Linguaggi di programmazione

Storia dell'Informatica e del calcolo automatico

Università Federico II
Corso S.I.C.S.I VIII ciclo
Mariarosaria Napolitano
Prof. Aniello Murano

L´Evoluzione dei linguaggi di programmazione nella storia

1

Sommario

- Le Origini della Programmazione
- Alcuni concetti fondamentali sui Linguaggi di programmazione
- Linguaggi ai vari livelli di Astrazione
- Carrellata Storica sull´evoluzione dei Linguaggi
- Paradigmi di programmazione
- Alcuni tra i linguaggio più famosi.

2

Un tuffo nella Preistoria

Il linguaggio nasce come bisogno primordiale dell'uomo di trovare una forma di comunicazione con l'ambiente esterno (affonda le sue radici nella preistoria).

Per capire l'importanza del linguaggio è necessario fare un salto nel passato.

Ma alle origine di tutti i tempi indovinate chi troviamo?

……..

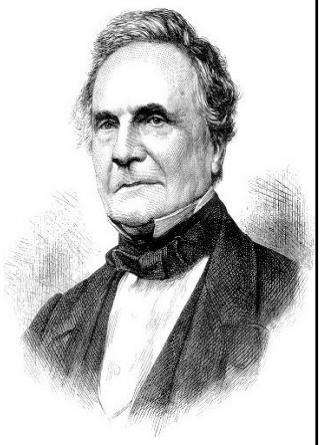
3

No…. non sono Adamo ed Eva che comunque diedero vita ad una forma di linguaggio anche se (fatto di gesti e di suoni) non evoluto per poter comunicare fra loro.

Ma troviamo coloro che gettarono le basi dell'era informatica: Charles Babbage e Ada Byron Lovelable

4

Il primo Vagito



Ebbene si all'origine dei nostri tempi……
troviamo proprio lui……
Charles Babbage (1791 - 1871),
matematico, filosofo e proto-informatico,
è considerato il "padre" dei calcolatori programmabili. Tentò di costruire
il primo computer (1843) malgrado
la tecnologia dell'epoca non fosse adeguata,
egli intuì che per far funzionare il computer avrebbe avuto bisogno di un programma.

5

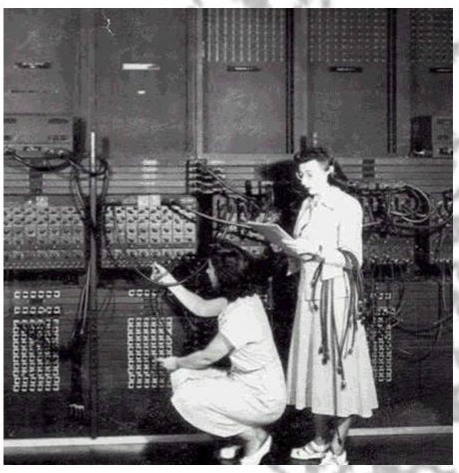
Fu così che gli venne in aiuto Alda Byron,Lovelance, "(1815 – 1852) la prima programmatrice che diede il nome al famoso linguaggio di programmazione ADA. Con l'intento di mostrare le forti potenzialità della Macchina Analitica che Babbage progettò nel 1840, Ada scrisse un programma capace di indicare alla macchina come realizzare un calcolo dei numeri di Bernouilli; questo programma viene considerato il primo software della storia.

Nella **macchina analitica,** l'idea di esecuzione automatica
delle istruzioni era affidata alle schede perforate, proprio
come nel telaio Jacquard: la macchina consisteva di una
Unità di input sotto forma di banco per schede perforate
E di due parti fondamentali che erano la memoria (store)
e l'unità di calcolo (mill). La macchina analitica, viene considerata
il **primo computer al mondo.**
Macchine Analogiche come queste di Babbage e Lady Lovelace
ossia funzionanti mediante il movimento di parti meccaniche,
(come ingranaggi, ruote), furono costruite agli inizi del
XIX secolo.

Anche se tutti i progetti estremamente ingegnosi dei due matematici non furono realizzati per la tecnologia dell'epoca, essi furono comunque considerati i fondatori del moderno computer.

