Il Dizionarietto del Diavolo del DP

Cento Leggi ribalde e Definizioni "cattive" dei termini del DP

di Emilio C. Porcelli

Prefazione

Di tutti i mondi possibili quello del DP è, almeno all'apparenza, uno dei più seri, anzi seriosi. Ma l'apparenza, si sa, inganna. Per convincersene basterebbe ascoltare certe conversazioni tra addetti ai lavori o spigolare pubblicazioni del tutto insospettati: sono spesso percorse da una vena di divertito cinismo che si esprime sotto forma di definizioni "cattive" (spacciate per Leggi, Princìpi, Teoremi e Corollari) e di battute fulminanti dietro le quali si nascondono spesso inquietanti verità. Abbiamo pensato di raccoglierne alcune in un nostro Dizionarietto del Diavolo.

Il Dizionarietto vanta illustri precedenti. Nel 1981 Stan Kelly-Bootle ha infatti pubblicato per i tipi di un austero editore, McGraw-Hill, un *The Devil's DP Dictionary* le cui voci (ma parlando di un dizionario sarebbe meglio dire lemmi) sono altrettante definizioni ribalde di termini DP. Un esempio per tutti: "Default - il vano tentativo di evitare gli errori attraverso l'inattività".

Ma anche il Dizionario di Kelly-Bootle ha dei precedenti, e questa volta illustri: sono *The Devil's Dictionary*, una raccolta di aforismi sulfurei pubblicato nel 1911 a New York da Ambrose Pierce e oggi considerato nel mondo anglosassone un vero e proprio piccolo classico.

Le nostre ambizioni sono assai più modeste: il Dizionarietto conta infatti solo 100 voci. Alcune, come le celeberrime Leggi di Murphy pubblicate a più riprese in Italia da Longanesi, nascono al di fuori del mondo del DP, anche se in esso trovano larga applicazione. Altre, e sono la maggioranza, sono state invece formulate al suo interno. Tra queste ultime non mancano le cattiverie "d'autore": qualche volta anche grandi firme dell'Informatica come DeMarco, Myers o Yourdon si lasciano un pò andare ...

Milano, gennaio 2002

A

Abend [dal tedesco *guten Abend* e cioè "Buona Sera"] - E' il saluto ironico con il quale un sistema software si accommiata dai suoi utenti nel momento del maggior bisogno.

* Il riferimento alla sera si giustifica con il fatto che gli Abend tendono a manifestarsi nei momenti meno opportuni, quando le persone in grado di risolverli hanno già fatto ritorno alle loro dimore. Va da sè che ci sono Abend e Abend, ma quelli più fastidiosi capitano invitabilmente la sera che precede un bel dì di festa.

accaloP (detta anche zciweisakuL) - Notazione **Polacca** (*vedi*) inversa.

Acronimo - Parola nata per essere memorabile, dietro la quale si nascondono realizzazioni forse non altrettanto memorabili.

* L'ideazione dell'acronimo - qui IBM, Microsoft e Oracle insegnano - è il primo passo per chi ponga mano alla creazione di una nuova architettura, sistema software o linguaggio. Il passo successivo consiste nell'associare ciascuna lettera dell'acronimo alle iniziali di un certo numero di parole scelte più o meno a caso. Tanto più oscuro il significato della frase risultante, quanto maggiore il potere evocativo dell'acronimo. Un esempio per tutti: SQL. Starebbe per Structured subQuery Language ma si pronuncia Sequel, sarebbe azzardato considerarlo un vero linguaggio e poi non è nemmeno strutturato.

Acuratezza - E' la qualità di ciò che è esente da erore (ANSI/IEEE Std 730, 1981).

* La casistica degli "erori" software è ampia, variata e spettacolare. Tre esempi per tutti. (1) Nel giugno del 1962 la prima sonda spaziale americana per Venere, Mariner I, uscì dall'orbita e dovette essere distrutta in volo a causa di un errore in un programma del suo calcolatore di bordo; una istruzione Fortran DO 10 I = 1,5 scritta con un punto al posto della virgola era stata infatti risolta dal compilatore con l'assegnazione del valore 1,5 a una variabile DO10I dichiarata implicitamente. (2) La luna è stata una volta scambiata per un missile in arrivo dallo statunitense Ballistic Missile Early Warning System (BMEWS). (3) In Italia l'Auditel ha rischiato di perdere i dati di ascolto della serata conclusiva del Festival di Sanremo 1992. Si era infatti al 29 febbraio, data non prevista dai programmi che avrebbero dovuto elaborarli.
A proposito, questa frase contiene due erori? Vero o falso?

Andreotti (Legge di, rivisitata) - Il software *non* logora chi *non* lo usa.

Astrazione (Principio di) - Guardare le cose dall'alto per arrivare al nocciolo dei problemi senza disperdersi nei dettagli. Per un analista applicarlo rigorosamente è un pò come inerpicarsi sulla vetta di una erta montagna: ma da lì i problemi dei suoi utenti gli sembreranno piccoli, e lui sembrerà piccolo ai suoi utenti.

* C'è chi lo considera il principio guida dell'Analisi. E chi invece sostiene che un grammo di applicazione vale più di un quintale di astrazione.

B

Bachman (Teorema di, altrimenti detto Legge della **Ineluttabilità**) - Quanto più alto è il costo della realizzazione di un progetto, tanto minore è la probabilità che venga abbandonato, anche se nel corso della realizzazione ci si accorge che non servirà a nulla. Vedi **shelfware**.

* Anche la casistica dei progetti software che si sono trascinati fino all'ineluttabile costoso fallimento è nutrita e spettacolare. Barry Boehm cita il caso del sistema per la prenotazione dei posti sviluppato nel 1970 da Sperry per conto della United Airlines e costato qualcosa come 56 milioni di dollari di allora; al momento della cancellazione del progetto una singola transazione richiedeva l'esecuzione di 146 mila istruzioni, contro le 9 mila inizialmente previste. In Italia resta

ancora da scrivere la storia del progetto Athena affidato nei primi anni '70 dal Ministero delle Finanze a IBM e dalle cui ceneri è nata l'attuale Anagrafe Tributaria, realizzata dal gruppo Finsiel. Per venire a tempi più recenti, va ricordato il fallimento del progetto per il sistema informativo della svedese Volvo raccontato da **Gilb** (*vedi*).

Baskin-Robbins (Effetto) - Altrimenti detto effetto Ce-n'è-per-Tutti-i-Gusti o Se-ne-Vedono-di-Tutti-i-Colori.

* Si dice di un programma sul quale abbiano esercitato i loro talenti intere generazioni di manutentori.

Benûtzefreundlichkeit - Il modo meno *user friendly* che ci sia per dire *user friendly.*

Boyle-Mariotte (Legge di) - Le attività del DP si comportano esattamente come le molecole di un gas inerte: in certe condizioni di temperatura (di progetto) e di pressione (da parte del management) tendono a occupare tutto lo spazio a disposizione. Vedi Legge di **Parkinson**.

* E' l'estensione in campo informatico di una nota legge della fisica: se avete per esempio allocato al testing di un sottosistema un'intera settimana, potete star certi che i test non termineranno un attimo prima dello scadere della settimana (che riescano poi a scoprire i difetti presenti nel software è un'altra storia).

