Lo Zen e l'arte della stabilità dei personal computer

[Dopo avere letto questo articolo potrete dare una pronta risposta a chi vi chiederà "perché il mio computer si blocca sempre?" o a chi continua ad affermare che i programmatori sono incapaci di fare un programma decente...]

- Introduzione -

Lo scritto che avete di fronte nasce per caso da uno scherzo di una collega che, un giorno, esasperata dal comportamento anomalo del suo PC e trovandosi tra le mani una specie di barzelletta sui computer, colse l'occasione per inviarmela via e-mail nel tentativo di prendere un po' in giro tutto il dipartimento dell'EDP. La storiella, che non riporto integralmente in quanto non ne conosco l'autore e non vorrei incorrere in problemi di copyright :-), si concludeva con la frase seguente:
" Il paradosso metafisico, ma non troppo, vuol far riflettere fino a che punto l'uomo sarà disponibile al condizionamento dalle macchine che storicamente erano strumenti per liberarci dalla fatica , ma che attualmente sembra abbiano preso coscienza della nostra dipendenza e ci costringono a lavorare per migliorare loro stesse in una sorta di sottile perfida vendetta."

- Disagio Tecnologico -

La frase finale riportò il mio pensiero ad un testo di Alan Cooper, dal titolo "il disagio tecnologico", da me letto qualche tempo prima. In tale testo si evidenzia (con abbondanza di esempi) la frustrazione provata dall'utente comune nel tentativo di addomesticare i computer e far loro eseguire i compiti desiderati. Cooper, come molti altri, ipotizza la necessità di uno sforzo ulteriore da parte dei programmatori per migliorare l'interfaccia utente allo scopo di diminuire questa sensazione di disagio provata dagli utenti. Nelle conclusioni estreme si arriva ad ipotizzare e auspicare l'eliminazione della necessità di una "interfaccia" per fare eseguire i corretti compiti ai PC. Egli infatti attribuisce la causa di tutti i mali che affliggono il mondo informatico ad una cattiva interfaccia utente. Ritengo che questa sia solo una mezza verità, e una mezza conferma a questo pensiero arriva dal fatto che il mondo dell'Open Source è estremamente ricco di esempi di programmi molto stabili ed efficienti e dotati di un' interfaccia scarna, se non addirittura inesistente.

Comunque questa serie di associazioni di idee meritava qualche approfondimento e mi misi ad analizzare il problema dell'interfaccia utente e delle cause del problema dell'instabilità dei sistemi computerizzati. Più mi addentravo nell'argomento più i punti da tenere in considerazione si moltiplicavano. Il contesto è ampio e si deve tenere conto delle differenze del modello elaborativo di un essere umano e quello di un computer, dell'inadeguatezza dei linguaggi di programmazione e degli strumenti di analisi e di chissà quali parametri che il sottoscritto non è nemmeno in grado di immaginare.

Non avevo intenzione di addentrarmi molto nell'argomento anche perché sembrava più facile uscire dalle sabbie mobili che dalla mole di argomenti e di informazioni che stavo toccando. Era inoltre necessario fare i dovuti approfondimenti di tutti i punti esaminati. Con questo obiettivo nell'angolo del cervello iniziai quindi a prendere appunti prima mentali e poi scritti fino ad arrivare a stendere lo scritto presente.

-Differenze uomo macchina - i modelli -

Qualunque utilizzatore di computer tende istintivamente ed inconsapevolmente ad assegnare delle facoltà cognitive a questo tipo di apparati. In realtà il computer è intrinsecamente una macchina stupida e le capacità di ragionamento che gli attribuiamo sono, al limite, un mero riflesso di un modello comportamentale che il programmatore ha riversato all'interno del programma mentre definiva la logica dello stesso. Già qui si potrebbe dedurre una prima giustificazione ai "comportamenti" anomali riscontrabili in certe situazioni: errare è umano, i programmatori sono (al di là delle affermazioni di molti) esseri umani. Di conseguenza i programmatori commettono degli errori ed i programmi contengono il risultato di questi errori.

