Storia del Software

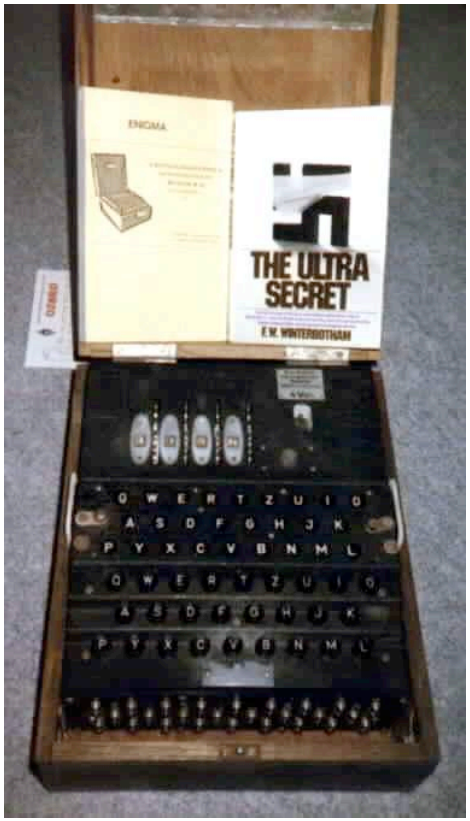
Dario Carotenuto

Università degli Studi di Napoli "Federico II", Via Cinthia, 80126 - Napoli, Italy

April 5, 2007

Il Software: origini

Il termine *Software* ha origine durante la seconda guerra mondiale.



Enigma

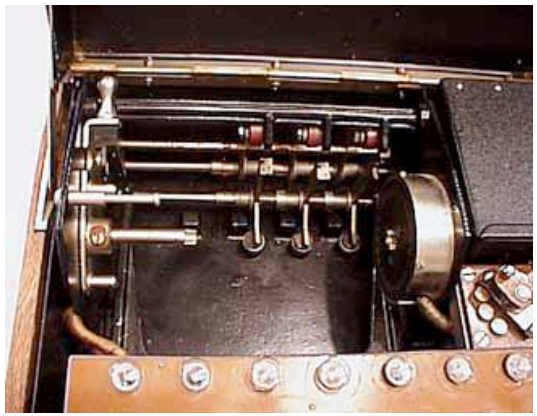
I tecnici dell'esercito inglese erano impegnati nella decrittazione dei codici tedeschi di Enigma, di cui già conoscevano la meccanica interna (detta hardware, roba dura, nel senso di ferraglia) grazie ai servizi segreti polacchi.

Il Software: origini

La prima versione di Enigma sfruttava tre rotori per mescolare le lettere.

Il Software di Enigma

Dopo il 1941, ad Enigma venne aggiunto un rotore, e il team di criptanalisti inglesi, capitanati da Alan Turing, si dovette interessare non più alla sua struttura fisica, ma alle posizioni in cui venivano utilizzati i rotori della nuova Enigma. Dato che queste istruzioni erano scritte su pagine solubili nell'acqua per poter essere più facilmente distrutte, e per contrasto con hardware, furono chiamate software.



Software: cos'è?

Definizione

"Software: le parti di un computer che non funzionano."

Arthur Bloch Murphy' Law 2000 Penguin Putnam Inc. May 1999

- Il senso moderno del termine deriva dalle istruzioni date ai computer, ed è stata utilizzata per la prima volta nel 1957 da John W. Tukey, noto statistico statunitense.
- Dal 1950 l'analogia tra l'hardware ed il corpo umano e quella tra il software e la mente umana si è fatta molto forte, dal momento che Turing ha sostenuto che il progresso tecnologico sarebbe riuscito a creare entro il 2000 delle macchine intelligenti, in grado di pensare.

Codice e potenziamento dell'hardware

- Alla storia dell'evoluzione del software è legato lo sviluppo dell'hardware.
- Una minaccia alla velocità di elaborazione viene dal software, sempre più complesso.
- Ciò che conta per un'utente non è tanto la velocità di elaborazione del processore, quanto la velocità effettiva di processamento del codice, calcolata in base al tempo che occorre alla CPU per svolgere un compito (come la scrittura di un testo, la creazione di una cartella, ecc.).

La complessita' del software

Crescita del software

Nathan Myhrvold, direttore dell'Advanced Technology Group della Microsoft, ha effettuato uno studio sui prodotti Microsoft calcolando le linee di codifica per le successive release dello stesso software:

- Basic: 4.000 linee di codice nel 1975 a 500.000 nel 1995
- Word: 27.000 linee di codice nel 1982 a 2.000.000 nel 2002

Solo aggiunta di funzionalità?

Confronti

Progresso tecnologico

- La continua aggiunta di nuove funzionalità al software esistente giustifica la costante richiesta di processori più veloci memorie più grandi e più ampie capacità di I/O (Input/Output).
- Infatti, anche le altre tecnologie si sono evolute di pari passo:
- i dischi rigidi da 10 MB (1982) a 500 GB (2006);
- i modem analogici da 300 bit/sec a 56 kbit/sec e oggi si arriva a trasferire 20Mb/sec sul doppino di rame (adls)

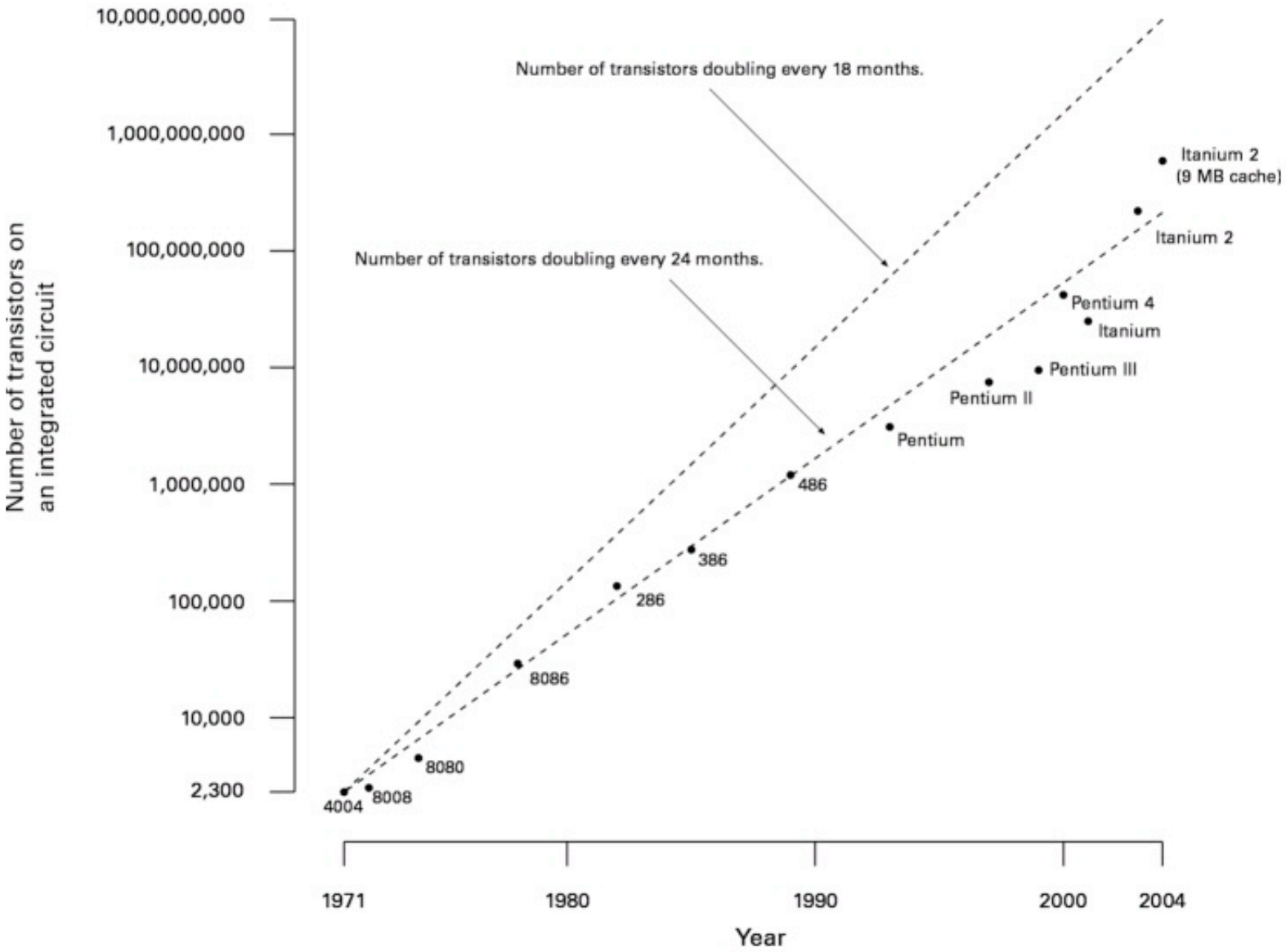
Legge di Moore

La legge di Moore, più che una legge vera e propria, è una osservazione empirica.