Brooks (Miti di) - Primo Mito: Tutti i programmatori sono ottimisti. Secondo Mito: Il Mese Uomo come unità di misura è un mito illusorio e pericoloso. Terzo Mito: Assegnare nuove risorse a un progetto che è in ritardo significa farlo ritardare ancora di più. Vedi Legge di **Mealy**.

* Da *The Mytical Man-Month* di Frederick P. Brooks, Addison-Wesley (1975).

Bug [in italiano reso qualche volta con *baco*] - (1) Mitico animaletto che si annida tra le pieghe di un sistema hardware/software pregiudicandone il funzionamento. (2) Una funzione non documentata (*vedi* **fixing**). (3) Una manifestazione del destino cinico e baro.

* Per trovare traccia del primo bug documentato della storia occorre risalire agli anni '40 quando una sfortunata falena si immolò tra due contatti di ENIAC, il progenitore degli odierni calcolatori, provocando un corto circuito. Da allora *bug* è tutto ciò che causa un malfunzionamento hardware/software.

Postscriptum: nell'inglese colloquiale falene, cavallette e altri insetti vengono chiamati indistintamente *bug*. A scoprire il primo bug è stata Grace Hopper. Ma in inglese *hopper* vuol dire "cavalletta". Sarà solo un caso?

Bus - Ponderoso veicolo adibito al trasporto, a intervalli irregolari, di persone o di bit.

* Per una persona, o per un bit, perdere il bus può essere un bel guaio.

C

Complessità (del Software) - Una buona misura della complessità di un programma è il numero di dita necessarie per leggerlo. Questo almeno secondo Tom DeMarco.

* Alle misure della complessità il gran libro dell'Ingegneria del Software dedica interi capitoli. Ma anche così il suggerimento di DeMarco (tratto da *Controlling Software Projects*, Yourdon Press, 1982) merita almeno tanta considerazione quanto la Software Science di Halstead o il numero ciclomatico di McCabe.

Confucio (Legge di) - Lunga preparazione, breve opera.

* Citata da Giangiacomo Casonato in *Leadership, Innovazione, Risultati* (Franco Angeli, 1991). Qualche volta l'opera che segue alla lunga preparazione più che breve è brevissima, anzi inesistente: infatti secondo un recente studio condotto negli Stati Uniti dallo Standish Group (1994) il 31% dei progetti software abortisce prima del completamento.

Conway (Legge di) - La struttura di un sistema rispecchia la struttura della organizzazione che lo ha

costruito.

* Mel Conway dice in sostanza che ciascuno ha il sistema che si merita. Se è vero poi che il Sistema Informatico è lo specchio dell'Azienda, può darsi che l'Azienda specchiandosi si veda brutta. Ma, se rompe lo specchio, passerà sette anni di guai.

Costo (del software) - Perchè il software costa "così" tanto? Perchè si suppone che venga sviluppato da persone che costano "così" poco.

D

Data Warehouse - Dallo slang dei neri d'America: *Where is Houzed dis Inbormation?*

* Implicita ammissione del fatto che buona parte delle informazioni in azienda, costate centinaia e centinaia di milioni, è inaccessibile e inutilizzabile. La scelta del termine **Warehouse** sembrerebbe attestare un certo scadimento del valore dell'informazione. Ieri veniva gelosamente custodita nelle banche dati, oggi può bastare un semplice magazzino.

Dato (dal latino *datum* e cioè dono) - Di questa etimologia sono ben consci gli uomini del DP quando agli utenti che si lamentano - come si lamentano - dei dati loro forniti rispondono: "A caval donato non si guarda in bocca".

Default - Il vano tentativo di evitare gli errori attraverso l'inattività.

* Lasciamo al lettore scoprire il sottile filo rosso che lega l'uso dei valori di default a una virtù praticata dagli stoici, l'atarassia.

De minimis non curat... - A non curarsi dei dettagli, almeno secondo le più accreditate metodologie, dovrebbe essere l'Analista, meglio se di Organizzazione. Tanto ci penseranno poi quelli che si occupano della **implementazione** (*vedi*).

* Anche Napoleone agiva in base al principio che *l' intendence suivra*. Ma Napoleone era Napoleone... e poi non dimentichiamoci di Waterloo.

Detroit (Sindrome di) - Ci sono almeno tre modi per fare una cosa: quello giusto, quello sbagliato e quello della Marina. Ma a Detroit erano convinti che per costruire automobili ci fosse solo un modo, il loro. Poi sono venute Honda, Toyota e le altre.

* Secondo Edgar F. Codd a soffrire gravemente della sindrome di Detroit sarebbe IBM che per anni si è crogiolata nella convinzione della propria indiscussa superiorità e della ineluttabilità delle quote di mercato conquistate. Le ultime notizie su Big Blue sembrerebbe dar ragione a Codd.

Digitale - (1) Genere di piante della famiglia delle Scrofulariacee. (2) Un modo come un altro per rappresentare quantità discrete nato dall'uso invalso presso i bambini, le popolazioni primitive e i moderni elaboratori a far di conto con le dita.

* Naturalmente si può far di conto con le dita anche in binario. Ma state attenti alla rappresentazione binaria digitale del numero 17, la vostra gestualità potrebbe essere male interpretata. A proposito, sapete perchè il numero 17 gode di cattiva fama? I Romani lo scrivevano XVII e anagrammando ottenevano un infausto VIXI.

Dijkstra (Tesi di) - Il test di un programma può comprovare la presenza di errori, mai attestare la loro assenza.

Divide et Impera - E' la speranza di risolvere un problema irrisolvibile scomponendolo in un insieme di sottoproblemi egualmente irrisolvibili. Per esempio, come fare entrare cinque elefanti in una Volkswagen? Facendo accomodare i primi due sul sedile anteriore e sistemando gli altri tre in quello posteriore. Il principio del **Divide et Impera** ha come esatto contrario quello dell'**Orda Mongola** (*vedi*).

* Il **Divide et Impera** ha molto a che vedere con l'approccio Top-Down: di fronte ad un compito complesso si individuano dei sottosistemi e se ne fissano le specifiche, che vengono poi passate ai

gruppi di lavoro che li devono progettare e costruire. Secondo John McCarthy (citato da Gabriele Lolli in *La Macchina e la Dimostrazione*, il Mulino 1987) "si dà cioè per scontato che, se i sottosistemi soddisfanno le loro specifiche e sono connessi tra loro nel modo previsto, allora anche il sistema globale soddisferà le sue specifiche. Quanto più complicato è il sistema, tanto meno probabile è che ciò accada".

Dollo (Legge di) - Il passato è indistruttibile. Corollario: Affermare che il passato è indistruttibile non significa che non si possa ripercorrere a ritroso il cammino già fatto, anche se difficilmente ci si ritroverà esattamente al punto di partenza.

* Louis Dollo è un paleontologo belga vissuto a cavallo del secolo scorso. Resta da vedere se i princìpi della paleontologia sono applicabili al mondo del software. Forse sì. Nel nostro caso il "passato indistruttibile" è rappresentato dagli 80 e passa miliardi di linee di codice Cobol esistenti. La manutenzione dei sistemi *legacy* ha molti punti di contatto se non con la paleontologia, almeno con l'archeologia. Quanto poi alla speranza di percorrere a ritroso il cammino già fatto, questa ha un nome: Reverse Engineering.

Down-time - Di tutti i possibili stati di un sistema, l'unico nel quale è rigorosamente esente da errori perchè immune dagli input dei suoi utenti.

