Tony C.A.R. Hoare

ACM Turing Award per "fondamentali contributi alla definizione ed alla progettazione di linguaggi di programmazione".

Il premio gli è stato consegnato alla Conferenza Annuale dell' ACM in Nashville, Tennessee, il 27 Ottobre 1980, da Walter Carlson, CPresidente del comitato per i Premi.

Una trascrizione della conferenza di Hoare[12] è stata pubblicata nelle *Communications of the ACM*.[2]

I Vestiti Vecchi dell'Imperatore

di Tony C.A.R. Hoare
Communications of the ACM, 1981
(in occasione del conferimento del Turing Award)

Il mio primo e più piacevole obbligo da assolvere con questa conferenza, è quello di esprimere la mia profonda gratitudine alla **Association for Computing Machinery** per il grande onore che ha voluto attribuirmi, e per la presente opportunità di rivolgermi a voi con un argomento di mia scelta.
In realtà si tratta di una scelta davvero difficile!
I risultati scientifici da me ottenuti, così largamente riconosciuti da questo premio, sono già stati ampiamente descritti nella letteratura scientifica.
Anzichè riproporre gli astrusi dettagli tecnici del mio mestiere, vorrei parlarvi in modo informale di me, delle mie esperienze personali, delle mie speranze e dei miei timori, dei miei modesti successi, e anche dei miei meno modesti insuccessi.
Ho imparato di più dai miei errori di quello che possano rivelare le fredde parole stampate di un articolo scientifico, ed ora mi piacerebbe che anche voi possiate ricavare da essi un insegnamento.
Inoltre, dei fallimenti è molto più piacevole sentir parlare dopo qualche tempo;
essi sul momento non risultano così divertenti.
Faccio partire il mio racconto nell'Agosto del 1960, quando divenni un programmatore per un relativamente piccolo fabbricante di computer, una divisione della Elliott Brothers (London) Ltd., dove nei successivi otto anni avrei ricevuto la mia educazione di base in computer science.
Il mio primo compito fu la realizzazione, per il nuovo computer Elliott 803, di una subroutine di libreria per attuare un nuovo metodo di ordinamento interno rapido, da poco inventato da Shell.
Mi piacque molto la sfida che consisteva nel massimizzare l'efficienza del semplice codice macchina ad indirizzamento decimale di quei tempi.
Il mio capo ed istruttore, Pat Shackleton, fu davvero contento del mio programma, una volta che fu completato.
Io allora affermai timidamente che pensavo di avere inventato un nuovo metodo di ordinamento che normalmente avrebbe girato più velocemente di SHELLSORT, senza richiedere molta memoria aggiuntiva. Egli scommise con me mezzo scellino che in realtà ciò non era vero.
Sebbene il mio metodo fosse piuttosto difficile da spiegare, alla fine dovette ammettere che avevo vinto la scommessa.
Scrissi varie altre subroutines di libreria in codice molto compatto, ma dopo sei mesi mi fu dato un compito decisamente più importante, quello di progettare un nuovo e avanzato linguaggio di programmazione ad alto livello per il prossimo computer della ditta, l'Elliott 503, il quale doveva avere lo stesso codice istruzione dell'esistente 803, girando però sessanta volte più veloce.
Nonostante la mia educazione nell'ambito delle lingue classiche, questo era un compito per il quale io ero ancor meno qualificato di quelli che lo affrontano oggi.