6

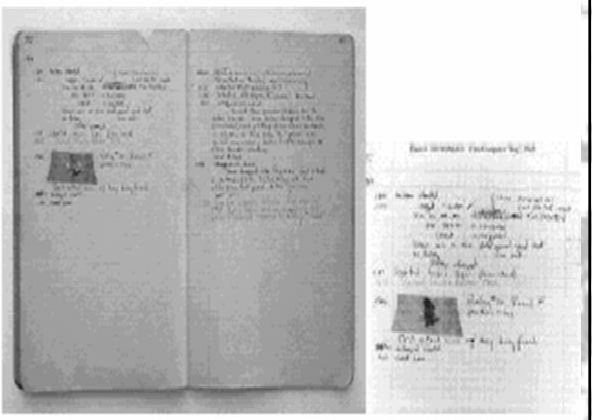
- I primi computer moderni (1945-1955) (senza parti di tipo meccanico) sfruttavano la tecnologia a valvole ed anche se erano poco potenti ed affidabili, erano di dimensioni enormi e potevano occupare intere stanze
- Un team di specialisti costruiva, manteneva e programmava la macchina (appunto in linguaggio macchina)
- **Non c'era sistema operativo**, nemmeno l'assembler, bisognava eseguire un programma per volta (generalmente tabelle matematiche) sotto la supervisione del programmatore



7

- Erano lentissimi e molto costosi
- Potevano permetterseli soltanto governi, grossi centri di calcolo o Università
- Erano molto inaffidabili, in quanto le valvole che li componevano si rompevano spesso
- Il programma da eseguire veniva inserito ad ogni esecuzione in codice binario attraverso dei primitivi lettori di schede perforate e dopo alcune ore il risultato veniva inviato ad una stampante

Log book with a bug



In 1947, engineers working on the Mark II computer at Harvard University found a moth stuck in one of the components.

8

Finalmente i primi passi

- Nel (1943-1945) si incominciano a costruire le prime macchine per il calcolo automatico che fossero programmabili e che prendevano come modello quello della Macchina di Von Neuman .

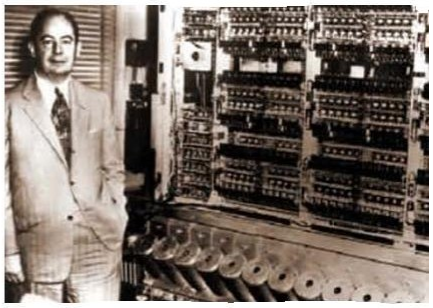
- Von Neuman progettò quella che diventerà la prima architettura di base di un vero computer.

- Il programma era caricato in memoria come i dati, specificando una zona di memoria dove caricare le istruzioni.

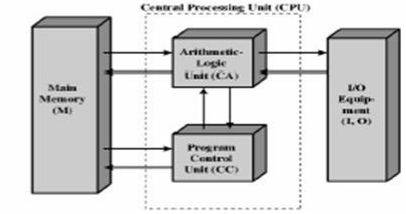
9

Macchina di Von Neuman

- Programma memorizzabile come i dati
- Istruzioni in memoria: decidere il programma specificando una porzione di memoria
- Idea di John von Neumann (consulente ENIAC)
- Proposta di nuovo calcolatore (1945): EDVAC (Electronic Discrete Variable Computer)
- Nuovo elaboratore completo nel 1952 (IAS, presso Institute for Advanced Studies, Princeton)



Struttura della macchina di von Neumann



Una breve pausa……

Per introdurre alcuni concetti fondamentali sui linguaggi di programmazione.

11

Linguaggi…. ma solo di programmazione

Prima di poter definire e parlare di linguaggi di programmazione abbiamo bisogno di dare alcune definizioni:

come quello di **Algoritmo e Programma**.

12

Algoritmi & Programma

- Un algoritmo è una sequenza finita di passi elementari per la risoluzione di un problema, e deve essere finito, non ambiguo, deterministico e generale.
- Dato quindi un algoritmo, un programma è la sua descrizione in un particolare linguaggio di programmazione.