E

Education - Insegnare qualcosa nel Terzo Millennio è come invitare qualcuno nel più grande ristorante del mondo e fargli mangiare la lista delle vivande. La rappresentazione delle idee ha preso il posto delle idee stesse.

* A dirlo è Alan C. Kay citando il fisico Murray Gellmann.

Efficienza - E' più facile far sì che un programma che gira sia efficiente, che far girare un programma efficiente.

* Lo sostiene Ed Yourdon in *Managing the Structured Techniques* (Yourdon Press, 1979).

Einstein (Principio della sovrasemplificazione di) - Le cose dovrebbero essere semplici per quanto è possibile, ma non più semplici di quanto è possibile.

Entry (Level) - Nelle strategie IBM l'esatto equivalente del Cavallo di Troia: il più piccolo ed economico di una famiglia di sistemi che, nelle intenzioni di Big Blue, una volta installato presso il cliente dovrebbe aprire la strada ai suoi fratelli più potenti e costosi.

* Poco prima di interrompere la produzione dei Sistemi/36 IBM annunciò l'Entry/36. Non sarebbe stato meglio chiamarlo Exit/36?

Esperto - Una persona che, evitando tutti i piccoli errori, punta dritto alla catastrofe.

* La definizione è di Gerald Weinberg e ci viene da un classico della letteratura del DP, *The Psychology of the Computer Programmer* (Van Nostrand-Reinhold, 1971), oggi sfortunatamente quasi introvabile. A Gerald Weinberg dobbiamo numerosi altri aforismi. Per esempio: nel DP il progresso si svolge a venerdì alterni.

Euristica - L'arte di sembrare terribilmente affaccendati mentre ci si attarda ad analizzare il proprio pollice (o quello altrui) per scoprirne le regole.

F

Fixing - Attività consistente nel convincere l'utente a riformulare le proprie richieste, allo scopo di trarre vantaggio da qualche comportamento inatteso di un programma: *It' s not a Bug, it' s a Feature!*

Fool proof [letteralmente "a prova di scemo"] - Fai un programma che possa essere usato anche da uno scemo e solo uno scemo sarà disposto a usarlo.

* "Il mio lavoro consiste nel disegnare programmi interfaccia che impediscono che accada qualsiasi cosa di allarmante. Per creare un simile sistema il progettista deve immaginare fino a che punto può arrivare la stupidità. In un processo che può durare mesi arrivi a identificare colui che usa il [prodotto del] tuo progetto in un perfetto imbecille". Così Ellen Ullman in "*Out of Time: Reflections on the Programming Life*", *Resisting the Virtual Life* pubblicato nel 1995 da City Lights, un piccolo editore alternativo californiano.

FUD [*Fear, Uncertainty and Doubt*, e cioè Paura, Incertezza e Dubbio] - Acronimo coniato da Gene Amdhal che descrive la strategia adottata da IBM, e non solo IBM, per dissuadere i propri clienti dal percorrere strade che non siano le sue.

* Se FUD è la strategia, le tattiche sono le più diverse: vanno dai pre-annunci alle politiche OCO (Object Code Only) passando per il rifiuto di assistere i prodotti, se appena appena modificati.

G

Gilb (Legge di) - Gli elaboratori possono essere inaffidabili, ma le persone lo sono ancora di più. Corollario: Qualsiasi sistema che per la propria affidabilità dipende dalle persone è inaffidabile.

* A Tom Gilb dobbiamo tra l'altro un Evolutionary Delivery Method che rompe radicalmente con le pratiche del Project Management del passato.

Go To - Istruzione comune a molti linguaggi di programmazione e considerata pericolosa perchè permette al programmatore di trasferire il controllo a qualche remota porzione del programma ancora tutta da inventare prima di andare a prendersi un caffè.

* Ma il Go To è realmente pericoloso? Così sostiene Edsger W. Dijkstra una cui famosa lettera ("*Go To statement considered harmful*") pubblicata nel marzo 1968 su *Communications of the ACM* ha scatenato la cosiddetta rivoluzione strutturata, e della stessa opinione si sono detti Niklaus Wirth, Harlan Mills e molti altri. Di parere diametralmente opposto il dott. Eiichi Goto che, per tutta la durata del Congresso IFIP del 1971, si è aggirato tra i partecipanti cercando di convincerli della propria intrinseca innocuità.

Grosch (Legge di G. rivisitata) - Il costo dello sviluppo di un sistema è proporzionale al cubo del costo del sistema su cui avviene lo sviluppo.

* Nella sua formulazione originaria da parte di Herbert R. J. Grosch agli inizi degli anni '50 suonava invece così: "La potenza di calcolo è proporzionale al quadrato del costo di un sistema". Ma è stata clamorosamente smentita dall'avvento prima dei mini e poi dei personal.

Gruppo di lavoro (Legge del, altrimenti detta Quarta Legge di **Parkinson**) - In ogni gruppo di lavoro il numero delle persone aumenta indipendentemente dalla quantità del lavoro svolto.

H

Hawthorne (Effetto) - Se ti senti gli occhi di tutti puntati addosso, finirai inevitabilmente col lavorare, se non meglio, di più.

* L'effetto Hawthorne spiega perchè l'occhio del padrone ingrassa il cavallo e come mai gli splendidi risultati di un progetto pilota siano poi irripetibili. E' stato registrato per la prima volta nel 1932, quando alla Western Electric decisero di studiare il rapporto tra illuminazione ambientale e produttività del lavoro. Gli analisti, osservando molto da vicino il comportamento di tre diversi gruppi campione di lavoratori, constatarono che: (1) aumentando l'illuminazione, la produttività cresceva; (2) diminuendo l'illuminazione, la produttività cresceva; (3) mantenendo l'illuminazione ai livelli consueti, la produttività cresceva ...

I

Implementazione - Un momento della eterna lotta tra chi sa, ma è sottopagato, e chi invece non sa, ma viene retribuito con generosità.

Inglese - Il più innaturale dei linguaggi naturali.

Intelligenza Artificiale - Termine coniato nel 1956 da John McCarthy per designare un particolarissimo tipo di protesi.

* Il direttore di una certa rivista chiese una volta a Edsger W. Dijkstra di scrivere un articolo sul tema "Le macchine possono pensare?". D'accordo, rispose Dijkstra, ma a una condizione che sullo stesso numero la rivista pubblichi anche un articolo su "I sommergibili possono nuotare?".

Interfaccia - (1) Linea di demarcazione arbitraria tra due sistemi tracciata dagli addetti allo sviluppo allo scopo di sfuggire alle proprie responsabilità. (2) Il meccanismo che permette ai malfunzionamenti di propagarsi da un sistema all'altro.

Irregolari (Verbi) - In tutti i linguaggi troviamo dei verbi irregolari, ma i verbi dell'Informatica sono un po' più irregolari degli altri.

Tabella dei verbi irregolari

Io assemblo
Tu compili
Egli interpreta.

Io sono transazionale
Tu sei time-sharing
Egli aspetta.

Io pianifico
Tu controlli
Egli sgobba.

Io sviluppo
Tu installi
Egli prega.

J

Jackson (Leggi di) - Prima Legge: E' bene sorvolare sulle fasi di Analisi e Disegno e precipitarsi a scrivere i programmi, allo scopo di guadagnare il tempo necessario per rimediare agli errori presenti nei programmi per aver sorvolato sulle fasi di Analisi e Disegno. Seconda Legge: per un programmatore la saggezza incomincia quando si rende conto della differenza che passa tra tra un programma qualsiasi che gira nel modo giusto e far girare il giusto programma.