Per buona fortuna mi venne in mano una copia del **Report on the International Algorithmic Language ALGOL 60**.
Naturalmente tale linguaggio era ovviamente troppo complicato per i nostri clienti.
Come avrebbero mai potuto capire tutti quei **begin** *e* **end** quando nemmeno il *nostro* venditore era in grado di farlo?
Era più o meno Pasqua del 1961, quando fu proposto a Brighton, in Inghilterra, un corso sull'ALGOL 60 con Peter Naur, Edsger W. Dijkstra, e Peter Landin in qualità di istruttori.
Frequentai questo corso con una mia collega nel progetto del linguaggio, Jill Pym, con il Manager Tecnico della nostra divisione Roger Cook, e il nostro Manager alle Vendite, Paul King.
Fu lì che io appresi per la prima volta che cosa erano le procedure ricorsive, e vidi come programmare il metodo di ordinamento che avevo trovato in precedenza e così difficile da spiegare.
Fu allora che scrissi la procedura, con poca modestia denominata QUICKSORT, sulla quale è fondata la mia intera carriera quale computer scientist. Devo dare il giusto credito al genio dei progettisti di ALGOL 60 che inclusero la ricorsione nel loro linguaggio, e permisero a me di descrivere al mondo la mia invenzione in modo così elegante. Ho sempre considerato come il più alto obiettivo della progettazione di un linguaggio di programmazione quello di permettere che le buone idee si possano esprimere con eleganza.
Dopo il corso ALGOL a Brighton, Roger Cook stava riportando me e i miei colleghi a Londra, quando d'improvviso chiese. “Invece di progettare un nuovo linguaggio, perché semplicemente non implementiamo ALGOL 60?”. Noi tutti fummo subito d'accordo – una decisione, in retrospettiva, molto fortunata per me. Ma sapevamo tutti che non avevamo, in quel momento, le abilità e l'esperienza per implementare l'intero linguaggio, così mi fu affidato l'incarico di progettarne un sottoinsieme ridotto e modesto.
In quel progetto adottai alcuni principi di base che credo siano altrettanto validi oggi come lo erano allora.
1. Il primo principio era la *sicurezza*: Il principio cioè che ogni programma non sintatticamente corretto doveva essere rigettato dal compilatore, e che ogni programma sintatticamente corretto doveva fornire un risultato o un messaggio di errore che fosse deterministico e comprensibile in termini del linguaggio sorgente del programma stesso.
Quindi alcun *dump* di memoria sarebbe mai dovuto essere necessario. Doveva essere logicamente impossibile per qualsiasi programma in linguaggio sorgente far accadere che il computer fosse in uno stato di funzionamento senza controllo, sia al momento della compilazione che durante l'esecuzione.
Una diretta conseguenza di questo principio è che ogni occorrenza di ogni indice di ogni variabile con indice doveva essere in ogni occasione controllata, durante l'esecuzione, con riferimento ai limiti inferiore e superiore dichiarati per il vettore cui si riferiva.
Molti anni dopo chiedemmo ai nostri clienti se essi avrebbero desiderato che noi introducessimo come opzione la possibilità di eliminare questi controlli, a favore di una maggiore velocità di esecuzione per i programmi di *produzione.* Unanimemente essi ci chiesero di non farlo – essi già avevano constatato con quale frequenza gli errori legati agli indici si verifichino nel funzionamento dei programmi *in produzione* quando l'incapacità di rivelarli potrebbe divenire disastrosa.
Noto con timore e disappunto che anche oggi nel 1980 i progettisti di linguaggi e gli utilizzatori non hanno compreso questa lezione. In qualsiasi altro ramo rispettabile dell'ingegneria la mancanza nell'osservare tali elementari precauzioni sarebbe da lungo tempo dichiarata contro la legge.

2. Il secondo principio nel progetto dell'implementazione era la *brevità del codice oggetto prodotto dal compilatore e la compattezza dei dati di lavoro run-time.* Per questo c'era una motivazione molto chiara: la dimensione della memoria principale di qualsiasi computer è limitata, e l'aumento delle sue dimensioni comporta qualche tipo di ritardo ed aumento di spesa. Un programma che sia più grande del limite fissato dall'hardware, anche di una sola parola, non può comunque essere posto in esecuzione, soprattutto perché molti dei nostri clienti non avevano l'intenzione di acquistare memorie di supporto aggiuntive.