G. Moore

Gordon Moore, cofondatore di *Intel* con Robert Noyce, è stato lo studioso che ha dato il via alla corsa all'evoluzione dei processori, grazie alle sue supposizioni, poi diventate leggi, conosciute proprio come prima e seconda legge di Moore. Nel 1965 Gordon Moore, che all'epoca era a capo del settore R&D della Fairchild Semiconductor e tre anni dopo fondò la Intel, scrisse un articolo su una rivista specializzata nel quale spiegava come nel periodo 1959-1965 il numero di componenti elettronici (transistor) che formano un chip raddoppiava ogni anno.

Diagramma Legge di Moore



La *crisi del software*

Perchè fare software era difficile

- "Software crisis" è un termine usato nei primi giorni dell'Ingegneria del software per descrivere l'impatto della rapida crescita della potenza degli elaboratori e la complessità dei problemi che dovevano essere affrontati. Le parole chiave della software crisis erano complessità, attese e cambiamento.
- I requisiti, continuamente in conflitto tra loro, impedivano lo sviluppo del software. Per esempio, mentre gli utenti domandavano un *largo numero di funzionalità*, i comittenti, generalmente, chiedevano di *minimizzare i costi dello sviluppo ed i tempi*.
- Le cause della software crisis era collegate alla complessità dei processi software ed alla relativa immaturità dell'ingegneria del software.

Legge di Moore ancora valida?

Superamento della *crisi del software*

Myhrvold traccia un parallelismo con la legge di Moore: "abbiamo aumentato la dimensione e la complessità del software ancora più rapidamente di quanto non prevedeva la legge di Moore", "gli utenti del software hanno sempre consumato le maggiori capacità di elaborazione ad una velocità uguale o superiore a quella con cui i produttori di chip le mettevano a disposizione" (1995).

Cosa ha causato questa inversione di tendenza?

Ciclo Windows-Intel (WINTEL)

Hardware/Software

- La maggior potenza di ogni nuovo chip Intel, deriva da una nuova generazione di applicazioni basate sul sistema operativo Windows (noto per la pesantezza del codice), che dà origine a una rinnovata domanda di chip ancora più potenti necessari per far girare nuovi software a loro volta ancora più esigenti in termini di potenza.
- Questo ciclo "vizioso" ha preso il nome nome di "Ciclo WINTEL", dal nome dai colossi di riferimento (processori Intel e sistemi Windows).
- Oggi si afferma che: "Every time Andy Grove makes a faster chip, Bill uses all of it" (ogni volta che Andy Grove crea un chip più veloce, Bill Gates lo usa tutto quanto).

Realizzazione del software

Varietà linguistica e modularità

- Un software viene normalmente realizzato utilizzando uno o più linguaggi di programmazione.
- Se il progetto diventa complesso, è opportuno dividere il programma in uno o più moduli, che possono essere così affidati a diversi programmatori, modificati più semplicemente e *riutilizzati* in altri progetti.

Realizzazione del software

Fasi

- La fase detta di *compilazione*, traduce ogni file del codice sorgente in un file oggetto contenente il programma in linguaggio macchina adeguato all'architettura hardware di destinazione.
- In seguito tutti i file oggetto attraversano una fase di *linking* per giungere al prodotto finale: il file eseguibile.
- Alcuni *software non* vengono *compilati* in quanto le istruzioni contenute nel codice sorgente vengono subito eseguite da un software detto *interprete*.

Qualità del software

Processo di sviluppo

La gestione del processo di sviluppo è caratterizzato dalla scelta di un *modello di sviluppo* del software codificato nell'ambito dell'Ingegneria del Software (Software Engineering), tra gli altri esistono:

- Il modello classico, o a cascata (water-fall)
- Il modello a spirale (object oriented)

Qualità del software

I Limiti alla qualità

La realizzazione del software è un'attività complessa articolata in più fasi. Per questo motivo può essere associato ad un prodotto ingegneristico, ma se ne differenzia soprattutto per alcune caratteristiche:

- è molto "malleabile";
- è un prodotto *human intensive* (e cioè un prodotto che richiede un considerevole sforzo in risorse umane perché si concentra soprattutto sulla progettazione e sull'implementazione).
- Limiti teorici *concreti*

Licenze d'utilizzo e distribuzione

Licenze e diffusione del software

Le licenze d'utilizzo e distribuzione sul software sono numerose (almeno 54), ma quelle effettivamente diffuse sono poche.

- Per l'89% si tratta di GPL, LGPL o BSD (licenza storica di Unix, tornata in uso dall'avvento di Linux).
- La licenza d'uso è un documento che accompagna il software e specifica i diritti e i doveri di chi lo riceve e di chi lo diffonde.
- Esistono licenze liberali (come la licenza Open Source) e licenze proprietarie.
- Tutte le licenze d'uso traggono il loro valore legale dalle norme sul diritto d'autore (il copyright).
- Nasce in seguito anche l'Open content che ha come scopo quello di trasferire le licenze su opere diverse dal software.

Una definizione

Definizione..

Software e' un termine generico che definisce programmi e procedure utilizzati per far eseguire al computer un determinato compito. Viene in generale suddiviso in:

- *software di base* o di sistema perchè è indispensabile al funzionamento del computer dal momento che senza di esso non sarebbe che hardware inutilizzabile. Viene identificato con il sistema operativo;
- *software applicativo*: realizzati sulla base delle funzionalità offerte dal sistema operativo, ad esempio applicazioni gestionali destinate alle esigenze specifiche di un utente o di un'azienda, office automation, ...

Una definizione

..incompleta

- Un software deve essere corredato da documentazione, manuali e possibilmente deve essere comprensibile all'utente finale
- Deve essere chiaro cosa fa il programma!

Ma cos'è un programma?

Programmi

- Un programma è l'implementazione di un algoritmo in un linguaggio di programmazione comprensibile all'agente che deve eseguirlo
- Sostanzialmete è una sequenza di istruzioni di un linguaggio formalmente definito
- Da un punto di vista puramente sintattico, un programma è una stringa o frase appartenente ad un linguaggio di programmazione

Lo sviluppo

Sviluppo del software

- Quando si realizza un prodotto o un sistema software è importante svolgere una *serie di passi*, una sorta di percorso che aiuti ad ottenere risultati di alta qualità in un tempo prefissato.
- Tale percorso è chiamato *processo di sviluppo software*.
- Esso è composto da alcune attività che rappresentano un'insieme di compiti da svolgere per sviluppare un software:
  - Attività portanti: una serie di compiti da svolgere necessariamente;
  - Attività ausiliarie: possono aumentare la qualità di un software da produrre, di solito tali attività sono considerate dalle aziende che cercano una certa qualità.

Attività portanti

Attività portanti

Si possono definire cinque attività portanti in un processo software:

- la stesura delle specifiche:
  - tale attività comprende innanzitutto la definizione dei servizi che il software deve offrire, le funzioni che deve offrire e le prestazioni minime che deve esibire (la descrizione di cosa il software deve fare - specifica dei requisiti);
  - specifiche di progetto, che descrivono come il sistema deve essere realizzato;
- lo sviluppo del software:
  - Questa attività comprende la produzione del software, il quale deve rispondere alle specifiche precedentemente stabilite;
- l'ispezione del software, allo scopo di verificarne l'aderenza alle specifiche e, più in generale, di validarlo;
- Il collaudo del software;
- la manutenzione del software.

Attività ausiliarie

Attività ausiliarie

Tra le attività ausiliarie di un processo software si elencano:

- La localizzazione del software
- La gestione della qualità del software
- La gestione della configurazione (Configuration management)

Caratteristiche del processo software

Un processo software deve possedere alcune caratteristiche fondamentali

- Comprensibile: si deve capire perché si è scelto di seguire un modello di sviluppo piuttosto che un altro;
- Visibile: si deve capire a che punto si è giunti nello sviluppo, seguendo i dati precedentemente riportati sulle documentazioni di ciascuna fase del ciclo di vita del software;
- Supportabile: il processo dev'essere supportato dagli strumenti che si decide di utilizzare per lo sviluppo del software;
- Accettabile: un processo dev'essere accettabile da coloro i quali si accingono a realizzarlo;
- Robusto: un processo deve risultare robusto al punto di essere flessibile ai cambiamenti che potrebbero influenzare lo sviluppo del software;
- Rapido: un processo dev'essere rapido nel produrre il software

I linguaggi

Definizione

- Un linguaggio di programmazione è un *linguaggio formale* dotato di una sintassi ben definita, e generalmente descritta con strumenti quali la notazione *BNF* oppure i *grafi sintattici*, e si usa per scrivere programmi per calcolatori, cioè per codificare algoritmi e strutture dati in un tutto organico, in una forma più vicina al linguaggio umano scritto.
- L'alternativa sarebbe scrivere direttamente le istruzioni nel codice macchina del particolare processore, un compito improponibile per programmi meno che semplicissimi.
- In effetti un linguaggio di programmazione deve soddisfare il Teorema di Bohm-Jacopini, il cui enunciato informale è che un linguaggio deve essere sequenziale (cioè eseguire le istruzioni in sequenza), possedere una struttura di selezione logica (il cosiddetto IF) ed una di iterazione (di norma espressa come FOR).