L

La Favolosa Macchina di Chicago - Da una parte della Macchina Favolosa entravano maiali vivi e dall'altra uscivano salsicce. E' l'esatto contrario del principio su cui si basa il Re-engineering. Qui infatti da una parte entrano programmi sorgenti e dall'altra dovrebbero uscire le loro specifiche.

* Il primo a parlare della Favolosa Macchina di Chicago è stato, agli inizi del secolo, il matematico francese Henry Poincaré. Il problema allora era se fosse realmente possibile dimostrare tutte le verità matematiche usando il metodo assiomatico. Poi venne Gödel e il suo teorema sulla indecidibilità (1931).

Leader (di Progetto) - Colui che apre la marcia lungo il cammino di progetto. Quando incontra un inciampo, è tentato di rivolgere a qualche suo seguace la fatidica frase: "Vai avanti tu, che a me viene da ridere".

Ledgard (Legge di) - Tanto prima incominci a scrivere il tuo programma, tanto più tempo ci vorrà per finirlo.

M

Manutenzione (del software) - Non è una faccenda che ci dovrebbe riguardare. Infatti il software, essendo fatto di bit e non di atomi, non è soggetto a rotture o a usura. Tuttavia... Vedi Legge di **Andreotti** rivisitata.

* In un suo documento ufficiale la US Air Force ha stimato che, se le cose non non fossero cambiate, nell'anno 2000 il 25% della popolazione statunitense in età di leva (dai 18 ai 25 anni) sarebbe stata impegnata nella manutenzione del software. E c'è chi si è spinto più in là. Infatti Capers Jones ha calcolato che, se il numero degli abitanti del nostro pianeta e la richiesta applicativa aumenteranno ai ritmi attuali, nel giugno del 2052 (non c'è dato sapere il giorno e l'ora) saremo tutti programmatori e manutentori.

Mealy (Legge di) - Se un progetto ritarda, aggiungere una nuova persona al gruppo di lavoro consuma più risorse di quante ne produca.

Mese persona - Misura dello sforzo di progetto che sostituisce l'arcaico **Mese uomo** (*vedi*).

* Quiz: quale fattore di conversione usereste dovendo passare da Mesi uomo a Mesi persona?

Mese uomo - Misura arcaica dello sforzo di progetto, resa obsoleta dalla sempre più massiccia presenza femminile negli organici del DP.

* Oltre oceano, dove ci si è resi presto conto da che parte tirava il vento, il lessico informatico ha da tempo ripudiato termini ostentatamente maschilisti come *father* e *son*, sostituendoli con gli asessuati e politicamente corretti *parent* e *child*. Da noi invece, a proposito di segmenti e file, si parla ancora di *padre*, di *figlio* e perfino di *nonno*. Informatiche di tutta Italia unitevi!

Metodologia - Ovvero come raggiungere il successo nello sviluppo applicativo nonostante l'incompetenza.

* Questo secondo Tom DeMarco lo scopo inconfessato delle metodologie, almeno di "certe" metodologie. Per il resto sarebbero troppo ingombranti e pedantemente prescrittive per risultare di una qualche pratica utilità.

metriche (del Software) - Non si può controllare ciò che non si sa misurare.

* Così Tom DeMarco in *Controlling Software Projects*. Ma Albert Einstein ribatte: "Non tutto ciò che conta può essere contato e non tutto ciò che può essere contato conta".

Mille Programmatori (Legge dei) - Se assegnate mille programmatori a un progetto senza aver ben definito il disegno del sistema, otterrete un sistema di almeno mille moduli - anche se non ne dovrebbe contare più di cento.

Murphy (Legge di) - Se qualcosa può andare storto, non mancherà di farlo.

* Di tutte le "leggi" (si fa per dire) cui soggiaciono le vicende del DP, e non solo del DP, è probabilmente quella che ha valore più universale: questo concentrato di pessimismo è stato enunciato nel 1949 da Ed Murphy, ingegnere aeronautico e capitano della US Air Force,

osservando il risultato dei test che stava eseguendo. Dalla Legge di **Murphy** nasce tutta una serie di corollari: vanno da "Se qualcosa non può andare storto, lo farà lo stesso" a "Niente va mai così male che non possa andare anche peggio".

Myers (Legge di) - La probabilità dell'esistenza di altri errori in una sezione di programma è direttamente proporzionale al numero di errori già reperiti in quella sezione di programma.

* Enunciata da Glenfor Myers nel suo *The Art of Software Testing* (Wiley & Sons, 1979) può essere dedotta da comuni esperienze domestiche: la scoperta anche di un solo ospite indesiderato nella dispensa è un forte invito a procedere a una disinfestazione generale (del resto la caccia e l'eliminazione dei difetti nel software non vanno sotto il nome di debugging?).

N

Next Release - Di tutte le possibili versioni di un prodotto software la sola rigorosamente esente da difetti ed errori, l'unica a risolvere tutti i problemi, attuali e futuri, dell'utente.

* Ma, attenzione: ciò che distingue un Last Release da un Next Release non è solo il fattore tempo. Il Last Release è puntuale, ma inefficace. Il Next Release è invece risolutivo, ma immateriale. Infatti, come osserva George Schussel, non lo si può nè vedere, nè toccare.

	Si può vedere?	Si può toccare?
Reale	Si	Si
Virtuale	Si	No
Trasparente	No	Si
Next Release	*No*	*No*

Novanta per Cento (Sindrome del) - Il primo novanta per cento di un lavoro viene svolto nel novanta per cento del tempo, e il restante dieci per cento nel restante novanta per cento.

* Ne sono vittime gli addetti ai lavori quando il tempo passa e un progetto software, che era lì lì per per giungere in porto, continua a restare incompiuto. Barry Boehm nel suo *Software Engineering Economics* (Prentice-Hall, 1981) cita il caso di quel programmatore che, dopo sei settimane di lavoro, annunciò di essere arrivato al 90% (tutti i programmi erano stati scritti, compilati e provati; restava solo da individuare la causa di un paio di piccoli errori e da apportare altrettante modifiche di poco conto ai programmi). Ma la settimana dopo non era andato al di là del 95% e per completare il tutto ci vollero altre cinque settimane.

O

Orda Mongola - E' la speranza di risolvere un problema complesso con la sola forza del numero. Si basa sull'idea che, se un muratore costruisce una casa in cento giorni, cento muratori possono farlo in un giorno; se un aereo attraversa l'Atlantico in quattro ore, due aerei possono farlo in due ore; se a una donna per mettere al mondo un figlio occorrono 9 mesi, a nove donne... E' l'esatto contrario del principio del **Divide et Impera**.

* L'applicazione del principio dell'Orda Mongola passa attraverso le leggi di **Mealy** e dei **Mille Programmatori** (*vedi*).

Ossimoro (Dal greco **ontr**, che vuol dire acuto, e **lxqor** che vuol dire ottuso) - E' la figura retorica che risulta dall'alleanza dei contrari; per esempio, un*silenzio eloquente*. Più rigorosamente un ossimoro può essere definito come la concatenazione delle stringhes_1 e s_2 che, in una logica a due valori, non abbiano lo stesso valore di verità.

* In campo letterario gli ossimori sono una chicca, in campo informatico un fatto della vita.