Questo principio della compattezza del codice oggetto è ancora più valido oggi, quando i processori sono banalmente economici in confronto delle dimensioni di memoria principale che essi possono indirizzare, e le memorie di supporto sono comparativamente ancora più costose e più lente, secondo rapporti di molti ordini di grandezza. Se come risultato di particolari attenzioni avute nella fase di implementazione l'hardware rimane più potente di quello che può sembrare necessario per una particolare applicazione, il programmatore di applicazioni può quasi sempre trarre vantaggio delle capacità extra disponibili per accrescere la qualità del proprio programma, la sua semplicità, la sua robustezza e la sua affidabilità.
3. Il terzo principio del nostro progetto era che *le convenzioni di ingresso e uscita per le procedure e le funzioni dovevano corrispondere ad altrettanta compattezza ed efficienza di quella possibile per subroutines codificate nel modo più compatto ed attento in codice macchina*.
Il mio ragionamento fu che le procedure erano una delle caratteristiche più potenti di un linguaggio ad alto livello, dato che contemporaneamente semplificavano il compito della programmazione e rendevano più corto il codice oggetto. Non bisognava dunque che vi fosse alcun ostacolo al loro utilizzo il più frequente possibile.
4. Il quarto principio era che *il compilatore doveva richiedere un solo singolo passo.* Il compilatore fu strutturato come una collezione di procedure mutuamente ricorsive, ciascuna in grado di analizzare e tradurre una unità sintattica fondamentale del linguaggio, una istruzione, una espressione, una dichiarazione, e così via. Fu progettato e documentato in ALGOL 60, e quindi codificato in codice macchina decimale usando uno stack esplicitamente per la ricorsione. Senza il concetto di ricorsione dell' ALGOL 60, a quel tempo assai controverso, non saremmo del tutto stati in grado di scrivere quel compilatore .
Posso ancora raccomandare una discesa top-down ricorsiva a passo singolo come metodo di implementazione e come principio di progetto per un linguaggio di programmazione.
Per prima cosa, certamente vogliamo che i programmi siano letti da *persone*, e le persone preferiscono leggere le cose una sola volta in un unico passo.
Come seconda, per l'utilizzatore di un sistema *time-sharing* o *Personal computer,* l'intervallo fra l'introduzione da tastiera del programma (o la sua correzione) e l'avvio dell'esecuzione di quel programma risulta completamente improduttivo. Esso può essere ridotto al minimo dalla alta velocità di un compilatore a passo singolo.
Infine, la strutturazione di un compilatore secondo la sintassi del suo linguaggio in ingresso dà un grande contributo ad assicurare la sua correttezza. A meno che non abbiamo una completa fiducia in questa, non potremo mai vare completa fiducia nei risultati prodotti da uno qualsiasi dei nostri programmi.
Per rispettare questi quattro principi, scelsi un sottoinsieme piuttosto piccolo dell'ALGOL 60.
Via via che il progetto e l'implementazione progredivano, gradualmente scoprii metodi per rendere meno stringenti le restrizioni adottate senza compromettere alcuno dei principi. Così, alla fine, fummo in grado di implementare quasi la completa potenza dell'intero linguaggio., compresa anche la ricorsione, anche se varie caratteristiche vennero rimosse, ed altre furono un pò limitate.
A metà del 963, principalmente quale risultato del lavoro di Jill Pym e di Jeff Hillmore, fu rilasciata la prima versione del nostro compilatore. Dopo pochi mesi cominciammo a meravigliarci e a chiederci se qualcuno stava usando il nostro linguaggio e se teneva in considerazione i nostri successivi occasionali aggiornamenti, che incorporavano metodi operativi migliorati. Solo quando un cliente aveva qualche motivo di reclamo ci contattava, e molti non avevano lamentele. I nostri clienti si sono ora spostati su computer più moderni, e hanno adottato linguaggi più aggiornati e alla moda, ma molti mi hanno più volte parlato dei loro piacevoli ricordi legati al sistema ALGOL Elliott, e l'affezione non è dovuta semplicemente alla nostalgia, ma piuttosto all'efficienza, affidabilità e convenienza di quel primitivo e semplice sistema ALGOL.
Come risultato di questo lavoro sull'ALGOL nell'agosto del 1962 fui invitato a prestare servizio nel nuovo Working Group 2.1 dell' IFIP, cui era delegata la responsabilità della manutenzione e dello sviluppo dell' ALGOL.
Il primo e principale compito del gruppo era quello di progettare un sottoinsieme del linguaggio in