13

La nozione di linguaggio

A questo punto possiamo dire che un linguaggio di programmazione è una notazione formale che può essere usata per descrivere algoritmi, avente due aspetti fondamentali:

- Sintassi
- Semantica

14

SINTASSI & SEMANTICA

- **Sintassi**: **l'insieme di regole formali per la scrittura di programmi** in un linguaggio, che dettano le ***modalità per costruire frasi corrette*** nel linguaggio stesso.
- **Semantica**: **l'insieme dei significati** da attribuire alle frasi (sintatticamente corrette) costruite nel linguaggio.

NB: una frase può essere **sintatticamente corretta** e tuttavia ***non avere significato*!**

15

Linguaggi a vari livelli di astrazione

– **Linguaggio Macchina:**
implica la conoscenza dei metodi di rappresentazione delle informazioni utilizzati.

– **Linguaggio Assembler:**
implica la conoscenza dettagliata delle caratteristiche della macchina (registri, dimensioni dati, set di istruzioni)

- **Linguaggi di Alto Livello:**
Il programmatore può astrarre dai dettagli legati all'architettura ed esprimere i propri algoritmi in modo simbolico

Sono indipendenti dalla macchina hardware sottostante
ASTRAZIONE

16

Alcuni Esempi

- **Linguaggio Macchina:**

```
0100 0000 0000 1000
0100 0000 0000 1001
0000 0000 0000 1000
```

Difficile leggere e capire un programma scritto in forma binaria

- **Linguaggio Assembler:**

```
… LOADA H
  LOADB Z
  ADD
  …
```

Le istruzioni corrispondono univocamente a quelle macchina, ma vengono espresse tramite nomi simbolici (parole chiave).

- **Linguaggi di Alto Livello:**

```
main()
{ int A;
  scanf("%d",&A);
  if (A==0) {…}
…}
```

Sono indipendenti dalla macchina

17

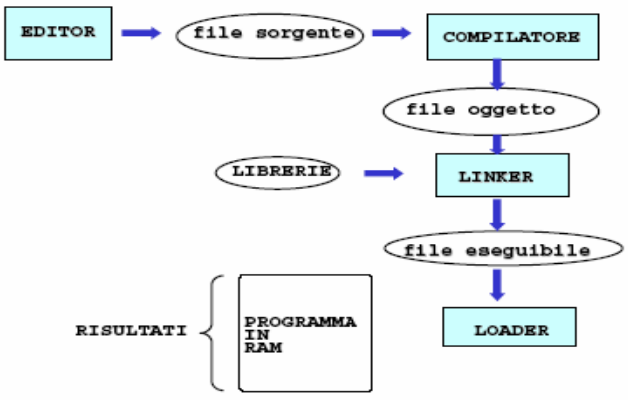
Fasi di sviluppo di un programma

Le fasi di preparazione di un programma scritto in un linguaggio ad alto livello sono:

- **Codifica**: In particolare, l'editor **consente di scrivere il programma sorgente.**
- **Compilazione**: opera la **traduzione di un programma sorgente** (scritto in un linguaggio adalto livello) **in un programma oggetto** direttamente eseguibile dal calcolatore.

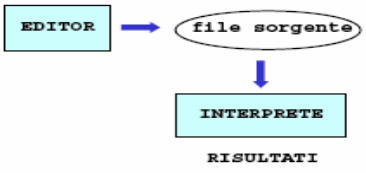
  PRIMA si traduce **tutto il programma POI** si esegue **la versione tradotta**.
- **Linking**:((collegatore) nel caso in cui la costruzione del programma oggetto richieda l'unione di **più moduli** (compilati separatamente),il linker provvede a **collegarli** formando un unico programma eseguibile**.**
- **Debugger:** consente di **eseguire passo- passo** un programma, **controllando via via quel che succede**, al fine di **scoprire ed eliminare errori** non rilevati in fase di compilazione.

APPROCCIO COMPILATO: SCHEMA



19

APPROCCIO INTERPRETATO: SCHEMA



20

Viaggio alla velocità della luce

…Per analizzare l ‘evoluzione dei linguaggi con riferimento ai processi di sviluppo ..