Tavola degli ossimori

- Assicurazione di Qualità.
- Backup e Restore.
- Data di Consegna.
- Facile da Usare.
- Linguaggio Naturale.
- Programma Strutturato.
- Tempo di Risposta.
- Versione Finale.

O'Toole (Chiosa di) - Murphy era un ottimista (*vedi* **Murphy**, Legge di).

Ottimizzare (un programma) - (1) Non farlo; (2) Aspetta a farlo.
* Il consiglio ci viene da Michael Jackson nei suoi *Principles of Program Design* (Academic Press, 1975).

P

Paralisi da Analisi - Colpisce chi ha più fiducia nella propria capacità di sviscerare un problema che in quella di risolverlo.
* *Analysis Paralysis* è uno degli AntiPattern descritti da Brown & C.: "Nella fase di Analisi ci si è posti l'obiettivo di raggiungere la perfezione e la completezza. Questo AntiPattern è caratterizzato da continue revisioni dei modelli e dalla generazione di modelli con dettagli la cui utilità per i processi a valle è meno che nulla".

Pareto (Legge di) - Il 20% delle istanze fa l'80% delle importanze.
* E' la estensione in campo informatico del celebre principio enunciato da Vilfredo Pareto. Tre esempi per tutti: (1) Il 20% degli scenari (= funzioni) primari di un sistema soddisfa di norma l'80% delle esigenze dei suoi utenti. (2) Lo sforzo necessario per scoprire, descrivere e implementare l'80% delle funzioni di un sistema può essere minore di quello per portare a termine il restante 20%. (3) In un sistema software l'80% (o più) dei difetti si addensa nel 20% (o meno) dei moduli. Capers Jones segnala che dei 425 moduli di cui consisteva IMS/VS più di 300 erano completamente esenti da errori, mentre il 7,3% dei moduli raccoglieva il 57% dei malfunzionamenti segnalati (APAR).

Parkinson (Prima Legge di) - Il lavoro si espande fino a occupare tutto il tempo disponibile: più è il tempo e più il lavoro sembra importante e impegnativo.
* Per una diversa formulazione *vedi* anche Legge di **Boyle-Mariotte**.

Paul (Principio di) - In una organizzazione ogni membro tende a raggiungere una posizione in cui i suoi skill tecnici diventano obsoleti nell'arco di cinque anni.
* Va da sè che gli effetti congiunti dei Princìpi di **Paul** e di **Peter inverso** (*vedi*) sono dirompenti: l'unico per prevenirli è il *promoveatur ut amoveatur.*

Peter (Principio di) - In una organizzazione ogni membro tende a raggiungere il proprio livello di incompetenza. Corollari: (1) Col tempo ogni posizione tende a essere occupata da un membro che è incompetente a svolgere quel lavoro (*vedi* anche Principio di **Paul**). (2) Il lavoro viene svolto dai membri che non hanno ancora raggiunto il loro livello di incompetenza.

Principio di **Peter inverso** - In una organizzazione ogni membro che fa carriera sale fino a ricoprire una posizione nella quale diventa insostituibile. E a quella posizione resta abbarbicato per sempre.

* Ecco come ce lo racconta Barry Boehm: "Può accadere che qualche programmatore divenga così versato nelle intrinseche complessità e nei rituali elaborativi di certi programmi che l'azienda, credendo di fare il proprio bene, rifiuti di farlo lavorare su qualcosa di diverso. Ma a questo punto il programmatore di solito se ne va e per l'azienda cominciano sul serio i guai".

Piranha (Legge del) - Il software è come l'entropia. E' difficile da afferrare, pesa quasi nulla e obbedisce alla Seconda Legge della Termodinamica: il suo appetito cresce incessantemente.

* Da *Augustine' s Laws* di N. R. Augustine, Viking Press (1985).

Polacca (Notazione) - Espediente linguistico cui ricorrono tutti coloro che non sono capaci di pronunciare il nome del professor Lukasiewicz, ma desiderano rendere comunque omaggio alla sua terra natale.

Portatile - Lo si dice di un personal computer che può essere trasportato da una sola persona: "Datemi una maniglia e trasporterò il mondo".

* Il *trasportable* IBM dei primi anni '80 pesava qualcosa come 14 chili. Ma allora è trasportabile anche il Duomo di Milano, basta trovare una maniglia adatta.

Potemkin (Effetto) - Altrimenti detto Effetto **Facciata**. E' il principale rimprovero che Bertrand Meyer muove ai fautori del Prototyping. Dopo aver visto un modello all'apparenza perfettamente funzionante del loro sistema, gli utenti non sanno capacitarsi del perché occorrano mesi e mesi per entrarne in possesso.

* Un po' di storia: dopo essersi sbarazzata del marito Pietro ed essere così salita sul trono della Santa Russia, Caterina II la Grande non si dimostrò del tutto immemore delle sorti del suo popolo (allora come oggi non è che in Russia le cose andassero tanto bene). Per compiacerla il principe Potemkin, che di Caterina era primo ministro e anche qualcosa di più, ricorse a un ingegnoso espediente: un tour guidato a finti villaggi di isbe messe su nella migliore tradizione hollywoodiana (di vero c'erano solo le facciate), popolati da mugiki e mugike ben pasciuti e festanti (ma erano solo figuranti).

Prezzo/Prestazioni (Rapporto) - Un modo come un altro per portare acqua al proprio mulino mettendo a confronto capre e cavoli. Così si può dimostrare praticamente di tutto. Occorre solo un pò d'attenzione: il denominatore dev'essere diverso da zero.

* Il rapporto prezzo/prestazioni ha qualcosa a che vedere con una frase che si sente spesso echeggiare nei *suk* arabi: "Caccia moneta, vedi cammello". A nessuno sfuggirà la profonda differenza che passa tra la materialità di quel "caccia" e la virtualità del "vedi cammello".

Programmatore residuale (Teorema del) - Più propriamente la speranza che, dopo l'*n*-esima ristrutturazione del gruppo di lavoro, per $n \to \infty$ l'insieme dei programmatori sopravvissuti non sia vuoto. Un corollario asserisce che, per $n + 1$, l'insieme sarà un pò meno non vuoto. Se dopo m ristrutturazioni l'insieme risulta definitivamente vuoto, l'm-esima ristrutturazione è quella che pone fine a tutte le speranze.

PTF [*Program Temporary Fix* o più realisticamente *Permanent Temporary Fix*] - Non c'è nulla di così permanente come un fix temporaneo.

* PTF è l'eufemismo usato in casa IBM per indicare il tentativo di ovviare a un errore software rilevato sul campo. Se il tentativo ha successo la PTF, nata temporanea, diventa permanente: verrà incorporata nelle successive versioni del prodotto assieme all'errore originario e potrà introdurne di nuovi.

Q

Qualità (del software) - La Qualità ha molti punti di contatto con il sesso. A parole tutti si dicono favorevoli (a certe condizioni, ben inteso). Tutti sono convinti di capirlo (anche se non sono disposti a spiegarlo). Tutti pensano che per esercitarlo basta seguire le proprie naturali inclinazioni (dopo tutto ne nasce sempre qualcosa). Infine sono in molti a credere che, se sorge un problema, la colpa è degli altri (si fossero solo presi il tempo di fare le cose per bene).