cui venivano rimosse alcune delle sue caratteristiche che meno avevano avuto successo. Anche in quei giorni ed anche con un linguaggio così semplice, riconoscevamo che un sottoinsieme poteva costituire un miglioramento rispetto all'originale.
Io apprezzavo molto la possibilità di incontrare e di ascoltare la saggezza di molti degli originali progettisti del linguaggio. Fui però sbalordito e un pò contrariato per il calore e quasi anche il rancore che erano nelle loro discussioni. Apparentemente il processo di progettazione originale dell'ALGOL 60 non aveva avuto uno sviluppo in quello spirito di spassionata ricerca della verità che la qualità del linguaggio mi aveva portato a supporre. Per dare un pò di respiro interrompendo ogni tanto il tedioso e intrinsecamente polemico compito di progettazione di un sottoinsieme, il gruppo aveva destinato un pomeriggio alla discussione delle caratteristiche che avrebbero dovuto invece essere incorporate nella successiva futura progettazione del linguaggio.
Ciacun membro era invitato a suggerire il miglioramento che egli considerava maggiormente importante.
L' 11 Ottobre del 1963 la mia proposta fu quella di estendere una richiesta dei nostri clienti e cioè di rendere meno stringente la regola dell'ALGOL 60 di dichiarazione obbligatoria dei nomi delle variabili, adottando una ragionevole convenzione di default sull'esempio di quella del FORTRAN. Fui sbalordito dal cortese ma istantaneo e fermo respingimento di questo in apparenza innocente suggerimento: fu sottolineato che la ridondanza dell'ALGOL 60 era la migliore protezione contro gli errori di programmazione e di codifica che potevano essere estremamente costosi da rilevare in un programma iin esecuzione, e risultare ncora più costosi se non venivano rilevati. La storia del razzo spaziale Mariner diretto a Venere , perso a causa della mancanza di obbligatorietà delle dichiarazioni nel FORTRAN, non doveva essere pubblicata che molto più tardi. Fui alla fine persuaso della necessità di progettare notazioni di programmazione in modo da massimizzare il numero di errori che non potevano essere fatti, o che se fatti potevano essere affidabilmente rilevati nel momento della compilazione. Questo avrebbe forse reso più lungo il testo dei programmi.
Ma non importava! Non sareste forse stati contenti se la vostra Fata Madrina vi avesse offerto di agitare la sua bacchetta magica sopra i vostri programmi e rimuovere tutti gli errori in essi contenuti ponendo la sola condizione che voi avreste riscritto ed inserito l'intero programma per tre volte?! Il modo per abbreviare i programmi consiste nell'usare le procedure, non nell'omettere informazione dichiarativa vitale.
Fra le altre proposte per lo sviluppo di un nuovo ALGOL c'era quella che la dichiarazione **switch** dell'ALGOL 60 doveva essere rimpiazzata da una caratteristica più generale, cioè un *array* di variabili con valori di tipo *label* e che un programma poteva essere in grado di modificare i valori di queste variabili mediante una assegnazione. Io ero molto contrario a questa idea, simile al **GO TO** assegnato del FORTRAN, perché svevo trovato un numero sorprendente di sottili problemi nella implementazione anche dei semplici *switches* e *labels* dell'ALGOL 60. Potevo intravvedere ancora più problemi nella nuova caratteristica, compreso quello di ri-saltare indietro all'interno di un blocco dopo che da esso si era appena usciti.
Cominciavo inoltre a sospettare che i programmi che usavano molte *labels* fossero più difficili da capire e da completare correttamente, e che i programmi che assegnavano nuovi valori alle variabili di tipo *label* lo sarebbero stati ancora di più.
Mi era accaduto che la notazione più appropriata per rimpiazzare lo *switch* dell'ALGOL 60 dovesse essere basata su quella della espressione condizionale dell'ALGOL 60 , che sceglie fra due azioni alternative in base al valore di una espressione booleana.
Così suggerii la notazione per una espressione "*case"* che sceglie fra un numero qualsiasi di alternative in base al valore di una espressione intera. Questa fu la mia seconda proposta per il progetto del linguaggio. Sono ancora molto orgoglioso di essa, perché essenzialmente non fa sorgere alcun problema né per l'implementatore, il programmatore o il lettore di un programma.
Ora, dopo più di quindici anni vi è la prospettiva di una standardizzazione internazionale di un linguaggio che incorpora questa notazione – un intervallo di tempo rimarchevolmente *corto* se confrontato in relazione ad altre branche dell'ingegneria.
Torniamo ora di nuovo al mio lavoro presso la Elliott,