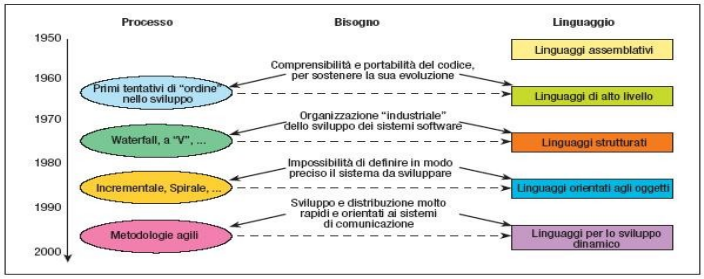


FIGURA 1
Evoluzione dei linguaggi

21

Carellata Storica

Un breve riassunto ……..

Nei primi calcolatori, il programma era definito da circuiti elettrici:veri e propri collegamenti fisici. Si passò poi ai codici binari e ai linguaggi assemblativi a essi associati, ciascuno dei quali si riferiva in modo quasi completamente univoco a specifici elaboratori. Cambiando elaboratore, di solito, cambiava anche il linguaggio.

22

Dagli anni 50 agli anni 60

Nasce la programmazione ad Alto Livello

Primo tentativo di ordine nello sviluppo

Il processo di sviluppo che determina l'affermarsi della programmazione ad alto livello fu motivato dal bisogno di avere programmi scritti in formalismi comprensibili da un largo numero di sviluppatori e non dipendenti da architetture specifiche.

Nascono i primi processi di **programmazione semplice**

Nascono i linguaggi orientati al problema con maggiore attenzione al modello di computazione.

Era il tempo delle :

- astrazioni con formule matematiche **FORTRAN (FORmula TRANslation)**
- astrazioni logiche con **il LISP** (LISt Processing language)
- delle transazioni economiche **COBOL (Common Business- Oriented Language)** introduce il concetto di File e di descrizione dei dati

Dagli anni 60 a 70

Nasce la programmazione strutturata

Il modello di sviluppo di riferimento fu chiamato **waterfall**, ovvero, a cascata.

La diffusione degli strumenti informatici nonché l'ingrandirsi dei sistemi da sviluppare, impose la definizione di un processo di sviluppo che si basava **su modelli modulari,** concentrati sulla suddivisione del lavoro in parti e la successiva specializzazione dei compiti.

Si diffusero le prime **metodologie di programmazione** che diedero vita al concetto di **programmazione strutturata**

Linguaggi che fanno la loro apparizione in questo periodo sono :

- **Pascal** come possibile risposta alla necessità di programmare con un metodo, usato per scopi didattici.
- **C** linguaggio ad alto livello con visibilità e accesso alla macchina
- **Prolog** linguaggio basato sulla logica , non convenzionale, diventato di nicchia.

In particolare sarà con i linguaggi di Modula 2 e con l'ADA che la programmazione strutturata acquisisce la sua completa espressione.

Gli anni 80

Qualcosa non và

Verso la metà degli anni 80, il "modello di sviluppo a cascata" cominciò a evidenziare i propri limiti, soprattutto per l'impossibilità di definire in modo completo e corretto fin dall'inizio le caratteristiche che il sistema software avrebbe avuto alla fine.

Modello incrementale Spirale

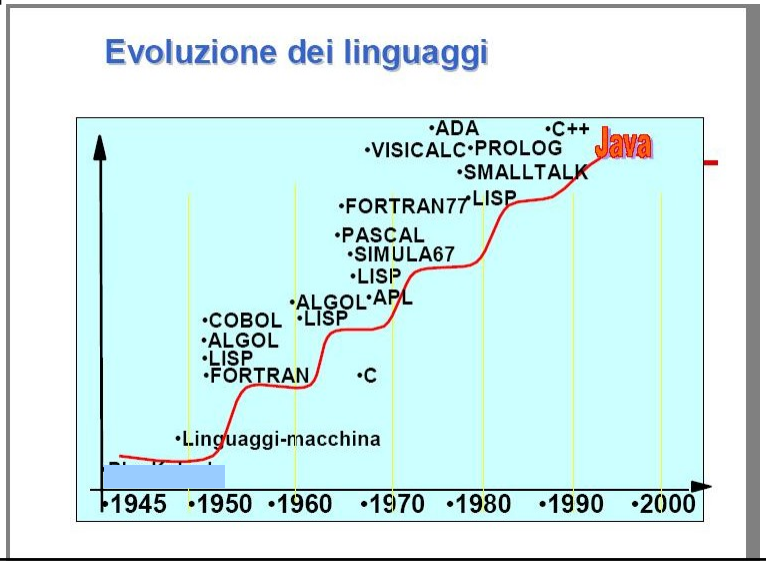
La comprensione di questa problematica permise un salto di qualità nelle modalità di sviluppo del software.