Come si è detto quindi il computer non è intelligente e i programmi sono le istruzioni operative che i programmatori gli riversano. Essi devono quindi a priori stabilire le reazioni ad ogni variante immessa dagli operatori. In un programma che non sia una computazione banale, infatti, l'operatore interagirà, nel tempo, con il sistema inserendo valori e parametri che andranno a modificare il flusso del programma durante l'esecuzione dello stesso (ovvero a run-time - per gli anglofoni). E' chiaro che si avranno risultati tanto più coerenti con le attese quanto più il programmatore avrà saputo prevedere tutte le possibili variabili/varianti operative del programma stesso.
A complicare le cose (pur senza tenere ancora conto del grado di complessità dei sistemi) bisogna tenere presente che i computer non sono applicati direttamente alla realtà ma a dei modelli di casi reali. Per quanto perfetti, tali modelli, non saranno mai totalmente vicini alla realtà. Tale similitudine spesso non è nemmeno auspicata in quanto si desidera operare in un ambiente che simuli un sottoinsieme delle situazioni possibili nella realtà rappresentata al fine di ridurre il numero di elementi di

cui tenere conto in fase di programmazione o in fase operativa.

I programmi sono quindi utilizzati per generare/rappresentare dei modelli sia che si tratti della rappresentazione dei movimenti delle merci di un azienda, sia che si tratti della descrizione dei movimenti di un conto corrente bancario, così come di un modello di reazione chimica. In tali modelli si potrà tenere conto di un numero elevatissimo di variabili e parametri ma non si potrà mai riuscire a rappresentare tutte le situazioni possibili nel mondo reale. Ne consegue che, qualora la situazione di input sottoposta dall'operatore ricadesse in un caso non correttamente previsto dal programmatore o comunque non completamente aderente al modello da questi ipotizzato, il risultato della elaborazione si allontanerebbe da quello corretto e potrebbe addirittura portare il sistema in una situazione di instabilità in quanto il flusso corretto del programma è ormai compromesso. Il caso più noto è quello del loop ovvero, il programma a causa delle variabili inserite, è entrato in un cerchio di istruzioni limitato e le esegue in continuazione senza portare alcuna variazione ai valori che consentirebbero al programma stesso di modificare il flusso dell' elaborazione ed uscire da questa situazione. In mancanza di un feedback visivo verso l'operatore questo loop non è nemmeno percepibile dall'esterno e quindi l'unico effetto visibile è quello di un computer irreversibilmente bloccato. Se a tale situazione si aggiunge un sistema operativo mal costruito, nel quale il blocco di una elaborazione si riflette al S.O. stesso e quindi costringe al riavvio dell'elaboratore, si comprende la frustrazione provata da alcuni utilizzatori che sperimentano questa situazione più volte al giorno e che ogni volta perdono parte del loro lavoro. Anche in questo caso il mondo Open Source porta degli esempi di valide soluzioni grazie alla disponibilità di un certo numero di sistemi *NIX-like che per filosofia costruttiva fanno lavorare i programmi in task separate e quindi ovviano alla possibilità di un blocco totale del sistema (agli addetti ai lavori questa sembrerà una semplificazione eccessiva ma non vorrei dedicare un libro all'argomento...). Un'altro aspetto fondamentale di cui tenere conto è l'adattabilità del cervello umano rispetto ad un calcolatore. Due sono gli elementi in questo caso: il primo è dato dal fatto che per un elaboratore gli schemi sono definiti in maniera rigida dal flusso del programma mentre, un essere umano, può improvvisare e reagire a nuovi elementi in maniera autonoma. Qualcuno potrebbe dissentire osservando che l'educazione umana è a suo modo un programma rigido che in alcuni casi condiziona il cervello umano ad alcune reazioni. Questo tipo di argomentazione rende ancora più valida la possibilità di confrontare il comportamento umano e quello di un calcolatore e rende ancora più evidente la capacità del primo ad adattarsi a nuove situazioni.
Il secondo è dato dal fatto che un programma può solo elaborare dati logici e binari che ricadono entro i limiti architetturali della macchina stessa, mentre per un uomo ogni grandezza e analogica e uno stesso valore può essere spostato dal suo valore assoluto anche da una scala di "convenienza" che non solo dipende dalla situazione in esame ma anche da una somma di esperienze. Per banalizzare con un esempio se informo il calcolatore che ho a disposizione 1 Kg di zucchero il calcolatore assumerà tale informazione in senso assoluto e mi potrebbe dire nel contesto di un programma di ricette che è sufficiente a preparare la mia torta che richiede 1Kg esatto di zucchero. Un essere umano accetterebbe l'utilizzo di 990 grammi riducendo altri ingredienti o accettando una torta meno zuccherata. Il computer scarterebbe sicuramente tale valore impedendomi di proseguire nella preparazione. L'esempio è di una banalità folle ma può essere rapportato a migliaia di situazioni nelle quali abbiamo ritenuto illogica la mancanza di adattabilità degli elaboratori.

