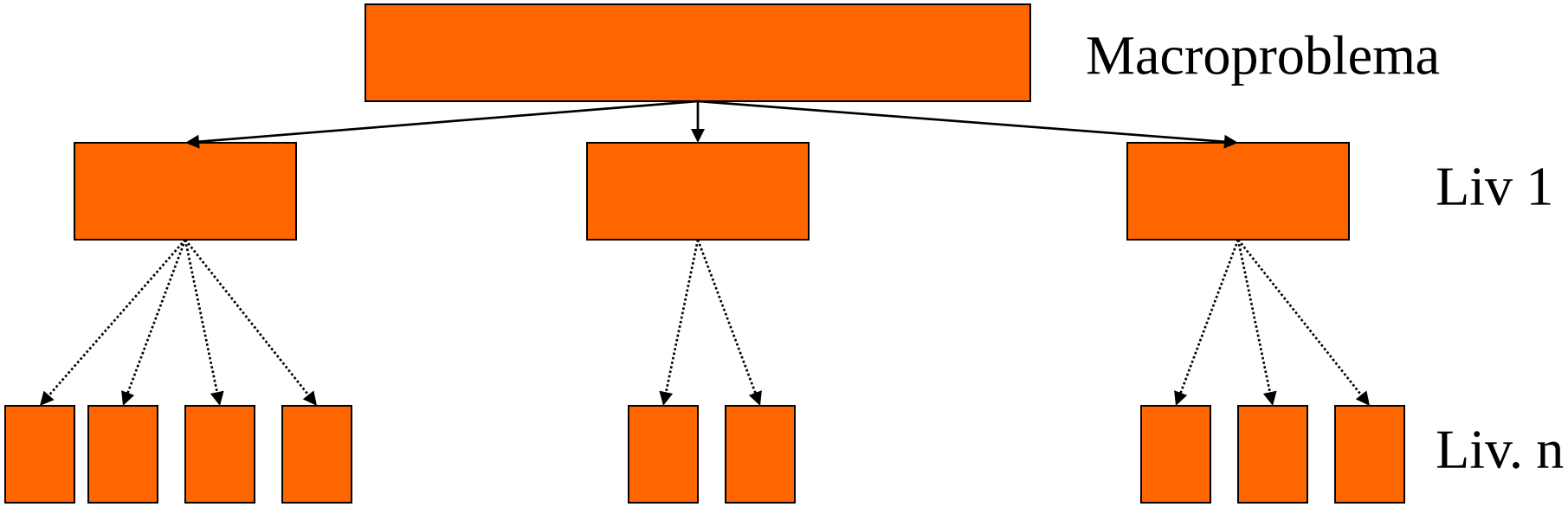


Ciclo a fontana

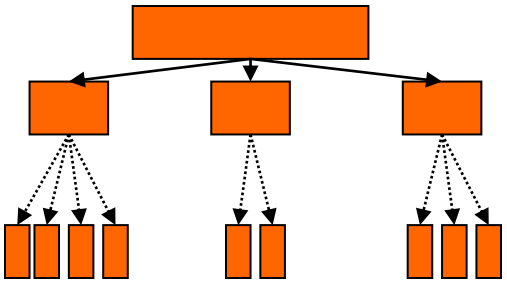
- Si intuisce che è un ciclo non una sequenza di cose da fare;
- da qualunque punto si può tornare indietro.
- Gli obiettivi rimangono sempre quelli del ciclo a cascata.

“Top-Down” “Bottom-Up”

- Il sistema da realizzare si presenta come un macro problema, la fase di analisi deve servire per scomporre il macro problema in sotto problemi di più facile soluzione;
- Iterare questo procedimento sino al raggiungimento di problemi di elementare soluzione.

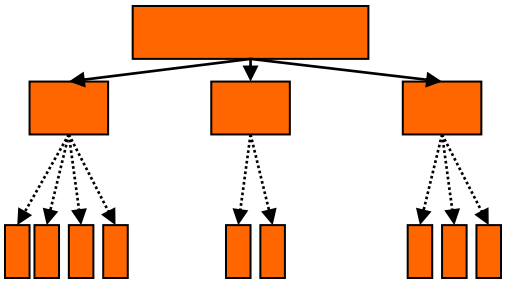


Top-Down



- Si comincia a sviluppare i sotto problemi, si compongono alla fine per il raggiungimento dell'obbiettivo finale;
- Si fa poco uso di codice già scritto;
- Non si sfrutta a pieno la modularità

Bottom-up



- Il denominatore comune con la metodologia Top-Down è la scomposizione in sotto problemi di elementare soluzione;
- Quando si è arrivato ai problemi elementari si controlla se si sono risolti in precedenza problemi analoghi;
- In caso affermativo si fa forte riuso di questi moduli, magari utilizzando l'ereditarietà.

Bottom-up

- Per la realizzazione di grandi progetti sorge la necessità di:
- lavorare in team;
- riuso di componenti già implementate e librerie sempre più fornite.

- Il codice diventa mezzo di comunicazione tra programmatori e tra programmatori e macchina (explanation).

- L'approccio Bottom-Up mette l'enfasi su ciò che accomuna le varie applicazioni (facilita la riusabilità).