Inizia l'epoca dei linguaggi orientati agli oggetti

Si affermano così i seguenti linguaggi che nascono dall'esigenza di maggiore modularità e di astrazione
C++,Smaltalk, Object- C

25

Per non fare torto a nessuno … qui ci sono tutti



26

Ora che sono quasi ..tutti nati possiamo parlare di…..

Paradigmi di programmazione

Forniscono la filosofia e la metodologia con cui si scrivono i programmi.

I linguaggi devono consentire ma soprattutto spingere all'adozione di un particolare paradigma

- Procedurale
- Funzionale
- Orientato agli oggetti

27

Paradigma Procedurale

Enfasi sulla soluzione algoritmica dei problemi
Aderenti al modello della macchina di von Neumann
Che si basa su:

- Concetto di istruzione
- Concetto di sequenzialità e iterazione
- Molto efficienti
- Ha mostrato limiti nello sviluppo e mantenimento di sw complessi.
- I linguaggi **imperativi** che hanno solide radici nell'architettura di von Neumann **sono** : **Pascal**, **C**

28

Paradigma funzionale

Primo tentativo di non rifarsi al modello di macchina di von Neumann:

- La computazione avviene tramite espressioni e funzioni che processano dati di input e forniscono nuovi valori di output.
- Ogni funzione è un modulo a sé dipendente unicamente dal valore dei suoi argomenti.
- L'effetto globale è ottenuto concatenando opportunamente funzioni anche richiamando sé stesse (ricorsione)
- Modello che si rifà alla teoria delle funzioni ricorsive
- Scarso supporto ai costrutti di ripetizione tramite iterazione
- Un tipico esempio di linguaggio funzionale è il **Lisp.**

29

Paradigma della programmazione ad oggetti

Storicamente, le prime idee alla base di questo paradigma sono sorte con il Simula '67, un linguaggio di simulazione discreta di scuola scandinava, e sono state poi riscoperte negli USA negli anni '70-'80 con il linguaggio Smalltalk. Buona parte del gergo della OOP si deve a questi due linguaggi.

Dalla metà degli anni '80 esiste un generale accordo sulle caratteristiche che devono essere presenti in un linguaggio per poterlo classificare come OOPL:

insieme di operazioni astratte, associate ad un tipo ADT (Abstract Data Type)
stato dell'oggetto
ereditarietà

Il Paradigma ad oggetti propone il superamento del dualismo: dati-procedure, definendo alcuni concetti:

Oggetto: come un'entità che incorpora sia dati che azioni, dove i dati associati all'oggetto vengono chiamati proprietà e le azioni vengono chiamate metodi.Esso è un'entità dotata di Identità, Stato e Comportamento.

Classe: può essere definita come uno schema di creazione che determina univocamente l'identità di un oggetto, stabilendone lo stato iniziale. Essa consente quindi di istanziare un oggetto con tutte le sue caratteristiche.

L'Ereditarietà stabilisce infine che il linguaggio deve prevedere un meccanismo per definire delle relazioni IS-A ("è-un") tra classi di oggetti. Questo meccanismo consente in definitiva il **riuso** e il **polimorfismo,** due dei principali fattori alla base del successo della OOP.

Anni 90
La naturale evoluzione della OOP: C++

Nel 1979, un danese Bjarne. Stroustrup pensò di accoppiare C e oggetti. Il risultato fu C++, ancora oggi uno tra i linguaggi più popolari e sofisticati.

La filosofia del C++ è la stessa del C: Niente è vietato.Niente deve impedire al programmatore di abbandonare ogni prudenza per scrivere un programma velocissimo, o di rompere tutte le barriere di sicurezza e "spaccare il bit", lavorando direttamente in memoria. La compatibilità all'indietro con il C ha permesso a C++ di sedere sul trono dei linguaggi "seri", per anni fino all'arrivo di Java. C++ è oggi un linguaggio che invecchia. E' molto più specialistico di qualche anno fa, e per alcuni settori non ci sono ancora alternative alla sua potenza bruta ed alla sua capacità di lavorare a bassissimo livello.