* Phil Crosby (*Quality is Free*, McGraw-Hill, 1979) definisce così una Qualità che non è necessariamente solo quella del software. Ma aggiunge: "*Quality is free, but it is not a gift*".

R

Re-Engineering - Reingegnerizzare un'applicazione in esercizio è un pò come pretendere di sostituire un pneumatico su un'automobile in corsa.

* A dirlo è Eric Bush, uno dei padri del ReEngineering.

Ricchezza (Teoria Spagnola della) - Si basa sull'assunto che la quantità di ricchezza al mondo è finita. Quindi, dato che la ricchezza non può essere creata, l'unico modo per arricchirsi - come sapevano bene i *conquistadores* - è spremere gli altri come limoni.

* Tom DeMarco e Tim Lister sostengono nel loro *Peopleware* (Dorset House Publishing, 1987) che in certi ambienti del DP la Teoria Spagnola della Ricchezza è tutt'ora in auge. E ammoniscono: "Le persone che vengono spremute non è che lavorino meglio, lavorano solo più in fretta".

Ricorsivo - Vedi **Ricorsivo**.

S

Sarson (Legge di) - Scomporre un DFD è un pò come compitare la parola 'Massachusetts'. Non si sa mai quando si è arrivati in fondo.

* Con lei si è aperta l'era dei DFD. Trish Sarson è infatti coautrice con Chris Gane di *Structured Systems Analysis* (Improved Systems Technologies, 1977).

Scott (Legge di) - Una volta individuato e corretto un errore, ci si accorge presto che le cose andavano meglio prima. Corollario: Quando ci si accorge che le cose andavano meglio prima, è troppo tardi per tornare indietro.

Shelfware - E' il software nel cassetto. Si dice di quei prodotti hardware, software o metodologici che, dopo essere stati acquistati magari a caro prezzo, vengono accantonati da qualche parte senza venire mai usati.

* Secondo i risultati di uno studio condotto nel 1982 dalla Butler-Bloor il 47% dei sistemi software commissionati dal governo statunitense non è mai stato usato.

Sistemista - (1) Colui che sa ciò che gli analisti ignorano e non vogliono sapere, e che i programmatori invece sospettano anche se nessuno glielo dice. (2) Colui che sistema i sistemi: "Adesso ti sistemo io".

Spaghetti (Codice a) - Componente essenziale della cosiddetta dieta mediterranea e di qualsiasi parco programmi che si rispetti (*vedi* anche Effetto **Baskin-Robbins**).

* Nel termine si perpetua quel tanto di prevenzione che i *Wasp* (bianchi, anglosassoni e protestanti) nutrivano nei confronti delle abitudini alimentari e non dei nostri connazionali approdati negli Stati Uniti. Ma non è il caso di offendersi. Dopo tutto noi non li ricambiamo della stessa moneta parlando di "truffa all'americana"?

Specialista - E' colui che aspira a saperne sempre di più su un argomento sempre più circoscritto. Così finisce col sapere tutto su nulla.

* Lo specialista ama guardare le cose molto da vicino. Ma, se ispezionate un elefante troppo da vicino, potrete soltanto dire che è grigio.

Stabilizzare - Nel meraviglioso linguaggio IBM "stabilizzare" suona come una campana a morto. Quando vi hanno detto "L'8100 è stato stabilizzato" forse avete pensato: "Adesso finalmente funziona". Nulla di tutto ciò. Significava semplicemente. "Tra poco smetteremo di produrlo, abbiamo già smesso di occuparci del suo software e non siamo nemmeno tanto intenzionati ad assisterlo".

Stop dinamico - Così lo chiama l'autore, se il programma è il suo. Fosse stato scritto da un altro, sarebbe un loop senza fine.

Stupidità (Prima Legge Fondamentale della) - Sempre e inevitabilmente ognuno di noi sottovaluta il numero di individui stupidi in circolazione.

* Questo almeno secondo Carlo Cipolla. E Murphy aggiunge: "Gli stupidi sono sempre più ingegnosi delle precauzioni che si prendono per impedire loro di nuocere".

T

Tallone d'Achille (Principio del) - Se si è trascurato di esercitare uno stretto controllo anche su un solo fattore critico di progetto, il progetto fallirà proprio perchè si è trascurato quel fattore critico (*vedi* Legge di **Toffler**).

Tempo E' la sola cosa che salva gli uomini del DP dal rischio che tutti i guai capitino loro addosso contemporaneamente.

* Ma anche così il tempo non ci è necessariamente amico. Un corollario della Legge di **Murphy** (*vedi*) infatti ammonisce: Se c'è una possibilità che varie cose vadano male, quella che causa il guaio peggiore sarà la prima a farlo.

Titanic (Effetto) - Pensare che un disastro è impossibile rende possibili disastri impensabili.

Toffler (Legge di) - In un progetto ci si può anche dimenticare di qualche fattore critico, ma difficilmente questo fattore si dimenticherà di noi (Alvin Toffler in *The Third Wave*, Collins 1980).

U

Useless Case - Una funzione superflua, un Caso d'Uso senza valore.

* Qual'è la differenza tra le tradizionali specifiche funzionali e i Casi d'Uso? Le prime rispondono alla domanda "Cosa deve fare il Sistema?". Con i Casi d'Uso la domanda viene riformulata così: "Cosa deve fare il Sistema per ciascun suo utente?". Solo tre parole in più, ma con conseguenze - osserva Ivar Jacobson - non di poco conto. Gli *Use Case* infatti "ci guidano a trovare le funzioni di cui ciascun utente ha bisogno. Ma ci aiutano anche a non proporre funzioni superflue di cui nessuno sente la necessità". Gli *Useless Case* appunto.

Utente - Di lui si dice spesso che non sa quello che vuole. Sarà anche vero, ma una cosa è certa: l'utente sa benissimo quello che non vuole.

V

Valutazione (di Progetto) - La breve pausa che intercorre tra la richiesta di un Alto Papavero "Voglio che tutto sia finito per domani" e il vostro "Sissignore".

* Allo Standish Group stimano che il 52,7% dei progetti software termina in ritardo ed eccede gravemente, dell'80% e più, i limiti di spesa budgettati. Di chi è la colpa? Naturalmente dei capiprogetto e degli sviluppatori. Ma c'è chi sospetta - come Tom DeMarco - che i progetti costano di più e terminano in ritardo semplicemente perchè le stime dei loro tempi e costi sono sempre troppo strette.

Vaporware - Sta per fumisteria. E' ciò cui ricorrono, in mancanza di meglio, i produttori di hardware/software.

* Un buon esempio di **vaporware** è il **Next Release** di un qualsiasi prodotto software (*vedi*).

Voltaire (Legge di, da *Candide*) - Tutto va per il meglio nel migliore dei mondi possibili. E' quanto sostengono gli ottimisti. E i pessimisti? Temono che gli ottimisti abbiano ragione.

* Ma gli uomini del DP - si sa - sono degli inguaribili ottimisti (*vedi* Primo Mito di **Brooks**). Ecco infatti come reagiscono alle sciagure che si abbattono su di loro. Il programmatore: "Non si trova più il sorgente di quel programma? Meglio così! Non dovrò più metterci le mani." Il sistemista: "Il master file è stato cancellato per errore? Splendido! Così potremo finalmente scoprire se il nostro sistema di back-up/recovery funziona veramente". In sala macchine: "L'inverno scorso abbiamo rischiato di morire di freddo quando è venuta a mancare la corrente. Fortuna che il gruppo di continuità è andato a fuoco".