I programmatori

Cos'è un programmatore?

E' colui che ti risolve in modo incomprensibile un problema che non sapevi di avere.

- La professione del programmatore è relativamente recente e si è sviluppata di pari passo con l'aumento dei campi di applicazione dell'informatica. L'attività del programmatore o "attività di codifica" è successiva all'attività di analisi e precedente a quella di collaudo, solitamente eseguita da altri professionisti; gli *analisti* ed i *collaudatori beta tester*.
- In pratica il programmatore sulla base del documento di specifica (tecnica o funzionale), in cui sono descritti i comportamenti e gli algoritmi che devono essere realizzati dal programma software, redatto dell'analista, realizza il programma vero e proprio.
- Terminata la fase di programmazione con la produzione di un programma eseguibile, quello le cui istruzioni sono eseguite dalla macchina, inizia la fase di collaudo, tesa a determinare la conformità del programma alle specifiche descritte nel documento di analisi.

I programmatori

Gli strumenti

Il programmatore per l'esecuzione della propria attività si avvale di alcuni strumenti informatici:

- Ad esempio editor (programmi che permettono la scrittura di programmi software),
- debugger,
- helper,
- compilatori o interpreti,
- script,
- database ed altri ancora.

Il primo programma

Ada Lovelace è considerata la prima programmatrice della storia, avendo per prima espresso un algoritmo inteso per l'utilizzo su un computer, la macchina analitica di Charles Babbage nell'ottobre del 1842.

Gli sviluppatori

Sviluppatore

- Lo sviluppatore è chi produce applicazioni o sistemi software e segue l'intero ciclo di sviluppo del software (non solo la codifica quindi).
- Vedremo in seguito come l'approccio a tale mansione si sia modificato nel corso del tempo

Modello di Von Neumann

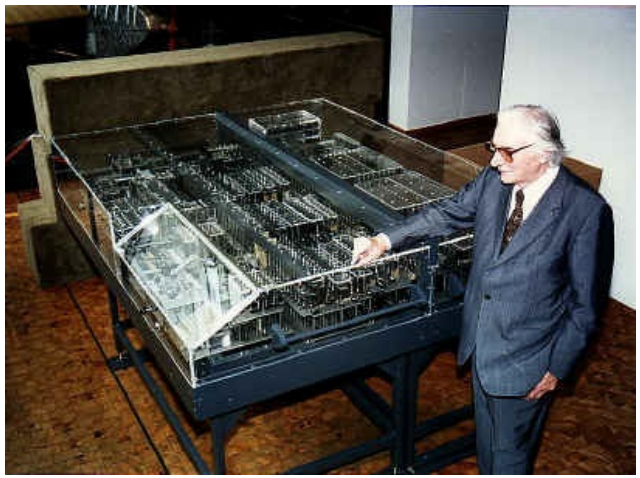
L'idea della programmabilità delle macchine è espresso dal modello computazionale noto come "Architettura di von Neumann",

von Neumann

Una delle personalità scientifiche preminenti del XX secolo cui si devono fondamentali contributi in campi come teoria degli insiemi, fisica quantistica, economia, informatica, teoria dei giochi, fluidodinamica e in molti altri settori della matematica. Per capire quanto sia stato importante il contributo di von Neumann alla scienza basti pensare che la definizione del modello computazionale noto come "Architettura di von Neumann", modello su cui si basano tutti i computer che usiamo abitualmente, è uno dei suoi contributi meno significativi.

Nascita del software

La prima vera macchina programmabile risale forse al 1936 ed è lo Z1 di Konrad Zuse.



Si trattava di un apparecchio programmabile, in grado di processare numeri in formato binario e le cui caratteristiche più apprezzabili, viste con il senno di poi, furono la netta distinzione fra memoria e processore.

Nascita del software

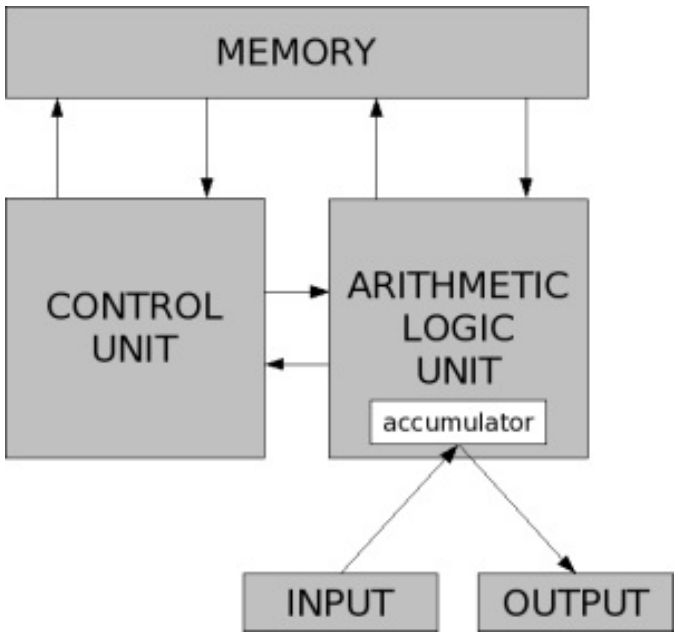
Questa architettura, che non venne adottata dall'ENIAC o dal Mark I, (i primi computer realizzati negli Stati Uniti quasi dieci anni più tardi), rispecchia la definizione di calcolatore enunciata nel 1945 da John von Neumann. Lo Z1 conteneva tutti i componenti di un moderno computer, anche se era completamente meccanico, come ad esempio: unità di controllo, memoria, micro sequenze, logica floating point, ecc.

Qualche dato tecnico:

- frequenza di lavoro: 1 hertz
- velocità media di calcolo: 1 moltiplicazione in 5 secondi
- memoria: 64 celle a 22 bit
- consumo elettrico: 1.000 watt
- numero di relays: nessuno, usava migliaia di piastre in metallo, circa 20.000.

Nel 1986 Konrad Zuse decise di ricostruire il perduto e famoso Z1. Lo Z1 era dotato anche del primo linguaggio di programmazione: il Plankalkül (pubblicato nel 1946).

Architettura di Von Neumann



Schema di progettazione di calcolatori elettronici

Lo schema si basa su quattro componenti fondamentali:

- CPU o unità di lavoro che si divide a sua volta in
  - Unità operativa, nella quale uno dei sottosistemi più rilevanti è l'ALU (Arithmetic Logic Unit)
  - Unità di controllo
- Unità di memoria, intesa come memoria di lavoro o memoria principale (RAM, Random Access Memory)
- Unità di input, tramite la quale i dati vengono inseriti nel calcolatore per essere elaborati
- Unità di output, necessaria affinché i dati elaborati possano essere restituiti all'operatore
- All'interno dell'ALU è presente un registro detto accumulatore.

Lo Shortcode

Lo Shortcode

La programmazione dei primi elaboratori veniva fatta in Shortcode (1949), da cui poi si è evoluto l'assembly o assembler, che costituisce una rappresentazione simbolica del linguaggio macchina.

Lo Shortcode

Example

```
a = (b+c)/b*c
was converted to Short Code by a sequence of substitutions and a
final regrouping:
X3 = (X1+Y1)/X1*Y1 substitute variables
X3 03 09 X1 07 Y1 02 04 X1 Y1 substitute operators and
parentheses.
07Y10204X1Y1 group into 12-byte words.
0000X30309X1
```

- La sola forma di controllo di flusso è l'istruzione di salto condizionato, che porta a scrivere programmi molto difficili da seguire logicamente per via dei continui salti da un punto all'altro del codice.
- Lo schema di traduzione non era automatizzato.