- L'inadeguatezza dei linguaggi di programmazione -

Non si può tralasciare nell'analisi delle cause del disagio tecnologico l'inadeguatezza dei linguaggi di programmazione. Che i linguaggi di programmazione siano inadeguati non è una mia conclusione ma solo l'anticipazione dei ragionamenti e delle definizioni che seguono.
Iniziamo l'analisi proprio dallo scopo iniziale di utilizzo degli elaboratori elettronici. Il termine computer -letteralmente tradotto- indica una macchina calcolatrice ovvero una macchina in grado di eseguire dei calcoli e tale fu lo scopo per il quale il computer fu creato. Fa parte del bagaglio di conoscenza di ogni informatico la nozione che i primi computer non avevano un vero e proprio software e che la loro programmazione avveniva attraverso la modifica dei circuiti elettrici che li componevano. Tali operazioni erano chiaramente riservate ad una classe eletta di operatori che, in realtà, erano, spesso, gli stessi tecnici che avevano costruito tali macchine. I calcoli compiuti da questi computer erano relativamente elementari e le risposte dovevano venire interpretate tramite la lettura delle spie accese in un quadro sinottico.
Avevamo a che fare quindi con un linguaggio di programmazione che "era" la macchina stessa e richiedeva delle conoscenze, estremamente sofisticate, sia della macchina sia del problema di calcolo che si doveva affrontare. L'interfaccia utente semplicemente non era concepita.

Con l'evoluzione le cose migliorarono e si comincio ad avere una programmazione ad interrutori, prima, e a **schede perforate**, poi, con risposte stampate su carta (chiaramente sto riassumendo un evoluzione in poche parole). Dal punto di vista della programmazione erano sempre necessari dei tecnici che comprendessero il funzionamento dell'hardware ma almeno non si doveva ricostruire il computer ad ogni elaborazione. L'interfaccia utente era migliorata notevolmente in quanto una risposta scritta era sicuramente un grandioso passo avanti.