Dopo l'inatteso successo del Compilatore ALGOL, i nostri pensieri si rivolsero ad un progetto più ambizioso: provvedere un ambiente di software di sistema operativo per configurazioni più grandi del computer 503, con lettori di schede, stampanti, nastri magnetici, e anche il supporto di una memoria ausiliaria a nuclei due volte più economica e due volte più grande della memoria principale, anche se quindici volte più lenta.
Questo progetto sarebbe stato noto con il nome di sistema software Elliott 503 Mark II. Esso comprendeva :
1. Un assemblatore per un linguaggio *assembly* simbolico in cui tutta la parte rimanente del software doveva essere scritta.
2. Uno schema per la supervisione e gestione degli *overlay* di codice e dati, sia dal nastro magnetico che dalla memoria a nuclei di supporto. Questo schema doveva essere usato dal resto del software.
3. Uno schema di *buffering* automatico di tutto l'*input* e l'*output* da e verso ogni disponibile dispositivo periferico – di nuovo, anche questo schema doveva essere usato da tutto l'altro software.
4. Un*a gestione del file system* sul nastro magnetico con *facilities* per l' *editing* ed il *job control.*
5. Una implementazione completamente nuova dell' ALGOL 60, che eliminasse tutte le restrizioni non-standard che avevamo imposto nella nostra prima implementazione.
6. Un compilatore per il FORTRAN, nello stato in cui il linguaggio era allora.
Scrissi vari documenti che discutevano i concetti rilevanti e le possibilità offerte, e li recapitammo ai clienti esistenti ed a quelli che in prospettiva lo potevano diventare. Il lavoro iniziò don un team di quindici programmatori, e la scadenza finale per il rilascio fu fissata a circa diciotto mesi più avanti, nel marzo del 1965.
Dopo aver cominciato il progetto del software Mark II, fui improvvisamente promosso al vertiginoso rango di Assistente Ingegnere Capo, responsabile dello sviluppo e progettazione avanzata dei prodotti della compagnia, sia hardware che software. Sebbene io fossi ancora gestionalmente responsabile per il software 503 Mark II, gli dedicai meno attenzione rispetto ai nuovi prodotti della compagnia, e quasi mancai di accorgermi quando la scadenza per il suo rilascio passò senza che si verificasse alcun evento significativo. I programmatori aggiornarono le loro tabelle dei tempi di implementazione, ed una nuova data di rilascio fu stabilita per circa tre mesi più tardi, a Giugno 1965.
Senza bisogno di dirlo, anche quella data passo ' senza alcun accadimento degno di nota. In quel momento i nostri clienti stavano divenendo piuttosto arrabbiati, e i miei *managers* mi sollecitarono a prendermi carico *personalmente* del progetto.
Io chiesi ancora una volta a tutti i programmatori senior di rivedere e aggiornare le loro previsioni dei tempi di realizzazione, e ciò di nuovo mostrò che il software avrebbe potuto essere rilasciato entro altri tre mesi. Volevo disperatamente crederci, ma semplicemente non potevo.
Lasciai da parte l'esame delle previsioni e cominciai ad indagare più a fondo sull' intero progetto. Ne risultò che avevamo sbagliato non facendo un qualsiasi piano complessivo per l'allocazione della nostra risorsa più limitata, la memoria centrale. Ciascun programmatore si aspettava che questo venisse fatto automaticamente, o dall'assemblatore simbolico oppure dallo schema di overlay automatico. Ancora peggio, avevamo semplicemente trascurato di conteggiare lo spazio usato dal nostro stesso software che stava già occupando la memoria centrale del computer, non lasciando quindi spazio per i nostri utenti per far girare i *loro* programmi. Limitazioni legate alla lunghezza massima degli indirizzi consentita dall' hardware impediva di aggiungere memoria centrale per ampliarla ulteriormente.
Chiaramente le originali specifiche del software non si potevano realizzare, e dovevano essere drasticamente ridimensionate.
Programmatori esperti ed anche manager furono richiamati da altri progetti. Decidemmo di concentrarci dapprima sul rilascio del nuovo compilatore per l'ALGOL 60, che una valutazione accurata mostrò avrebbe richiesto altri quattro mesi.
Io stabilii e convenni con fermezza con tutti i programmatori coinvolti che questa non era più solamente una previsione, ma era invece una promessa; se essi avessero trovato di non essere in grado di mantenere la loro promessa, diventava loro responsabilità personale reperire modi e mezzi