Nascita del software:l'assembly

- L'assembly, o linguaggio assemblatore (spesso chiamato impropriamente assembler) è, tra i linguaggi di programmazione, quello più vicino al linguaggio macchina vero e proprio.
- L'assembly ha lo scopo generale di consentire al programmatore di ignorare il formato binario del linguaggio macchina. Ogni codice operativo del linguaggio macchina viene sostituito, nell'assembly, da una sequenza di caratteri che lo rappresenta in forma mnemonica; per esempio, il codice operativo per la somma potrebbe essere trascritto come ADD e quello per il salto come JMP.

Example

```
MOV AX,BX ; Istruzione di carico del registro BX nel registro AX.
JMP ETICHETTA ; salto incondizionato alla label sopra
```

Classificazione

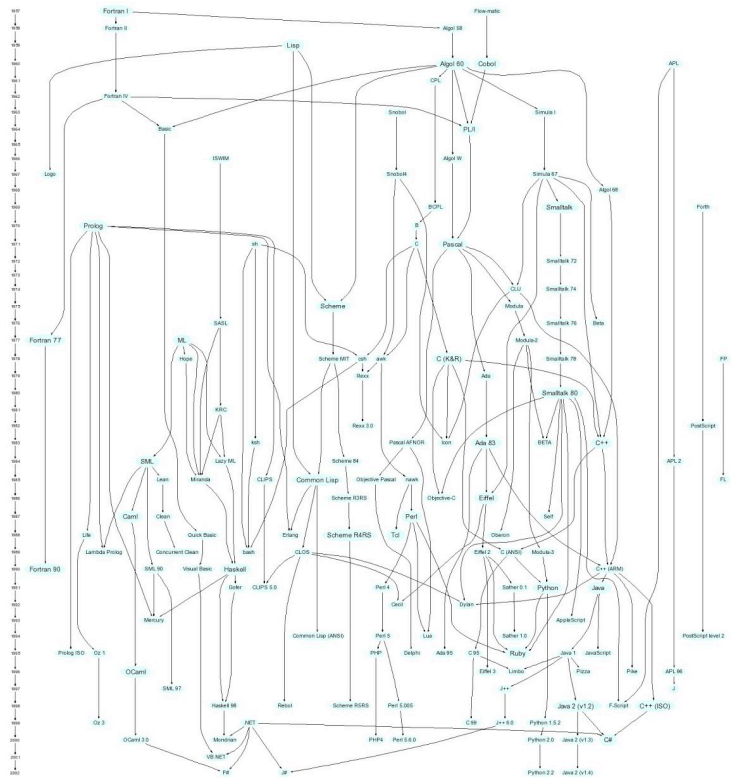
Prima classificazione

Linguaggi di programmazione classificati in base alle loro caratteristiche fondamentali.

- – Linguaggio macchina, binario e fortemente legato all'architettura.
- – Linguaggi Assembler, istruzioni del linguaggio macchina e riferimenti alle celle di memoria espressi mediante simboli.
- – Linguaggi di alto livello, tentativo di astrazione dell'architettura sottostante. Operazioni di più alto livello rispetto a quelle caratteristiche della macchina.

I Linguaggi di programmazione

el_f2.jpg 1149×1233 pixel 03/04/2007 05:36 PM



http://www.windoweb.it/edpstory_new/foto_storia_linguaggi/el_f2.jpg Pagina 1 di 1

Perché questa varietà?

Classificazione

Primi tentativi di astrazione

La maggior parte dei linguaggi di programmazione successivi cercarono di astrarre dalla macchina, dando la possibilità di rappresentare strutture dati e strutture di controllo più generali e più vicine alla maniera (umana) di rappresentare i termini dei problemi per i quali ci si prefigge di scrivere programmi.

I primi linguaggi

Fortran, BASIC

Tra i primi linguaggi ad alto livello a raggiungere una certa popolarità ci fu il Fortran, creato nel 1957 da John Backus, da cui derivò successivamente il BASIC (1964): oltre al salto condizionato, reso con l'istruzione IF, questa nuova generazione di linguaggi introduce nuove strutture di controllo di flusso come i cicli WHILE e FOR e le istruzioni CASE e SWITCH: in questo modo diminuisce molto il ricorso alle istruzioni di salto (GOTO), cosa che rende il codice più chiaro ed elegante, e quindi di più facile manutenzione.

Altri linguaggi storici

Algol, Lisp

Dopo la comparsa del Fortran nacquero una serie di altri linguaggi di programmazione storici, che implementarono una serie di idee e paradigmi innovativi: i più importanti sono l'ALGOL (1960) e il Lisp (1959). Tutti i linguaggi di programmazione oggi esistenti possono essere considerati discendenti da uno o più di questi primi linguaggi, di cui mutuano molti concetti di base.

Considerazioni sui primi linguaggi

Formalismi adatti a descrivere concetti

- A quel tempo, i problemi significativi non riguardavano tanto l'organizzazione del processo di produzione, di solito lasciato nelle mani di pochi esperti, quanto la comprensione e la memoria di ciò che veniva sviluppato all'interno della stessa comunità degli sviluppatori.
- Di qui, la necessità di trascrivere la computazione in un formalismo familiare ai programmatori dell'epoca; era il tempo delle astrazioni: formule matematiche (FORTRAN), astrazioni logiche (LISP) o transazioni economiche (COBOL).
- Inoltre, la diffusione e la diversificazione del mercato dei calcolatori elettronici rese pressante il bisogno di scrivere programmi che funzionassero su più piattaforme di calcolo.

Prima Evoluzione

Strutturare il processo di Sviluppo

- Presto ci si accorse che questo non bastava (fine anni '60 e primi anni '70). I programmi diventavano più grandi e richiedevano la collaborazione di più persone, anche più gruppi di persone.
- Occorreva strutturare il processo di sviluppo per renderlo più organico. Prendendo a prestito concetti e terminologie dalle teorie di management in voga all'epoca, e nei decenni precedenti per essere precisi, in primis una versione molto statica del *Fordismo*, ci si concentrò sull'idea di organizzare il processo di sviluppo in moduli indipendenti con interfacce chiare e componibili.
- Si diffusero concetti quali la programmazione strutturata, il data hiding e l'incapsulamento. Nacquero l'ALGOL, il C, il Pascal, il Modula 2 e l'Ada: anche il FORTRAN fu fatto evolvere in questo senso.

Il periodo del *Waterfall* Model

Sviluppo e crisi

- Il processo di produzione modulare nella sua traduzione informatica, il modello di sviluppo "a cascata", dominò gli anni '70 e i primi anni '80. I limiti di questo modello, soprattutto la sua incapacità di gestire la flessibilità richiesta dalla produzione del software, cominciarono a essere evidenti verso la metà degli anni '80.
- Ci furono piccole variazioni di rotta, come i modelli di sviluppo "a V", ma sostanzialmente il modello modulare e gli associati linguaggi strutturati rimasero predominanti.
- L'idea guida era che la mancanza di precisione, e l'incertezza che la causava, poteva e doveva essere risolta a priori, tramite specifiche più accurate e circostanziate.

Individuazione del problema

Carenze dell'approccio procedurale

- Negli anni '80 la comunità scientifica acquisì la consapevolezza che i problemi non erano solo legati alla carenza umana nelle attività di definizione del sistema, ma anche all'esistenza di una zona di ombra intrinseca allo sviluppo di un qualunque sistema software, che non permetteva di definire in modo completo e corretto fin dall'inizio le caratteristiche che il sistema software avrebbe avuto alla fine.
- Il punto di partenza per lo sviluppo di un sistema software sono, in effetti, bisogni veri o percepiti come tali.

Il problema come *oggetto* da sviluppare

Focus progressivo

- La comprensione di questa problematica permise un salto nelle modalità di sviluppo del software.
- La suddivisione modulare lasciò il posto a sviluppi del software in maniera incrementale, per così dire, *a piccoli pezzi*, in modo che si potesse usare ognuno di questi pezzi per calibrare lo sviluppo successivo.

Seconda Evoluzione

Nuovi sviluppi e nuovi linguaggi

- Lo sviluppo a piccoli pezzi si concretò in molteplici modelli di produzione, quale quello prototipale, quello incrementale, quello a spirale ecc.. In termini di linguaggi di programmazione viene dato l'avvio, nel suo complesso, all'epoca dei linguaggi orientati agli oggetti.
- Nei linguaggi orientati agli oggetti il sistema da costruire è rappresentato come un insieme di entità che interagiscono tra loro e che, interagendo, producono il risultato finale; pertanto, lo sviluppo si concentra nella identificazione di tali entità e nella successiva scrittura del codice relativo ad esse e alle loro mutue relazioni.
- I più famosi tra i linguaggi di programmazione orientati agli oggetti sono senza dubbio Smalltalk [16] e C++ [31], anche se il primo tra tali linguaggi è molto probabilmente SIMULA [9].