W X Y Z

Weinberg (Legge di) - Se i costruttori costruissero come i programmatori programmano, il primo picchio che passa potrebbe distruggere la civiltà.

* Senza commenti...

Chi l'ha detto?

Le vittorie hanno molti padri, le sconfitte restano invece orfane. Anche il nostro Dizionarietto ha registrato qualche sconfitta. Quando ci siamo chiesti "Chi l'ha detto?", molte volte ci siamo dovuti arrendere al fatto che molte voci, e non necessariamente le meno saporite, restavano ostinatamente orfane.

Quanto alle altre, abbiamo il forte sospetto che alcune cattiverie d'autore - come tali debitamente registrate nel Dizionarietto - circolassero già tra gli addetti ai lavori già un bel po' di tempo prima che qualcuno pensasse bene di farle proprie.

E non è tutto. Veniamo ai padri, putativi e non, delle Leggi, Princìpi, Teoremi e Corollari raccolti nel Dizionarietto. Quando si è trattato di tracciare un loro breve profilo, ci siamo imbattuti in una nuova manifestazione della Legge di Murphy: "Nel momento in cui vogliamo saperne di più sul conto di un certo personaggio, di colpo tutte le fonti di informazione su quel personaggio si inaridiscono".

* * *

Amdhal, Gene - Dopo aver progettato il Sistema/370 IBM, ha costituito la società produttrice di mainframe compatibili che porta il suo nome. Poi si è imbarcato in una serie di altre avventure, per la verità non tutte fortunate.

Andreotti, Giulio - Lui il potere non lo ha certamente logorato. Eletto deputato nelle prime dieci legislature della Repubblica e poi nominato senatore a vita, è stato infatti sei volte sottosegretario, venti volte ministro e sette volte presidente del consiglio.

Bachman, Charles - Nel 1973, quando era a capo del gruppo di studio ANSI/ SPARC cui dobbiamo l'architettura a tre livelli del database, ha ricevuto dalla Association for Computing Machines (ACM) il prestigioso premio Alan Turing per le sue attività pionieristiche nel campo del database. Negli anni '60 Bachman ha infatti realizzato per General Electric il primo vero DBMS della storia, IDS. Nel 1983, fondata la società che portava il suo nome, si è avventurato ancora una volta in un campo inesplorato: il Re-engineering.

Boehm, Barry W. - Dopo trascorsi presso la Rand Corp. e la TRW è stato a capo della DARPA/SISTO, il più importante centro di ricerca sul software del governo statunitense, e oggi insegna alla University of South California. Ma è meglio noto per essere il padre del modello COCOMO e l'autore di una monumentale *Software Engineering Economics*: 767 pagine piene zeppe di formule, grafici e tabelle. Chi ha avuto il coraggio di scorrerle fino in fondo, ha tuttavia scoperto che tra tanta scienza serpeggia una sottile vena di humor nero.

Boyle-Mariotte - In realtà due diversi personaggi: Robert Boyle (1627 - 1691), irlandese, e Edme Mariotte (1620 - 1684) francese. La Legge della fisica che prende il loro nome viene chiamata di Boyle-Mariotte in tutto il mondo, ma in Francia dove sono un pò sciovinisti è semplicemente la Legge di Mariotte.

Brooks, Frederick P. - Dopo aver diretto il gruppo di progetto che ha realizzato l'OS/360 IBM ha saltato il fosso per andare a insegnare all'Università di North Carolina. Il suo *The Mythical Man-Month* è stato scritto nel 1975, ma i maggiori successi di vendita sono venuti dieci anni dopo, nel 1985. Secondo Brooks anche questo è un segno dei tempi.

Bush, Eric - E' considerato, assieme a Charlie Bachman, uno dei padri del Re-engineering. Ha ceduto la casa di software di cui era fondatore per dirigere il Rosetta Stone Consortium, un centro di ricerca sul CASE molto vicino all'Università di Harvard.

Casonato, Giangiacomo - Una luminosa carriera in IBM Italia. Quando l'ho incontrato era partner nella

Coopers & Lybrand Consulenti di Direzione.

Cipolla, Carlo M. - Ha insegnato alla Normale di Pisa e all'Università di Berkeley ed è autore, tra l'altro, di una monumentale *Storia economica dell' Europa preindustriale*. Ma la vera fama gli è venuta con la pubblicazione nel 1988 di un libricino scherzoso di sole 81 pagine, *Andante ma non troppo*, edito da Il Mulino. E' stato il best seller dell'anno.

Codd, Edgar F. - E' il padre incontestato del Relazionale. Si è imbattuto nel suo celebre Modello nel 1969 dopo qualcosa come vent'anni di onorata milizia nei laboratori di ricerca IBM. Il "Doctor" Codd (negli Stati Uniti le cose non vanno come da noi e i titoli accademici vengono usati con maggiore parsimonia) è entrato infatti in IBM nel 1949 con la strana qualifica di "matematico programmatore" per il SSEC-Selective Sequence Electronic Calculator, con le sue dodicimila valvole il progenitore di tutti gli elaboratori di *Big Blue*.

Confucio (551 - 479 a.C.) - Confucio, o meglio il confucianesimo, è ancor oggi alla base della cultura cinese. Ma noi al suo convenzionale perbenismo preferiamo la saggezza di Lao Tze, specialmente quando, sempre nel V secolo a.C., osservava che l'utilità della creta nel modellare un vaso deriva dalla cavità prodotta dalla sua assenza.

DeMarco, Tom - Nel 1986 ha ricevuto il prestigioso premio Warnier per il suo contributo alla storia più recente dell'Informatica. E' infatti uno dei padri dell'Analisi Strutturata, oltre che autore di libri di successo come *Controlling Software Projects* (Yourdon Press, 1982) e *Peopleware* scritto a quattro mani con Tom Lister (Dorset House Publishing, 1987).

Dijkstra, Edsger W. - Per molti l'era della Programmazione Strutturata si è aperta con la pubblicazione nel 1968 di una sua famosa lettera all'editor di *Communications of the ACM*. Ma nella lettera di Dijkstra del termine "Programmazione Strutturata" non si trova traccia. Chi è stato allora il primo a usarlo? Sembrerebbe un certo J. D. Aron durante una conferenza organizzata a Roma dalla NATO. Era il 1969.

Dollo, Louis - Di professione paleontologo, ha dedicato tutta la sua vita al compito ingrato di preparare e descrivere le ossa di trentuno iguanodonti scoperti nel 1878 in una miniera di carbone belga.

Einstein, Albert (1879 - 1952) - Come Paolo Villaggio e Enrico Beruschi ha cominciato facendo il *travet*. Poi è diventato lo scienzato più famoso del XX secolo. Ma, a differenza di Beruschi e Villaggio, non ci risulta che abbia fatto i soldi.

Gilb, Tom - Deve la sua fama a una spietata analisi post mortem del progetto della svedese Volvo per un nuovo Sistema Informativo. Da questa ha tratto lo spunto per proporre nei suoi *Principles of Software Engineering Management* (Addison-Wesley 1988) un Evolutionary Delivery Method che rompe con le cattive pratiche del passato.

Grosh, Herbert R. J. - C'è chi lo chiama "doppietta". Ha infatti lavorato due volte per IBM, due volte per General Electric e due volte per il Governo Federale statunitense, l'ultima delle quali come direttore del Bureau of Standards Institute. Attualmente vivrebbe in Europa facendo a tempo perso il consulente.