per ottenere comunque un risultato soddisfacente.
I programmatori risposero in modo magnifico alla sfida posta. Essi lavorarono giorno e notte per assicurare il completamento di tutti quei componenti del software che erano richiesti dal compilatore ALGOL. Con nostra grande soddisfazione essi rispettarono la data di consegna prevista; si trattava del primo componente software importante e funzionante prodotto dalla compagnia durante un periodo di due anni.
La nostra soddisfazione ebbe breve durata; in realtà in compilatore non avrebbe potuto essere rilasciato. La sua velocità di compilazione era di soli due caratteri al secondo, un risultato sfavorevole in confronto alla versione esistente del compilatore che operava a circa mille caratteri al secondo. Identificammo subito la causa del problema: si trattava dei ripetuti trasferimenti fra la memoria centrale e la memoria interna di estensione e supporto, la quale era quindici volte più lenta. Fu facile attuare alcuni semplici miglioramenti, ed in capo a una settimana avevamo raddoppiato la velocità di compilazione – portandola a quattro caratteri al secondo. Nelle successive due settimane di investigazione e riprogrammazione, la velocità era di nuovo raddoppiata, a otto caratteri al secondo. Potevamo intravvedere dei modi con cui questo risultato, entro un mese, poteva ancora essere migliorato, ma la quantità di riprogrammazione richiesta era crescente, mentre la relativa efficacia era via-via decresceva; insomma, la via da percorrere era pazzescamente lunga. L'alternativa di accrescere la dimensione della memoria centrale, così spesso adottata in successive situazioni similmente problematiche era impedita da limitazioni nell'indirizzamento dettate dall'hardware.
Non c'erano scappatoie. L'intero progetto software Elliott 503 Mark II doveva essere abbandonato, e con esso uno sforzo di più di trenta anni-uomo di programmazione, praticamente equivalente a una intera vita lavorativa di un uomo; ed io ero responsabile, sia come progettista che come manager, per averlo sprecato.
Fu organizzato una riunione con tutti i nostri 503 clienti, e Roger Cook, che era allora il direttore della divisione *computing*, spiegò loro che non una singola *word* del software da lungo tempo promesso sarebbe mai stata loro rilasciata. Egli adottò nella esposizione un tono completamente tranquillo, e ciò fece si che nessuno dei clienti potesse interromperlo, o commentare a bassa voce nel fondo, o neanche agitarsi sulle sedie. Io rimasi ammirato, ma non potevo condividere la sua calma. Durante la colazione i nostri clienti furono gentili al punto di cercare di confortarmi.
Essi si erano resi conto già da lungo tempo che il software, nelle sue specifiche originali, non avrebbe mai potuto essere loro fornito, che anche se lo fosse stato essi non avrebbero saputo come utilizzare le sue caratteristiche molto avanzate, e che in ogni caso molti grandi progetti del tipo in esame vengono cancellati prima di arrivare ad essere rilasciati.
In retrospettiva, credo invero che i nostri clienti siano stati fortunati, in quanto limitazioni legate all'hardware li avevano protetti dagli eccessi arbitrari che derivavano dai nostri progetti software. Al giorno d'oggi gli utenti dei microprocessori godono di una protezione analoga, ma non per molto tempo ancora.
A quel tempo stavo leggendo i documenti preliminari che descrivevano i concetti e le caratteristiche di base del recentemente annunciato OS360, e del nuovo progetto di sistema time-sharing chiamato Multics. Questi sistemi erano molto più estesi, elaborati e sofisticati di qualsiasi cosa avessi mai immaginato, anche per la prima versione del software 503 Mark II.
Chiaramente IBM e MIT dovevano essere in possesso di qualche segreto, relativo alla progettazione e realizzazione con successo del software, sulla cui natura non potevo nemmeno cominciare a fare ipotesi. Fu solo più tardi che anch'essi si resero conto che nemmeno loro erano in grado di ottenere gli obiettivi che si proponevano.
Così ancora non riuscivo a vedere come avevo potuto causare un così grande problema alla mia compagnia. In quei momenti ero convinto che i miei manager avessero l'intenzione di licenziarmi. Ma in realtà non fu così, essi stavano predisponendo una punizione molto più severa.
“Va bene Tony,” mi dissero. “Tu ci hai messo in questo pasticcio, e adesso tu ce ne tirerai fuori”
Io protestai che non sapevo in che maniera, ma la loro risposta fu semplice.
“Bene, e allora dovrai trovarla.” Essi anche espressero la loro convinzione che ero in grado di