Perché imparlo

Per molti progetti è ancora indispensabile un linguaggio a basso livello che compila in codice nativo. E' ancora uno dei pochi liguaggi che consente un contrllo completo di tutto il flusso del codice.Se dovete scrivere un driver hardware o un sistema operativo , rimboccatevi le maniche: vi toccherà affrontare uno dei linguaggi più complessi.

Perché ignorarlo: potreste usarlo per anni senza mai dominarlo a fondo, e le sue potenzialità di produrre bug devastanti ed estremamente insidiosi sono pressochè **illimitate.**

31

Gli anni ruggenti : 2000

Limiti dell'approccio ad oggetti

La programmazione ad oggetti da un lato apportò un miglioramento sostanziale dello sviluppo del codice . Di contro però richiese una competenza decisamente superiore da parte degli sviluppatori. Quindi un'adeguata formazione in un linguaggio così complesso non era semplice e richiedeva insieme allo sviluppo di un progetto dei tempi molto lunghi.

Intanto 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 dell'interoperabilità tra sistemi 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, fino ad arrivare a linguaggi di programmazione completi, dove i più famosi sono: **Java e C#.**

È questo il periodo del-l'avvento delle **metodologie agili di program-mazione**, che danno sostanziale supporto alla dinamicità del sistema da sviluppare.

32

Il brutto anatroccolo diventa cigno: Java

Origini del linguaggio

Il linguaggio è nato dal linguaggio OAK, usato inizialmente dalla SUN MicroSystem per la gestione di hardware come la TV via cavo. Il progetto fu però abbandonato per scarse richieste hardware e la SUN, pensò di sfruttare allora il linguaggio in ambito INTERNET.

James Goslin produsse le prime versioni del linguaggio Java nel 1994.

Il nome J.A.V.A. sembra sia nato in ambito MARKETING come scherzoso acronimo: "Just Another Vague Acronym".

Come mai Java è oggi uno dei linguaggi di programmazione più popolare e richiesto?
La risposta è sui server.

Quindi lo stesso programma Java può girare su qualsiasi computer abbia una JVM- il che vuol dire che gira anche sul tuo.

Quello che all'inizio veniva spesso liquidato come un giocattolo lento e inaffidabile è diventato la base dei Sistemi Aziendali che divorano pagine Web, transazioni bancarie, informazioni finanziarie…

L'asso nella manica di Java è stata la sua "macchina virtuale". Il compilatore Java non genera codice nativo per una determinata macchina come fa il C++. Il computer che fa girare Java è a sua volta un programma, la **Java Virtual Machine.**

33

Il Motto di Java: Write once run everywhere

Perché impararlo:

Java è la scelta piu' ovvia per chi col computer ci lavora. A parte il mondo Microsoft non esistono ancora alternative valide a Java per buona parte dei grandi progetti. Le risorse su Internet, comprese le librerie "Open Source" e gli IDE Eclipse e NetBeans, sono innumerevoli, di altissima qualità e quasi sempre belle gratis.

Caratteristiche di Java

Nel rapporto "The Java Language Environment" di James Goslin si dice che Java è "orientato agli oggetti, stabile, sicuro, neutro rispetto all'architettura, ad alte prestazioni, interpretato, multi-threading e dinamico.

Java è neutrale rispetto all'architettura perchè il suo codice è eseguibile su qualsiasi CPU e portabile su qualsiasi Sistema Operativo. Questo è possibile perchè non è compilato, ma interpretato! Infatti Java viene compilato per ottenere non l'obj classico, ma un formato binario intermedio standard - definito bytecode. Ogni piattaforma ha una propria "Virtual.Machine" (VM), ovvero un interprete di bytecode

Perché ignorarlo: Da molto tempo ormai Java non è piu' il linguaggio giovane e ribelle ma si avvia verso una sonnolente mezza età. E se è vero che la domanda delle Aziende resta alta, è anche vero che tutti compreso mia nonna e il mio salumiere si vendono ormai per programmatori Java.