Metodologie agili di sviluppo

Dinamicità di sviluppo

- Lo sviluppo delle telecomunicazioni, ivi compreso il mondo del web, portò alla necessità di definire processi di sviluppo che tenessero conto di questa *realtà molto dinamica* e di linguaggi che supportassero gli associati processi di sviluppo.
- È questo il periodo dell'avvento delle *metodologie agili* di programmazione, che danno supporto esplicito sia alla mutevolezza di requisiti che alla dinamicità del sistema da sviluppare.

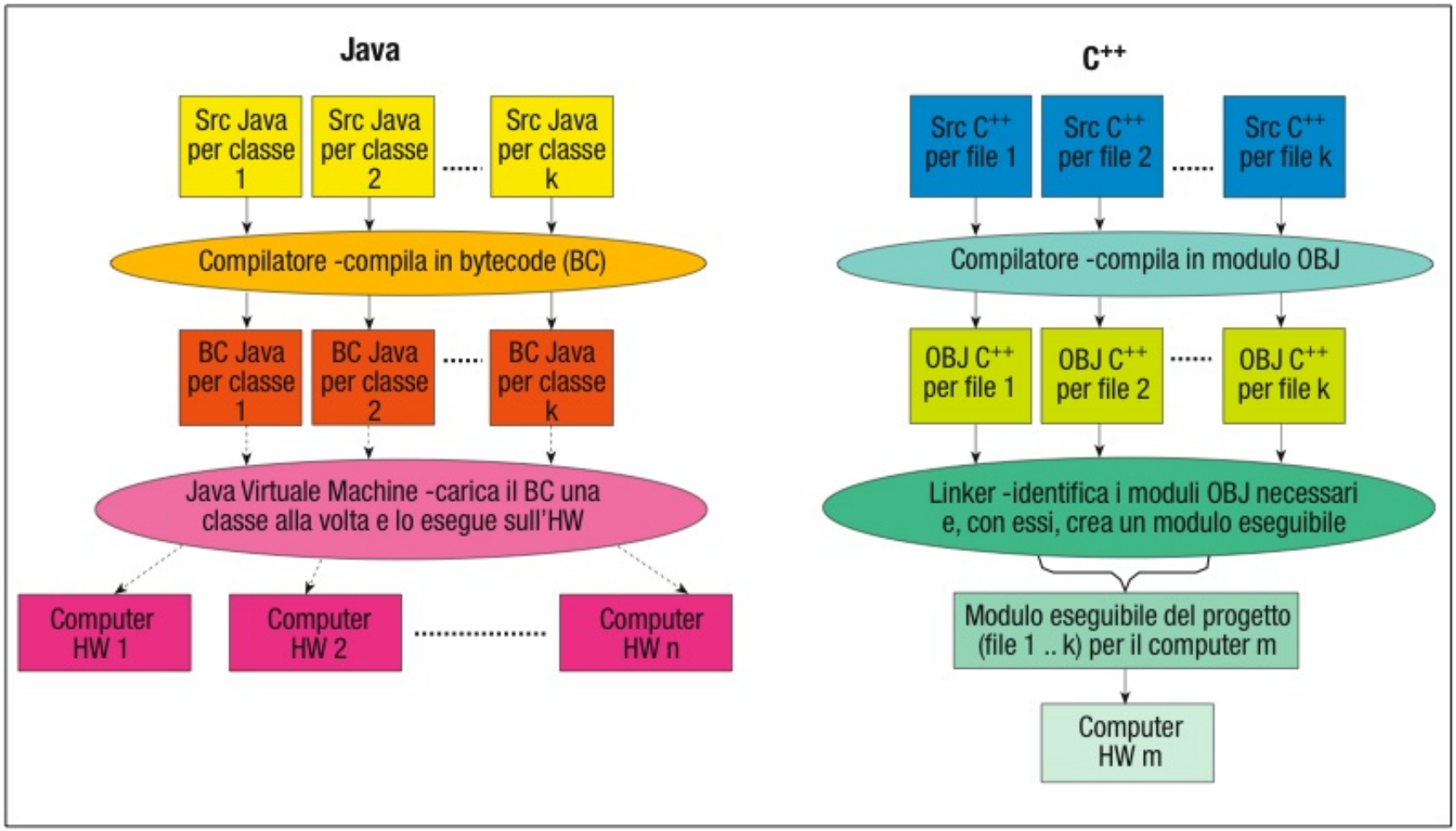
Linguaggi per lo sviluppo dinamico

Sistemi componibili dinamicamente

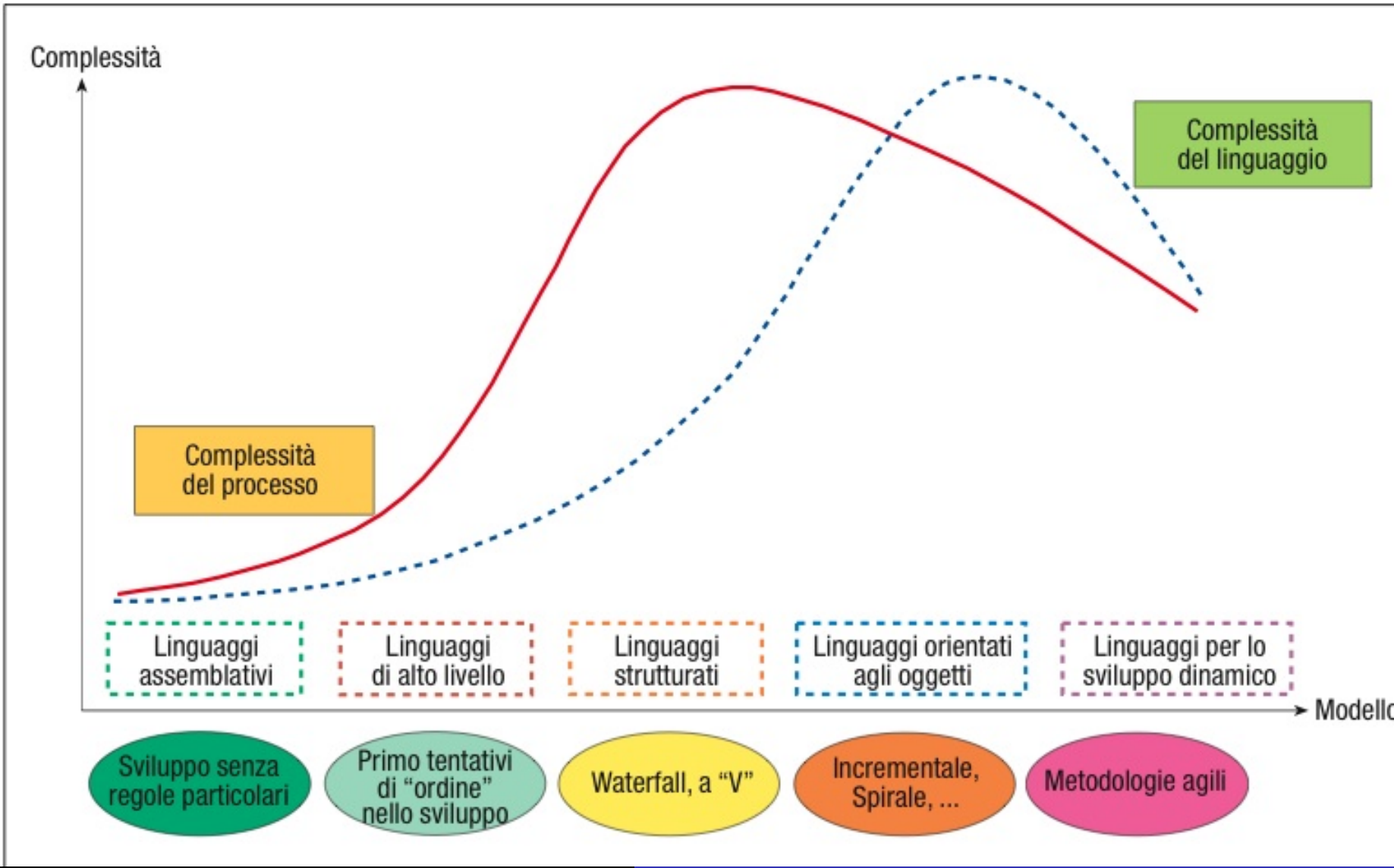
- I modelli orientati agli oggetti fornirono un importante punto di partenza in quanto permisero l'identificazione di entità base di computazione. Ma occorse dare un maggior impeto sia alla dinamicità dello sviluppo e della fornitura al cliente, che all'interoperabilità tra sistemi e con risorse disponibili in rete, quali archivi, liste di utenti, e banche dati in genere.
- Divennero allora popolari i cosiddetti linguaggi per lo sviluppo dinamico, in quanto sono in grado di sostenere lo sviluppo di sistemi componibili dinamicamente e interoperabili tra loro.
- Una parte di questi linguaggi era formata da semplici evoluzioni di linguaggi di scripting, ovvero linguaggi interpretati, finalizzati a guidare il sistema in ciò che esso doveva fare.
- Si diffusero, però, anche linguaggi di programmazione completi che ottenevano gli stessi scopi, incorporando anche quei costrutti che via via, nell'evoluzione dei linguaggi, si erano affermati come essenziali. I più famosi esempi di questi linguaggi sono Java e C#.