Con l'arrivo dei microprocessori la situazione mutò drasticamente e si comincio a parlare in termine di linguaggi di programmazione. Ma non immediatamente. Chiunque abbia affrontato uno dei tanti personal computer circolanti alla fine degli anni '70 o primi anni '80 può testimoniare che la programmazione avveniva andando a inserire i numeri binari nelle celle di memoria tramite una serie di interrutori e si faceva avanzare il puntatore alla locazione di memoria tramite un pulsante. Il risultato era disponibile su una fila di led (in formato binario). In realtà nello stesso periodo il sistema operativo UNIX era già diffuso nei grossi calcolatori ma la situazione di quei personal rifletteva quella dei primi elaboratori basati su microprocessore. Tali macchine si basavano sugli studi di John Von Neumann che creo un elaboratore composto da una CPU, una memoria di immagazzinamento ad accesso lento (l'antenato degli hard disk) ed una memoria secondaria ad accesso veloce che precorreva la RAM. Questa macchina immagazzinava istruzioni come valori binari e le eseguiva sequenzialmente. Ancora oggi gli elaboratori riflettono questo modello basilare nella loro struttura.
Il codice binario proprio di ciascun processore è, più propriamente, detto linguaggio macchina e rappresenta il livello di programmazione più basso e diretto e richiede sicuramente una totale conoscenza dell'HW. Il primo livello di astrazione è il linguaggio assembly che associa ad ogni istruzione compresa dalla CPU un codice mnemonico che facilita il lavoro dei programmatori e la stesura dei programmi. Il linguaggio macchina è sicuramente il primo livello di interfaccia utente impiegato per facilitare il lavoro sui computer. Il computer comprende solo il linguaggio macchina e qualsiasi compilatore o interprete alla fine trasformerà il programma in istruzioni macchina. Le astrazioni introdotte dai linguaggi di programmazione servono solo per facilitare il lavoro dei programmatori con una sintassi facile da ricordare o una più facile definizione del problema e del relativo algoritmo risolutivo.

Fra i linguaggi di programmazione di alto livello il primo fu il FORTRAN (FORmula TRANslator) che permetteva di esprimere con facilità le elaborazioni di tipo matematico. Il secondo fu il COBOL (COmmon Business Oriented Language) che, quasi per complemento, andava a coprire le elaborazioni per il mondo affaristico/commerciale che il FORTRAN non riusciva ad esprimere adeguatamente.

I primi linguaggi di programmazione si preoccupavano sicuramente più del problema che della gestione dell'interfaccia utente e la loro sintassi riflette questo stato di cose. L'output era limitato al minimo: il risultato dell'elaborazione o la richiesta di una nuova variabile. I programmi erano un insieme di istruzioni sequenziali con un flusso condizionato. Gli strumenti di analisi dell'epoca riflettono la situazione ed il flowcharting era uno degli strumenti più usati per seguire la logica del programma.

Abbandoniamo momentaneamente le riflessioni sull'astrazione operata dai linguaggi di programmazione e tocchiamo per un attimo l'argomento dell'interfaccia utente. Dal contesto sopra illustrato, si comprende come, la volontà/necessità di interagire con l'utente è nata in un periodo relativamente tardo del computing. Le macchine, i linguaggi e gli strumenti di analisi si concentravano in origine sul come far compiere le operazioni alla macchina e minimamente dell'interazione. A queste affermazioni si potrebbe sollevare un coro di obbiezioni in quanto gli studi di ergonomia, di interfacciamento uomo macchina ecc. sono relativamente vecchi ed i risultati degli studi compiuti fin dagli anni '70 ai Xerox Park Labs sono parte della storia dell'informatica. Quello che volevo evidenziare è comunque come la diffusione di massa di concetti relativi alla user-interface o comunque la cura dell'utente finale abbiano atteso la diffusione dei personal computer e la fine degli anni '80 per venire applicati.
Chiusa questa parentesi bisogna passare a notare come i linguaggi di programmazione siano poi nel tempo evoluti verso un' astrazione sempre maggiore fino all'attuale modello ad oggetti che si concentra sui dati del problema e sulle operazioni compiute sui dati stessi, ponendo una totale astrazione sull'hardware se non addirittura sui metodi di scrittura del software. Il limite di tale situazione è dato dal fatto che oggi un livello eccessivo di astrazione porta a programmi sempre più gonfi (fatware) e per niente ottimizzati che tendono a non sfruttare i veloci processori a disposizione. Inoltre vengono introdotti degli strati sempre più complessi di software su software e un errore di uno qualsiasi degli strati sottostanti può portare a instabilità il sistema rendendo difficile l'individuazione del problema.