Hawthorne - Non è il nome di una persona, ma quello dell'impianto di Chicago della Western Electric nel quale gli analisti della Elton Mayo rilevarono per la prima volta l'omonimo effetto.

Hopper, Grace M. - Uno dei primi casi di donna in carriera. L'ha iniziata nel 1944 come semplice programmatrice di ENIAC per concluderla con il grado di contrammiraglio della U.S. Navy. Alla sua partecipazione ai lavori del Codasyl dobbiamo la nascita del Cobol.

Jackson, Michael - Noto al grande pubblico per grandi successi discografici come *Thriller* ed editoriali come *Principles of Program Design*. Ma forse non si tratta esattamente della stessa persona.

Jacobson, Ivar - Dopo una vita passata in Ericsson, ha varcato l'oceano stabilendosi in California dove, con Grady Booch e James Rumbaugh, ha dato vita a UML. E' l'inventore dei Casi d'Uso. E con i Casi d'Uso scomposizione funzionale e approccio *top-down* addio!

Jones, Capers - Attualmente a capo della Software Productivity Research, è una delle maggiori autorità al mondo in fatto di metriche del software. In precedenza si è occupato di produttività dello sviluppo presso IBM, ITT e Nolan, Norton & Co.

Lukasiewicz, Jan (1878 - 1956) - Epistemologo e logico-matematico polacco, ha esteso le tradizionali Tavole di Verità. Della sua logica a tre valori si sono poi impadroniti quelli del Relazionale

Kay, Alan C. - E' stato tra i padri fondatori del Palo Alto Research Center (PARC) della Xerox dove sono nate tante innovazioni come le icone, il mouse e le finestre, oltre a Smalltalk. E' stato poi responsabile delle ricerche all'Atari e nel 1984 è passato alla Apple. Oggi è al MIT dove si occupa di ragazzi, insegnamento, reti e calcolatori.

Martin, James - Consulente, conferenziere e autore di libri di fama mondiale. Ma anche un uomo avveduto e fortunato. Per strapparlo dal suo castello in Irlanda o dall'isoletta dei Caraibi dove risiede a seconda delle stagioni sareste disposti a sborsare un *cachet* pari a quello di una *top-model*?

McCarthy, John - Dovessimo assegnargli un voto ci troveremmo in imbarazzo. Gli dobbiamo infatti una Teoria Generale della Computazione (essendo solo una teoria non può produrre molti guasti: voto 8). Ma sempre a lui e a Marvin Minsky dobbiamo anche un famoso progetto sull'Intelligenza Artificiale iniziato al MIT nel lontano 1975 (qui i risvolti pratici potrebbero essere temibili: voto 6 - -).

Meyer, Bertrand - Francese ma trapiantato negli U.S.A. è presidente di Interactive Software Engineering, professore alla Monash University, e autore di libri di successo tradotti anche qui in Italia. Ma il suo cruccio è *Eiffel*, il linguaggio OO con cui avrebbe voluto conquistare il mondo. Oggi Meyer osserva sconsolato le sterminate piantagioni di C++ e le sempre più estese coltivazioni di Java, mentre *Eiffel* sopravvive a stento nella fioriera di qualche appassionato. Ma non avere alle spalle Bell Atlantic e Sun-HP conterà pure qualcosa...

Murphy, Ed - E' un personaggio realmente esistito. Ingegnere aeronautico e capitano della US Air Force, non ha tratto - ma poteva essere diversamente? - particolari vantaggi dall'enunciazione della sua famosa Legge. Chi ha fatto invece fortuna è Arthur Bloch, che nel 1977 ha raccolto in un volumetto le Leggi di cui quella di Murphy è il capostipite. Grazie ai diritti d'autore riscossi avrebbe acquistato un'isoletta nei Caraibi dove farebbe la vita del pascià.

Myers, Glenford - E', insieme con L.L. Constantine e W. Stevens, uno dei padri del Disegno Strutturato. Constantine ha contribuito all'impresa con la sua l'esperienza e Myers con molte nuove idee. E Stevens? Di lui i maligni dicono che si sia limitato a trasformare da "Composite Design" a "Structured Design" (in IBM "strutturato" è da sempre una parola di sicura presa) il titolo dell'articolo, pubblicato nel 1974 su *IBM System Journal*, col quale si apriva l'era delle Tecniche Strutturate.

Pareto, Vilfredo (1848 - 1923) - Al marchese Pareto, grande economista, si deve l'osservazione che in Toscana alla fine del XIX secolo il 20% delle famiglie deteneva l'80% della ricchezza. Da qui la sua celebre Legge.

Parkinson, Northcote C. - Storico navale inglese, è giunto alla formulazione della sua Legge constatando che ai tempi di Nelson l'Ammiragliato britannico gestiva migliaia di navi con qualche decina di impiegati, mentre oggi accade esattamente il contrario.

Peter, L. J. - Da lui prende il nome il celebre Principio, anche se il libro che lo illustra - pubblicato in Italia da Garzanti - è stato scritto a quattro mani con R. Hull.

Poincaré, Jules Henry (1854 - 1912)- "Ma se ci vogliono ventisette equazioni per stabilire che 1 è un numero, quante ne saranno necessarie per dimostrare un vero teorema?". La risata di Poincaré, grande matematico francese degli inizi del XX secolo, risuona ancor oggi devastante contro la pretesa di scrivere matematica (o di specificare programmi) in un linguaggio formale.

Sarson, Patricia detta **Trish** - E' a lei e - in ordine rigorosamente alfabetico - a Tom DeMarco, Chris Gane e Ed Yourdon che dobbiamo i DFD e l'Analisi Strutturata. DeMarco, Gane e Yourdon sono ancora sulla cresta dell'onda, mentre di Trish si sono un po' perse le tracce.

Schussel, George - Futurologo tra i più accreditati. Quando l'ho incontrato era a capo di Digital Consulting Inc.

Toffler, Alvin - Futurologo e critico sociale di grande fama. E' stato Associate Editor di *Fortune* e autore di libri di successo, alcuni dei quali pubblicati in Italia come *The Thirt Wawe* e *Powershift* (*Powershift*, *la Dinamica del Potere*, Sperling & Kupfer, 1991).

Voltaire (nome d'arte di François Marie Arouet, 1694 - 1778) - Illustre precursore di Murphy. Nel suo *Candide* (1759) si prende infatti gioco dell'ottimismo leibnitziano (*Nihil omnino fit sine aliqua ratione*) impersonato dal Dottor Pangloss: "Tutti gli avvenimenti sono concatenati nel migliore dei mondi possibili".

Weinberg, Gerald M. - Di lui si sa poco, anche perchè vive e lavora in un luogo tanto improbabile come Lincoln nel Nebraska. Ecco come Ed Yourdon, che una volta è andato a trovarlo, descrive il panorama che si può osservare dalla veranda della sua casa: "E' così inesorabilmente piatto che basterebbe alzarsi sulla punta dei piedi perchè lo sguardo giunga fino alla West Coast".

Yourdon, Edward - Una delle voci più ascoltate dell'Informatica d'oltre oceano. Autore di libri di successo (pubblicati anche qui in Italia), oltre che consulente e conferenziere di fama internazionale, è uno dei padri dell'Analisi Strutturata. Ma è stato poi contagiato dalla febbre dell'OO.