Il fratello gemello: C#

A Microsoft, questa faccenda di Java non è mai andata giu',
Ma come?
Tutta questa fatica per dominare il mercato con Windows e improvvisamente arriva un linguaggio che tratta il Sistema Operativo con indifferenza?
E' per questo che nel 2001, l'azienda di Zio Bill pubblica la sua alternativa a Java:
un linguaggio quasi identico a Java, con una Virtual Machine come Java (un sistema che Microsoft ha chiamato framework . NET) ma inizialmente solo per Windows.
Che C# sia un clone di Java non c'è dubbio, ma a volte l'allievo supera il maestro.
Così C# non esita a sperimentare funzionalità nuove;
e mentre Java si preoccupa della sua fama di linguaggio "serio", C# non si vergogna di offrire semplificazioni in campi essenziali come le interfacce grafiche ed i Data Base.

Perché impararlo

In generale è un po' piu' produttivo di Java, almeno per i piccoli progetti

Perché ignorarlo

E' come Java ma senza il dinamico mondo "open source" che circonda Java. Quando si tratta di compatibilità multi-piattaforma, Java resta inarrivabile.

35

QUALITA' DEI LINGUAGGI

Un buon linguaggio deve consentire lo sviluppo di un software che sia:

- **AFFIDABILE**
  - leggibile e scrivibile, semplice, sicuro (ad esempio non deve essere possibile fare accesso ad aree di memoria protette)
  - robusto (devo avere dei costrutti che permettono al linguaggio di prevedere possibili malfunzionamenti, come ad esempio con la gestione delle eccezioni)
- **MANUTENIBILE**
  - modulare (per permettere di poter modificare solo alcune parti senza dover ridisegnare il software)
- **EFFICIENTE**
  - Anche se l'efficienza della programmazione è più importante

36

Quali conclusioni trarre?

Dopo questa carrellata storica sui vari linguaggi **linguaggi di programmazione** passati e presenti, la domanda sorge spontanea:

"Ma quale linguaggio è meglio usare C o Pascal? C++ o ADA? Java o C#?"

Generazioni di informatici hanno combattuto serie battaglie per difendere il "loro" linguaggio di programmazione, adducendo le più svariate motivazioni per difendere la loro scelta.

In realtà non esiste il linguaggio migliore in assoluto in quanto non esiste il linguaggio ideale che và bene per tutti gli obiettivi.

Secondo il famoso "Manuale del buon programmatore" :

Qualsiasi linguaggio si possa imparare e conoscere non bisogna cadere nella trappola di usare solo quello per i prossimi vent'anni.

Un programmatore completo deve conoscere almeno un linguaggio statico, uno dinamico ed uno di markup.

Un obiettivo ambizioso ma come tutti i grandi viaggi iniziano con il primo passo.

Bibliografia

[1] Backus J.W.: *Automatic Programming: Properties and Performances of FORTRAN Systems I and II*. Atti del Symposium of Mechanisation of Thought Processes, Teddington, UK, 1958.

[2] Backus J.W.: The History of FORTRAN I, II, and III. In: *IEEE Annals of the History of Computing*, Vol. 20, n. 4, 1998, p. 68-78.

[3] Banham R., Newman P.: *The Ford Century: Ford Motor Company and the Innovations That Shaped the World*. Artisan Sales, 2002.

[4] Bruy M., Krieg-Brückner B.: Derivation of Invariant Assertions During Program Development by Transformation. *Transactions on Programming Languages and Systems*, Vol. 2, n. 3, 1980, p. 321-337.

[5] Budd T.: *An Introduction to Object Oriented Programming*. Addison Wesley, 1991.

[6] Cox B.: *Object Oriented Programming – An Evolutionary Approach*. Addison Wesley, 1986.

[7] Chidamber S.R., Kemerer C.F.: A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering*, Vol. 20, n. 6, June 1994, p. 476-493.

[8] Church A.: *The Calculi of Lambda-Conversion*. Princeton University Press, Princeton, New Jersey, 1941.

[9] Dahl O.-J., Nygaard K.: SIMULA –An Algol Based Simulation Language. *Communications of the ACM*, Vol. 9, n. 9, 1966, p. 671-678.