Una domanda che al profano viene spontanea quando si avvicina a questo mondo è "perché esistono tanti linguaggi di programmazione?" Tale domanda non è di per sè oziosa in quanto anche osservando il mondo Open Source (di seguito O.S.) si nota che essi sono veramente molti e talvolta differiscono molto poco sintatticamente e cambia solo il campo applicativo.
La risposta, non troppo ovvia, si ricollega fra le altre cose alla necessità di astrazione sopra enunciata, ed è che "Nessun linguaggio di programmazione può essere ottimale per la risoluzione di tutti i problemi!". Prendendo in esame anche solo tre delle variabili che un linguaggio deve soddisfare esso dovrebbe essere ottimale per:

1. -una determinata applicazione
2. -un determinato sistema
3. -un determinato gruppo di programmatori

Per ottenere ciò esso dovrebbe raggiungere il massimo grado di specializzazione per ciascuna voce definita se non altro allo scopo di garantire la generazione di un codice ottimale o la migliore facilità di utilizzo. Questo altissimo grado di specializzazione esclude per definizione la possibilità che tale linguaggio possa essere ottimale per un numero elevato di applicazioni o sistemi o gruppo di programmatori. Pertanto i linguaggi di programmazione cercheranno di posizionarsi, ciascuno in una propria fascia, in un punto intermedio fra la specializzazione totale e il linguaggio detto 'general-purpose' ovvero di 'utilizzo generico'.
Bisogna tenere conto che linguaggi estremamente specializzati/ottimizzati hanno generalmente una curva di apprendimento alta e tendono ad tagliare fuori un programmatore generico anche perché la loro specializzazione li rende scarsamente portatili e rende difficile per il programmatore il riutilizzo delle conoscenze acquisite.
Chiaramente la proliferazione dei linguaggi rende difficile per un programmatore specializzarsi su uno solo di essi senza correre il rischio di essere tagliato fuori dal mercato. Non a caso nelle offerte di lavoro oggi viene richiesta la conoscenza di almeno un paio di linguaggi di programmazione. Tale situazione porta chiaramente un ulteriore difficoltà al programmatore che deve seguire le innovazioni dei linguaggi prescelti e deve sobbarcarsi l'onere di aggiornarsi. Ne consegue che alcuni programmi scritti con il linguaggio non preferenziale (fra quelli conosciuti dal programmatore) possono essere più facilmente soggetti all'introduzione di errori. Lo stesso risultato si ottiene se le conoscenze limitate del programmatore lo portano a scegliere il linguaggio a lui più congeniale ma meno adatto alla risoluzione del problema in esame.

Un parametro del quale spesso non si tiene conto fra le cause di instabilità dei sistemi è l'usura del software. Il software ovviamente non ha alcuna parte soggetta ad usura per il suo utilizzo ma subisce comunque un deterioramento nel tempo. Confrontando il ciclo di vita dei componenti industriali con quello del software possiamo notare che i primi presentano un picco di problemi (e quindi di costi) quando il componente è nuovo e ciò accade principalmente per difetti dei prototipi. Segue una fase di stabilità fino ad arrivare al picco massimo dell'usura.
Il ciclo di vita del software presenta un picco iniziale dovuto agli errori iniziali di progetto o di implementazione, una progressiva decrescita verso la stabilità soprattutto grazie al debugging e dei picchi progressivi dovuti alle modifiche. Da ogni modifica si ha infatti un ritorno non verso la stabilità iniziale ma verso un livello stabile peggiore del precedente. Questo rappresenta il degrado del software che decresce la sua qualità ad ogni modifica fino alla necessità di sostituzione.
La soluzione a tale problema, in un dato momento della storia dell'informatica, sembrava essere il riutilizzo del software che sembrava avere trovato la massima espressione nell'utilizzo (e successivo riutilizzo) dei componenti.
Le promesse del riutilizzo dei componenti nel sw non sono state completamente realizzate ed ancora oggi si continua spesso a riscrivere i programmi da zero.Questo è causato soprattutto dalla unicità delle situazioni e dei modelli rappresentati dal programma per cui quanto più è complesso il sistema e tanto più è specializzato maggiore è la difficoltà di riutilizzo del codice già scritto.