Questa è la situazione in cui ci si trova oggi.

Schemi di compilazione



Sintesi dell'evoluzione dei Linguaggi



Cos'è un Sistema Operativo

Definizione

In informatica, un sistema operativo (abbreviato in SO, o all'inglese OS, Operating System) è il programma responsabile del diretto controllo e gestione dell'hardware che costituisce un computer e delle operazioni di base.
Si occupa dei processi che vengono eseguiti e della gestione degli accessi degli utenti.

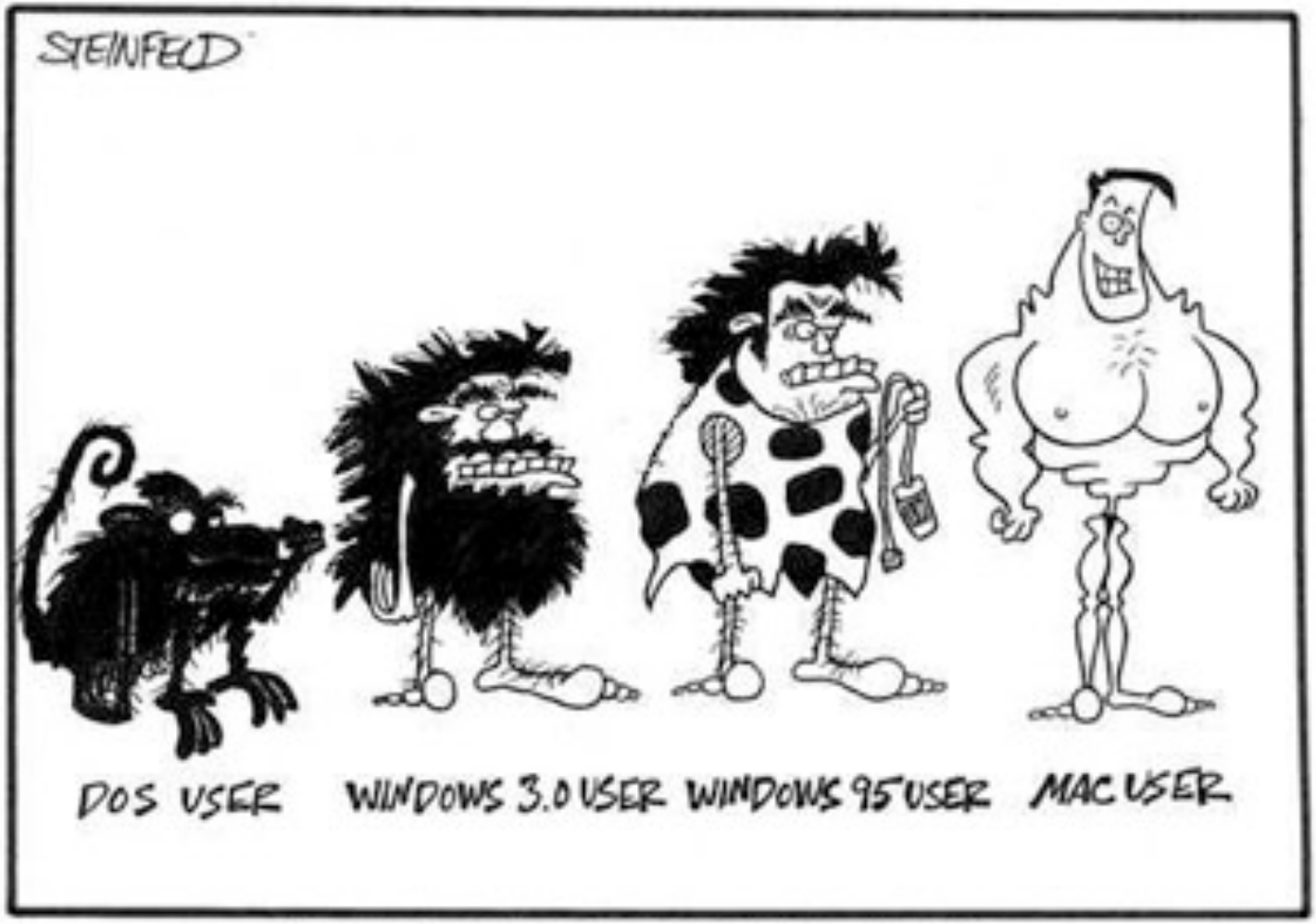
Cos'è un Sistema Operativo

Strutturazione di un SO

Un generico sistema operativo moderno si compone di alcune parti ben definite:

- un gestore di file system che si occupa di esaudire le richieste di accesso alle memorie di massa,
- un gestore di memoria virtuale che alloca pagine di memoria a richiesta e si assicura che questa sia presente nella memoria fisica al momento giusto,
- uno scheduler che assicura ai vari processi in esecuzione una ben definita quantità di tempo di elaborazione,
- uno spooler che accumula i dati da stampare e li stampa in successione,
- una interfaccia utente (shell o GUI) che permette agli esseri umani di interagire con la macchina ed un kernel, fulcro del sistema, che gestisce il tutto.

Evoluzioni dei sistemi operativi



EVOLUTION OF A COMPUTER USER

Mancanza di necessità di un SO

Gestione dei compiti

- I primi elaboratori non avevano bisogno di un sistema operativo, in quanto venivano usati da un programma per volta.
- All'inizio di ogni elaborazione, si caricavano i programmi e i dati da un pacco di schede perforate, o da un nastro perforato, o da un nastro magnetico, si eseguiva l'elaborazione, e si emettevano i risultati su stampante o su nastro magnetico.
- Per passare all'elaborazione successiva, si sostituivano i pacchi di schede e i nastri.
- I programmi applicativi avevano accesso diretto a tutto l'hardware, tipicamente tramite librerie di routine di input/output.

Il caricatore

Sistema automatico di caricamento

- Ci si rese rapidamente conto che tale procedimento era inefficiente, perché mentre l'operatore sostituiva i nastri e le schede, l'elaboratore era inutilizzato.
- Inoltre, per sfruttare l'elaboratore, occorreva la presenza continua di almeno un operatore.
- Pertanto, si passò in seguito al seguente procedimento. I pacchi di schede e i nastri perforati relativi a più elaborazioni venivano caricati in sequenza su uno stesso nastro magnetico, usando un dispositivo digitale dedicato, che non impegnasse l'elaboratore.
- I programmi applicativi erano progettati in modo da leggere i dati escusivamente da un nastro magnetico e scrivere i risultati esclusivamente su un altro nastro magnetico.
- L'elaboratore aveva a disposizione più unità a nastro magnetico, almeno due per la scrittura e due per la lettura.
- In tal modo, mentre l'operatore sostituiva un nastro, l'elaboratore ne utilizzava un altro.

Ciclo di elaborazione

Primo Sistema Operativo

- Un dispositivo digitale dedicato permetteva di stampare il contenuto di un nastro, senza impegnare l'elaboratore.
- Con tale procedimento, fin tanto che i nastri di ingresso contenevano dati da elaborare e i dati di uscita avevano spazio per ricevere risultati, l'elaboratore non aveva mai tempi morti.
- Tuttavia, per ottenere questo risultato, è stato necessario sviluppare *il primo rudimentale sistema operativo*.
- Il compito di tale software, caricato in memoria all'accensione dell'elaboratore, era eseguire la seguente sequenza di operazioni: Caricare in memoria principale il prossimo programma contenuto nell'unità nastro di ingresso. Avviare l'esecuzione del programma caricato.
- Quando il programma ha finito l'esecuzione, ricominciare il ciclo.

Il loader

Limiti di memoria

- Questo software di base era un semplice caricatore ("loader"), ma, pur occupando un po' di memoria principale, consentiva una notevole riduzione dei tempi morti, a condizione che i programmi fossero corretti.
- Si tenga presente che stiamo parlando di elaboratori con 16 o 32 KB (non GB né MB) di memoria principale, ma del costo orario pari a quello di decine di impiegati.

Il monitor residente

Come condividere il processore

- Un problema emerso con l'uso del caricatore era la presenza di errori nel software applicativo.
- Quando l'elaboratore era completamente gestito da un programma per volta, qualunque cosa facesse tale programma, non danneggiava gli altri programmi.
- Comunque, se un programma non aveva finito la sua esecuzione dopo il tempo previsto, l'operatore lo arrestava e passava al programma successivo.