farcela. Io non condividevo tale loro fiducia, ed ebbi la tentazione di dare le dimissioni.
Fu forse la più fortunata di tutte le mie fortunate ritirate il fatto che non lo feci. Ovviamente la ditta fece tutto quello che poteva per aiutarmi. Essi mi alleggerirono della responsabilità della progettazione hardware, e ridussero la dimensione dei miei gruppi di programmazione. Ciascuno dei miei manager espose accuratamente la sua propria teoria su che cosa era stato sbagliato, e tutte le teorie erano tra loro diverse. Alla fine, penetrò nel mio ufficio il manager più esperto e anziano di tutti, un direttore generale della nostra compagnia madre, Andrew St. Johnston. Io fui sorpreso che egli avesse in qualche modo avuto notizia del mio nome. "Sai che cosa è stato sbagliato?" Urlò – egli urlava sempre - "Tu hai lasciato fare ai tuoi programmatori cose che tu stesso non capisci." Io ammutolii per lo sbalordimento. Egli non era ovviamente a contatto e consapevole delle realtà del giorno presente. Come poteva una sola persona mai capire l'interezza di un prodotto software moderno come il sistema software Elliott 503 Mark II?
Mi accorsi più tardi che aveva invece assolutamente ragione ; egli aveva diagnosticato la vera causa del problema, e contemporaneamente aveva seminato il seme della sua successiva soluzione.
Io avevo ancora un team di circa quaranta programmatori e avevamo la necessità di mantenere la buona volontà dei clienti per la nostra nuova macchina, e anche di riguadagnare la fiducia dei nostri clienti per la macchina vecchia. Ma che cosa in realtà avremmo potuto proporci di fare quando sapevamo con sicurezza una sola cosa, che tutti i nostri piani precedenti avevano avuto esito fallimentare? Perciò fissai una riunione con tutti i programmatori senior di una intera giornata per il 22 ottobre 1965, per sviscerare la questione fra di noi.
Conservo ancora le annotazioni prese in quella riunione.
Per prima cosa facemmo un elenco delle principali lamentele dei nostri clienti.
Cancellazione di prodotti annunciati, incapacità di rispettare le scadenze, dimensioni eccessive del software, non giustificate dalla utilità delle possibilità offerte, il non essere in grado di tener conto della retroazione del cliente. Una attenzione più pronta prestata alle richieste (anche assai secondarie) dei nostri clienti avrebbe potuto pagare in dividendi di buona volontà quanto il successo dei nostri piani più ambiziosi.
Elencammo poi le nostre proprie lamentele. Mancanza di tempo macchina per il test dei programmi, non piena prevedibilità della disponibilità del tempo macchina, mancanza dei richiesti dispositivi periferici, inaffidabilità, quando anche era disponibile, dell'hardware, la dispersione dello staff di programmazione, scarsità dei dispositivi per la perforazione dei programmi, mancanza di date certe per la disponibilità dell'hardware, mancanza di un impegno adeguato per la scrittura tecnica della documentazione, mancanza di conoscenza del software all'esterno del gruppo di programmazione, interferenze da parte dei manager di livello superiore che imponevano decisioni ".. senza una piena comprensione delle più intricate implicazioni della materia cui si riferivano", e un troppo accentuato ottimismo se posti di fronte alla pressione da parte dei clienti o del settore Vendite.
Non cercavamo tuttavia di giustificare il nostro fallimento con queste lamentele. Ammettevamo, ad esempio, che era responsabilità dei programmatori educare i propri manager e gli altri dipartimenti della Ditta, fornendo le informazioni necessarie in una forma facilmente e esattamente comprensibile. "La speranza che le deficienze nelle specifiche originali potessero essere rimediate dalla abilità di un reparto di scrittura e documentazione tecnica erano mal riposte: la progettazione di un programma e la progettazione delle sue specifiche devono essere intraprese in parallelo dalle stesse persone, e devono interagire l'una con l'altra. Una mancanza di chiarezza nella specificazione è una delle più sicure indicazioni di una deficienza nel programma che essa descrive, e i due problemi devono essere rimossi simultaneamente prima che il progetto venga intrapreso. Vorrei aver seguito questo principio nel 1963; E vorrei che tutti lo seguissimo oggi.
Le mie annotazioni sullo svolgimento di quella giornta dell'ottobre 1965 comprendono un capitolo intero dedicato agli errori e ai problemi all'interno del gruppo software. Questo capitolo rivaleggia con la più abietta autocritica di un ufficiale revisionista nella rivoluzione culturale Cinese. Il nostro errore principale fu la troppa ambizione.
Gli obiettivi che ci siamo posti si sono mostrati troppo ovviamente superiori alle nostre capacità di raggiungerli. Ci sono anche stati errori nelle previsioni, nella stima delle dimensioni e della velocità