Un altro tipo di deterioramento è quello che io definisco inquinamento "ambientale" e si manifesta durante la fase di utilizzo. Ho inventato tale definizione per indicare quel fenomeno per cui a causa di successive installazioni di software un sistema (computer + software) stabile viene inquinato dal software aggiunto. L'inquinamento può essere legato per esempio a sostituzione di librerie dinamiche con nuove versioni non completamente allineate con il resto del sistema operativo o

semplicemente a una riduzione dello spazio su disco conseguente all'aggiunta del nuovo sw. Questo tipo di problema è secondo me il più facilmente risolvibile e nello stesso tempo quello che maggiormente dimostra quanto la diffusione dell'utilizzo del computer possa portare con se nuovi problemi. Il metodo migliore per sfuggire l'inquinamento "ambientale" è quello di riservare il computer per un certo compito primario e una volta che il sistema hardware più software è stabile evitare di alterare tale stabilità installando dei software di contorno. Purtroppo spesso si tende a sfruttare un computer, soprattutto quando esso funziona bene, per fargli svolgere il maggior numero possibile di funzioni e compiti. Il risultato come già detto è quello di spostare il sistema da un equilibrio stabile ad uno instabile e quindi portare più facilmente il tutto verso i funzionamenti anomali che maggiormente indispettiscono l'utente finale.

- L'inadeguatezza degli strumenti di analisi -

Come si è visto in precedenza a proposito dei linguaggi di programmazione abbiamo avuto nel tempo una astrazione sempre maggiore dei linguaggi di programmazione inizialmente per svincolare la logica del programma dall'hardware, in seconda istanza per svincolare il problema dal linguaggio e infine si è avuto la necessità di astrarre gli oggetti dal problema stesso in modo da distinguere con maggior chiarezza i dati dai metodi ad essi applicati e quindi potersi concentrare ancora meglio sui problemi. Una evoluzione analoga si è avuta per i sistemi/metodi di analisi:
nella programmazione (perlomeno in quella non banale) è necessario stendere un progetto prima di iniziare a scrivere del codice. Il progetto di un software passa attraverso una fase fondamentale che è l'analisi. L'analisi a sua volta si appoggia a metodologie e strumenti che si evolvono con tutto il resto del mondo dell'informatica.

Il primo tipo di analisi utilizzata è stata quella classica (o bottom-up) derivata dall'industria automobilistica. Semplicemente si seguiva un problema dall'inizio alla fine descrivendo dei macro-blocchi e risolvendo lungo il percorso i problemi che venivano incontrati. Si passava poi attraverso fasi successive affinando sempre di più la soluzione. Chiaramente tale tipo di analisi era legata a linguaggi di sviluppo strettamente sequenziali e tutto sommato ancora grezzi.
Negli anni 80 si ebbe un boom dei linguaggi e dell'analisi strutturata. L'analisi strutturata fra le altre cose imponeva di creare un modello iniziale molto forte nel quale tutte le procedure e i relativi dettagli dovevano essere perfettamente definiti. Si ebbe un periodo nel quale molti capitali venivano investiti nei modelli iniziali e molti progetti venivano abbandonati ancora prima che una sola riga di codice venisse scritta.
Oggi si tende a ridimensionare parecchio lo sforzo iniziale di analisi e si preferisce creare un modello iniziale ed un prototipo nel minor tempo possibile in modo da poter mostrare immediatamente al cliente/utente finale l'ipotesi di funzionamento del programma finito. Una volta raccolti i primi commenti e perfezionata l'analisi iniziale si passa a sviluppare il progetto sostitutivo e definitivo o, qualora il linguaggio ed il modello iniziale impiegati lo permettano, si passa ad una serie di refinement successivi fino ad ottenere il risultato desiderato.