Come gestire un pool di job

Problematiche e soluzioni

Usando il caricatore automatico, un programma poteva danneggiare gli altri programmi in molti modi, tra cui: Continuare a funzionare indefinitamente, o comunque per un tempo superiore a quello economicamente conveniente. Non leggere tutti i dati spettanti contenuti nei nastri di ingresso.

- Leggere dati o programmi non spettanti, contenuti nei nastri di ingresso.
- Scrivere sui nastri di uscita così tanti dati da riempirli, o comunque una quantità di dati superiore a quella economicamente conveniente.
- Sovrascrivere la parte di memoria principale in cui risiede il caricatore o i suoi dati.

Per rimediare a tali problemi, si adottarono le seguenti soluzioni:

- L'aggiunta di schede di controllo tra le sequenze di schede contenenti programmi e dati.
- La protezione hardware delle risorse (processore, memoria principale, unità a nastri).
- L'utilizzo di un sistema operativo più sofisticato, chiamato "monitor".

Le schede di controllo

Classificazione delle schede

- Siccome i primi elaboratori leggevano i dati direttamente dalle schede perforate, e siccome anche successivamente la preparazione iniziale dei dati avveniva perforando schede che poi venivano riversate su nastri, anche il contenuto dei nastri di ingresso era organizzato come una sequenza di record, ognuno corrispondente a una scheda.
- Per analogia, tali record venivano talvolta chiamati "schede".
- Il nastro di ingresso conteneva una sequenza di elaborazioni (dette anche "job"), ognuna strutturata nel seguente modo:
  - Record di inizio programma.
  - Numerosi record, ognuno contenente un'istruzione in linguaggio macchina.
  - Record di inizio dati di ingresso.
  - Numerosi record, ognuno contenente un blocco di dati.

Record speciali

Evoluzione in comandi

- I record di inizio programma e di inizio dati avevano un segno che li distingueva dagli altri, pertanto era possibile sapere con certezza quando finiva il programma e quando finivano i dati.
- In seguito, tali schede di controllo evolsero aggiungendo sempre più informazioni e diventarono veri linguaggi di comandi.
- Per esempio, inizialmente i programmi erano scritti solamente in linguaggio macchina, ma rapidamente si passò al linguaggio assemblativo, al FORTRAN e al COBOL, e poi ad altri linguaggi di programmazione.

Record speciali

Evoluzione in comandi

- Pertanto, la scheda di inizio programma doveva indicare anche in quale linguaggio era scritto il programma stesso.
- Leggendo la scheda si sapeva quale compilatore doveva essere utilizzato.
- Altre informazioni contenute nelle schede di controllo erano i requisiti del programma, cioè la quantità di memoria necessaria, la durata massima dell'elaborazione, e il numero massimo di record che verranno emessi come risultato.
- Avendo a disposizione tali informazioni, il sistema operativo poteva ottimizzare l'allocazione delle risorse e terminare le elaborazioni che superassero i requisiti dichiarati.

La protezione hardware

Multiutenza

- Con l'uso di un sistema operativo, l'elaboratore è utilizzato da almeno due utenti simultanei: l'operatore di sistema, che utilizza il sistema operativo per far funzionare in modo efficiente l'elaboratore; e l'utente che ha consegnato all'operatore il programma applicativo e i dati, che utilizza il programma applicativo per elaborare i propri dati.
- Nei sistemi multiprogrammati ci possono essere altri utenti simultanei.
- Se un programma contiene degli errori o comunque degli algoritmi che usano le risorse in modo non economicamente conveniente.

La protezione hardware

Multiutenza

- Siccome i programmi applicativi spesso contengono difetti, un sistema di elaborazione dati multiutente deve impedire ai programmi applicativi tali comportamenti devianti e dannosi.
- Il problema non si presenta per i difetti che può avere il sistema operativo stesso, in quanto, se il sistema operativo genera un malfunzionamento, comunque il funzionamento del software applicativo è compromesso.

La protezione hardware

Come evitare danni al sistema e ad altri programmi

- Quindi l'obiettivo è fare in modo che qualunque comportamento possa avere un programma applicativo, questo non possa danneggiare l'esecuzione del sistema operativo e di altri programmi.
- Questo era semplicemenți impossibile nei primi elaboratori negli anni '50 e nei primi microprocessori (quelli a 8 bit).
- Per renderlo possibile sono state apportate modifiche all'hardware dell'elaboratore.
- Tali accorgimenti tecnici prendono il nome di "protezione hardware".
- La protezione hardware è quindi l'insieme dei dispositivi elettronici che permettono al sistema operativo di assicurare che un programma applicativo non possa eseguire operazioni che danneggino l'esecuzione di altri programmi.

La protezione hardware

Come evitare danni al sistema e ad altri programmi

- La protezione hardware si basa essenzialmente su tre concetti:
  - il registro base,
  - il timer,
  - e la modalità del processore.
- Il registro base contiene l'indirizzo di memoria a partire dal quale viene caricato il programma applicativo.
- Le posizioni di memoria che precedono tale indirizzo sono quelle che contengono codice e dati del sistema operativo.

La protezione hardware

Funzionamento

- Tutte le volte che il processore emette un indirizzo sul bus indirizzi, per leggere un dato, per scrivere un dato, o per caricare un'istruzione, un circuito verifica che tale indirizzo non sia inferiore al registro base.
- In caso negativo viene generato un interrupt e il controllo torna al sistema operativo, che arresta il programma e passa al programma successivo.
- Il timer è un dispositivo che viene impostato da un'apposita istruzione ad un certo intervallo di tempo, e quando tale tempo scade, invia un interrupt che passa il controllo al sistema operativo.
- Prima di lanciare un programma applicativo, il sistema operativo imposta il timer al tempo massimo previsto per l'esecuzione.
- Se il programma applicativo non ha ancora terminato l'esecuzione dopo tale tempo, il timer passa il controllo al sistema operativo che arresta il programma e passa al programma successivo.

La protezione hardware

Le modalità su e user

- Il processore ha (almeno) due modalità di funzionamento:
- La modalità sistema (detta anche modalità protetta, o modalità supervisore, o modalità kernel).
- La modalità utente.

- La modalità sistema è quella priva di qualunque limitazione. Il processore è in modalità sistema solo quando è in esecuzione il codice del sistema operativo.
- La modalità utente è quella sottoposta a varie limitazioni. Il processore è in modalità utente quando è in esecuzione il codice applicativo, o anche il codice del sistema operativo che non richiede i privilegi propri della modalità sistema.

La protezione hardware

Azioni privilegiate

- Le istruzioni di alcuni tipi, dette istruzioni privilegiate, possono essere eseguite solo in modalità sistema.

  Esse sono quelle che svolgono i seguenti compiti:

  - Abilitazione e disabilitazione degli interrupt.
  - Ingresso/uscita (input/output).
  - Cambio di modalità del processore.
  - Impostazione del registro base.
  - Impostazione del timer.
- Il fatto che le istruzioni di ingresso/uscita siano privilegiate costringe i programmi applicativi a invocare il sistema operativo per effettuare qualunque operazione di ingresso/uscita.
- Questo genera un continuo passaggio del controllo tra il *codice applicativo* e il *codice di sistema*.

La protezione hardware

Azioni privilegiate

- Quando il sistema operativo lancia l'applicazione, pone il processore in modalità utente, e quando il controllo torna al sistema operativo a causa della termina dell'applicazione il processore torna in modalità sistema.
- Ma come fa l'applicazione a chiedere al sistema operativo di eseguire un'operatione di ingresso/uscita, dato che non può saltare direttamente nel codice del sistema operativo?

Chiamate di sistema

System Call

Per risolvere questo problema è stata introdotta una nuova istruzione, la "chiamata di sistema" (in inglese "system call" o "supervisor call").
Eseguendo tale istruzione il processore passa automaticamente in modalità sistema.

Il flusso di controllo del sistema operativo

Il flusso delle operazioni

- Tramite l'uso delle schede di controllo e dei dispositivi di protezione hardware, il sistema operativo può avere la seguente operatività:
  - Carica dal nastro magnetico di ingresso il primo record, verificando che si tratti di un record di inizio programma.
  - Carica eventuali altri record che completino la descrizione del programma, verificandone la correttezza formale.
  - Tra le informazioni lette, ci sono il tempo massimo previsto per l'esecuzione del programma, il numero di byte di memoria richiesti dal programma e dai suoi dati, e il numero massimo di record che potranno essere emessi.