[10] Dijkstra E.W.: *Primer of Algol 60 Programming*. Academic Press, 1962.

[11] Dodrill G.: *Coronado Enterprises Modula-2 Tutor*. 1987, URL: http://www.modula2.org/tutor/

[12] Ellis T., Miles R.: *A Structured Approach to FORTRAN 77 Programming*. Addison-Wesley Publishing, 1983.

[13] Ellis T., Miles R.: *FORTRAN 90 Programming*. Addison-Wesley Publishing, 1994.

[14] Feuer A.R., Gehani N.H.: Comparison of the Programming Languages C and Pascal. *ACM Computing Surveys*, January, 1982.

[15] Gehani N.H.: *Unix Ada Programming*. Prentice Hall, 1985.

[16] Goldberg A., Robson D.: *Smalltalk 80: The Language and Its Implementation*. Addison Wesley, 1983.

[17] Graham P.: *History of FORTRAN*, URL: http://www.paulgraham.com/history.html

[18] Hewitt C., Bishop P., Steiger R.: *A Universal Modular Actor Formalism*. Atti della 3rd International Joint Conference on Artificial Intelligence, Stanford, CA, 1973.

[19] Hansen P.B.: The Programming Language Concurrent Pascal. *IEEE Transactions on Software Engineering*, Vol. 1, n. 2, 1975, p. 199-207.

[20] Jensen K., Wirth N.: *PASCAL User Manual and Report*. Springer, 1975.

[21] Kernighan B.W., Ritchie D: *The C Programming Language*. Prentice Hall, 1978.

[22] Kruchten P.: *The Rational Unified Process: An Introduction*. Addison Wesley, 1998.

[23] Liebowitz S.J., StMargolis S.E.: Path Dependence, Lock in, and History. *Journal of Law, Economics, and Organization*; 1995, anche disponibile on-line all'URL: http://www.pub.utdallas.edu/~liebowit/paths.html

[24] McCarthy J.: *A Revised Version of MAPLIST*. MIT AI Lab., AI Memo n. 2, Cambridge, September 1958.

[25] McCracken D.D.: *A Guide to Cobol Programming*. Wiley, 1963.

[26] Meyer B.: *Eiffel. The Language*. Prentice Hall, 1992.

[27] Murtagh J.L., Hamilton J.A.: A comparison of Ada and Pascal in an introductory computer science course. *ACM SIGAda Ada Letters*, Vol. 18, n. 6, 1998.

[28] Reid J.: *The New Features of Fortran 2000*. URL: www.ukhec.ac.uk/publications/reports/N1507.pdf

[29] Rumbaugh J., Jacobson I., Booch G.: *The UML Reference Manual*. Addison Wesley, 1998.

[30] Sethi R.: *Programming Languages: Concepts and Constructs*. Addison Wesley, 1996.

[31] Stroustrup B.: *The C++ Programming Language*. Addison Wesley, 1996.

[32] Tsaptsinos D., Davies R., Rea A.: *Fortran 90 – An Introduction to the language for beginners*. URL: http://www.pcc.qub.ac.uk/tec/courses/f90/ohp/header_ohMIF_1.html

[33] US Department of Defence: *Department of Defense Requirements for High Order Computer Programming Languages: Steelman*. 1978.

[34] Wheeler D.A.: *Ada, C, C++, and Java vs. The Steelman*. Ada Letters, July/August, 1997.

[35] Wirth N.: *Programming in Modula-2*. Springer Verlag, 1982.

GIANCARLO SUCCI è ordinario di Informatica e direttore del Center for Applied Software Engineering presso la Libera Università di Bolzano, dove si occupa di metodologie agili per lo sviluppo software, metriche software, ingegneria del software empirica, strumenti di e-learning nell'industria software, linee di prodotto software. È stato ricercatore presso l'Università di Trento, professore associato alla University of Calgary e quindi professore ordinario alla University of Alberta, dove ha co-diretto l'Alberta Software Engineering Research Consortium. È autore di più di 150 articoli scientifici e di 5 libri.
e-mail: giancarlo.succi@unibz.it

38


That ́s All Folks
39

This document was created with Win2PDF available at http://www.win2pdf.com.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.
This page will not be added after purchasing Win2PDF.