L'analisi vera e propria è accompagnata da strumenti che sono di ausilio alla stessa. Spesso si tratta di meta-linguaggi neutri che permettono di esprimere il problema astraendosi dalle strutture del linguaggio vero e proprio. A volte questi meta-linguaggi sono notevolmente complessi e impongono al programmatore un ulteriore sforzo di apprendimento per il linguaggio di modellazione. Pur ritenendolo eccezionale io stesso ho avuto notevoli difficolta nell'apprendimento dell'UML uno dei linguaggi di analisi più noti e diffusi. Sospetto comunque che l'efficacia dei mezzi formali di analisi sia lontana dall'essere dimostrata, se non altro visto l'enorme diversità e numero di strumenti che si propongono come "la soluzione definitiva per l'analisi e la progettazione del software". Per contro penso sia facilmente dimostrabile come il costo aggiuntivo di un' analisi preliminare sia spesso scoraggiante per il cliente che invece desidera vedere immediatamente un prototipo concreto del programma. Questa situazione per mia esperienza è molto più marcata in Italia che in altri stati europei e tutto sommato penso che ciò sia riconducibile alla massiccia presenza di P.M.I. sul nostro territorio rispetto ad altre realtà europee.

- Conclusioni -

Chi mi avrà seguito fino a questo punto avrà mille argomenti con i quali dare risposta al quesito dell'instabilità dei PC e alle osservazioni sulle cattive interfacce. Ma vorrei aggiungere (per chi se lo fosse chiesto all'inizio) la risposta che diedi (via mail) alla collega che aveva scritto la storiellina di cui ho parlato all'inizio in quanto penso che in quelle parole scritte d'impulso si possa condensare parte di quanto finora espresso:

"In realtà l'analisi proposta è errata. Infatti gli elaboratori elettronici avevano sgravato l'uomo dalla fatica ed i robot industriali l'avevamo portato al di fuori degli ambienti produttivi nocivi. Il paradosso si è avuto quando il computer è diventato "personal" e l'utenza media ha chiesto agli sviluppatori (che impiegano anni di sacrifici per adattarsi al linguaggio delle macchine al fine di renderle produttive) di adeguare il linguaggio della macchina all'espressività umana. Il modo proprio di esprimersi dell'essere umano è totalmente incostante nel tempo, soggetto a fattori esterni, ambientali e non antepone alla logica pura e al beneficio comune l'ego e altri fattori non riconducibili a formule matematiche e pertanto non esprimibili in linguaggio macchina. Osservo inoltre che i PC impiegano il 90% della loro forza elaborativa per gestire le interfacce utente ed il 10% (o meno) a fare il loro lavoro.
Pertanto concludo che il vero paradosso è nell'affermare che il computer è facile da utilizzare ed è alla portata di tutti e nello sforzo compiuto dai programmatori per renderlo tale..."

Vorrei solo aggiungere, concludendo, che l'automazione non va spinta oltre la sua profittabilità o convenienza. Essa deve avere un giusto ritorno economico che permetta di reinvestire e far crescere le tecnologie stesse. Un eccesso di tecnologia può soffocare le normali procedure aziendali in schemi eccessivamente rigidi e creare disagi nelle comunicazioni intraaziendali. Non ultimo il crescente senso di inferiorità e frustrazione dell'utente rispetto al computer da lui usato può sicuramente diminuirne la produttività vanificando i miglioramenti sui tempi di lavorazione introdotti dall'automazione dei processi.

Spero di non avervi tediato e confido nei commenti che vi invito ad inviarmi...

di Rudi Giacomini Pilon