Il flusso di controllo del sistema operativo

Il flusso delle operazioni

- Carica i record che contengono il programma, fino a quello che segnala la fine del programma e l'inizio dei dati.
- Imposta il registro base all'indirizzo a partire dal quale è stato caricato il programma.
- Imposta il timer al tempo previsto per l'elaborazione.
- Imposta il processore in modalità utente.
- Salta alla prima istruzione del programma applicativo.
- Il programma applicativo esegue le sue istruzioni.

Il flusso di controllo del sistema operativo

Il flusso delle operazioni

- Qualunque tentativo di eseguire istruzioni privilegiate o di accedere allo spazio di memoria riservato al sistema operativo provoca un interrupt.
- Anche il fatto che scatti il timer provoca un interrupt.
- L'interrupt provoca il passaggio del controllo alla routine del sistema operativo di gestione dell'interrupt, e l'impostazione del processore in modalità sistema.
- Tale routine emette in output un record di segnalazione dell'errore, provvede a leggere tutti i record di dati rimanenti fino al prossimo record di inizio programma e torna al punto 1.

Il flusso di controllo del sistema operativo

Il flusso di esecuzione

- Quando il programma applicativo vuole leggere uno o più record dal nastro magnetico di ingresso o scrivere uno o più record sul nastro magnetico di uscita, esegue un'istruzione di chiamata di sistema, impostando in alcuni registri il tipo di operazione (lettura o scrittura), il numero della periferica, il numero di record da elaborare, e l'indirizzo di memoria da utilizzare per il trasferimento.
- Tale istruzione imposta automaticamente il processore in modalità sistema.
- Il sistema operativo effettua la lettura o la scrittura e poi memorizza in un registro l'esito dell'operazione, reimposta il processore in modalità utente, e salta all'istruzione del programma successiva a quella di chiamata di sitema.
- Quando il programma ha terminato il proprio compito, torna il controllo al sistema operativo, che emette sul nastro un record di fine elaborazione, e riaparte il ciclo.

L'uso delle memorie secondarie ad accesso diretto

Le memorie

- La memoria principale dei primi elaboratori era molto piccola, in quanto di elevatissimo costo unitario, era possibile contenere in memoria, oltre al sistema operativo, solo piccoli programmi, che elaborassero pochi dati per volta.
- Tuttavia, alcuni problemi di calcolo richiedono che il programma o i dati abbiano dimensioni significative.
- Per poter eseguire questo tipo di elaborazioni su sistemi con poca memoria, si è pensato di caricare e scaricare dalla memoria i dati e le porzioni di programma di volta in volta richiesti.
- Siccome tali caricamenti e scaricamenti non seguono un ordine lineare, l'uso dei nastri magnetici, che sono ad accesso sequenziale, si è dimostrato inadeguato, e sono state inventate memoria secondarie magnetiche ad accesso diretto.

L'uso delle memorie secondarie ad accesso diretto

Le memorie

- Tali memorie erano i "tamburi" magnetici, oggi non più utilizzati, e i "dischi" magnetici, tuttora ampiamente utilizzati.
- I tali dispositivi avevano costi unitari (cioè per bit), intermedi tra quelli della memoria principale e quelli delle unità nastro.
- Pertanto avevano tipicamente anche capacità intermedie, dell'ordine di alcune centinaia di chilobyte, o di pochi megabyte.
- Inizialmente, il tamburo magnetico era a completa disposizione del programma applicativo.

Come far rientrare i dati nella memoria

Gli overlay

- Quando il programma veniva caricato dal sistema operativo, in realtà lo copiava sul tamburo, e poi da lì caricava in memoria i pezzi di programma che servivano di volta in volta.
- Ogni pezzo era detto "overlay" (inglese per "copertura"), perché si sovrapponeva ad una precedente porzione di codice. Un overlay era costituito da una o più routine.
- Quando da una routine si chiamava una routine appartenente a un altro overlay, il controllo passava al sistema operativo che caricava dal tamburo l'overlay contenente la routine chiamata, sostituendo in memoria principale un altro overlay.
- Ovviamente il programmatore doveva organizzare il programma in modo da evitare eccessivi cambi di overlay, perché tali operazioni rallentavano l'elaborazione.

Aumento delle memorie

Dati non localizzati

- Con il crescere delle dimensioni delle memorie secondarie, si pensò di conservarvi dei dati persistenti, cioè che fossero riutilizzabili anche dopo l'esecuzione di un programma.
- Un tipico utilizzo dei dati persistenti su dispositivi ad accesso diretto era la ricerca di informazioni.
- Per esempio, se un archivio su nastro contiene tutti i saldi dei conti correnti di una banca in ordine di numero di conto, e un altro archivio contiene tutti i movimenti della giornata in ordine temporale, e si vuole aggiornare i saldi dei conti correnti, si può procedere nel seguente modo:
  - Si scandiscono i saldi dei conti correnti.
  - Per ogni conto corrente si cercano sull'archivio dei movimenti tutti i movimenti che lo riguardano e si aggiorna il saldo.

Aumento delle memorie

Dati non localizzati

Oppure nel seguente modo:

- Si scandiscono i movimenti dei conti correnti.
- Per ogni movimento, si cerca nell'archivio dei saldi il record relativo e lo si aggiorna.

In entrambi i casi, supponendo che sia i record dei movimenti che i record dei saldi siano così tanti da non essere contenuti in memoria, si devono effettuare un gran numero di ricerche.
Usando un nastro magnetico, tali ricerche hanno un costo proibitivo.
Invece, si può caricare su un disco magnetico l'elenco dei saldi, lasciare i movimenti su nastro, e usare il secondo dei due algoritmi citati sopra.
La ricerca su disco magnetico di un record contenente un saldo può essere effettuata cercando con un metodo dicotomico, eventualmente agevolato da un piccolo indice mantenuto in memoria principale.

Memorie ad accesso diretto

Problematiche di condivisione della memoria secondaria

- L'uso di memorie magnetiche persistenti ad accesso diretto faceva però sorgere il seguente problema.
- Se l'elaboratore è usato in sequenza da numerosi programmi, e non ci sono tanti dischi quanti sono i programmi, si deve condividere tra più programmi la stessa unità disco con i suoi dati.
- Se i programmi devono anche condividere i dati basta che i programmatori si mettano d'accordo, ma se ogni programma ha i suoi dati, non è ragionevole che ogni programmatore debba conoscere quali parti del disco sono o saranno utilizzate da altri programmi, con il rischio che un errore di programmazione possa compromettere il funzionamento di altri programmi.
- La gestione del disco condiviso è chiaramente un compito del sistema operativo.
- Siccome gli archivi su disco si chiamano anche "file", il sottosistema del sistema operativo che si occupa della gestione degli archivi su disco viene chiamato "file system".

Il file system

Accesso ai dati tramite file system

- Per condividere una unità disco tra vari programmi, assicurando che nessun programma possa danneggiare le funzionalità degli altri programmi è ovviamente necessario che tutti gli accessi all'unità disco siano mediati dal sistema operativo,
- cioè che l'unico modo incui un programma possa leggere o scrivere dati su un disco sia chiedendo al sistema operativo di svolgere tale servizio per conto del programma applicativo.

Livelli di protezione

Protezione

- In realtà, i primi sistemi operativi per microcomputer, come CPM e MS-DOS, non avevano alcuna protezione hardware, neanche per l'accesso all'unità disco, e tuttavia erano abbastanza robusti.
- Questo era dovuto al fatto che la stragrande maggioranza dei programmatori si atteneva alle raccomandazioni di accedere alle unità disco solo tramite richeste al sistema operativo.
- Tuttavia tali sistemi non protetti erano sistemi monoutente non critici per la vita.
- Invece, un sistema che debba garantire un'elevata sicurezza ha bisogno di protezione hardware anche per l'accesso alle unità disco.
- In realtà, esistono vari livelli di protezione, per vari livelli di rischi.

Livelli di protezione

Livelli

- Il primo livello, come abbiamo già detto, consiste nell'impedire ai programmi applicativi l'accesso diretto alle unità disco.
- Un secondo livello consiste nell'impedire ai programmi applicativi l'accesso a parti dell'unità disco per cui l'utente del programma non è autorizzato. Questo è il livello dei "diritti d'accesso".
- Un terzo livello consiste nell'impedire accessi non autorizzati neanche smontando il disco e collegandolo a un altro computer, per proteggere dati segreti (o sensibili come si dice oggi). Questo si ottiene tramite la crittografia in tempo reale.
- Un quarto livello consiste nell'impedire a un programma applicativo di occupare troppo spazio su disco, con il rischio di impedire il funzionamento di altri programmi per l'esaurimento dello spazio libero. Questo si ottiene tramite le "quote" di disco.
- Ulteriori livelli sono quelli di "tolleranza ai guasti" (in inglese, "fault tolerance"), che consiste nell'utilizzare sistemi ridondanti che continuino a funzionare anche se ci sono guasti nell'hardware.