dei programmi, nell'impegno richiesto, nella pianificazione dell'interazione e del coordinamento dei programmi, nel provvedere con sufficientemente anticipo degli allarmi sul fatto che le cose stavano andando in modo sbagliato. Abbiamo anche sbagliato nel nostro controllo delle variazioni del programma, nella documentazione, nei collegamenti con gli altri dipartimenti, con il nostro management, e con i nostri clienti.
Abbiamo sbagliato nel non dare chiare e stabili definizioni delle responsabilità dei singoli programmatori, e dei vari *project leaders*.
Oh, ma devo continuare? Quello che fu davvero sorprendente è che un gruppo numeroso di programmatori altamente intelligenti potesse lavorare così duramente e così a lungo su un progetto così poco promettente. Sapete, non dovreste fidarvi di noi programmatori intelligenti. Siamo in grado di escogitare argomenti veramente buoni per convincere noi stessi e chiunque altro della ragionevolezza delle più complete assurdità.
E specialmente non credete a noi quando promettiamo di saper ripetere un precedente successo, solo, la prossima volta, ottenendolo più grande e migliore.
L'ultimo capitolo della nostra inchiesta sui motivi del fallimento riguardò i criteri di qualità del software.
“Nel recente sforzo affannato di rilasciare un qualsiasi software purchessia, la prima ovvia vittima è stata la considerazione della qualità del software rilasciato. La qualità del software viene misurata da un certo numero di criteri totalmente incompatibili, che devono essere bilanciati con attenzione nel progetto e nella implementazione di ogni programma.”
Stilammo quindi una lista di non meno di diciassette criteri, che poi è stata pubblicata in un editoriale speciale ospitato nel volume numero 2 della rivista *Software Practice and Experience*.

Come siamo riusciti ad uscire e riprenderci da una tale catastrofe?
Per prima cosa suddividemmo i nostri 503 clienti in gruppi, in base alla natura ed alle dimensioni delle configurazioni hardware che essi avevano acquistato – ad esempio, quelli che avevano nastri magnetici erano tutti in un gruppo.
Poi assegnammo a ciascun gruppo di clienti un piccolo *team* di programmatori, ed al relativo *team leader* fu richiesto di visitare i clienti per determinare che cosa essi volevano, selezionare le richieste più semplici da soddisfare, e fare piani (ma non promesse) per realizzarle. In nessun caso avremmo nemmeno considerato una richiesta che avesse richiesto più di tre mesi per realizzarne la soluzione e fornirla. Il project leader avrebbe dovuto poi convincere me personalmente che la richiesta del cliente era ragionevole, che la progettazione della nuova caratteristica era appropriata, e che le previsioni e la temporizzazione ipotizzate per la realizzazion erano realistiche. E, soprattutto, io non permisi che fosse fatto alcunché che io stesso non avessi direttamente compreso.
Funzionò! Il software richiesto cominciò ad essere rilasciato alle date promesse. Con l'aumentare della nostra fiducia e di quella dei nostri clienti, fummo in grado di intraprendere il soddisfacimento di richieste leggermente più ambiziose. Nel volgere di un anno eravamo riusciti a tirarci fuori e a recuperare dal disastro. Entro due anni avevamo anche alcuni clienti moderatamente soddisfatti.
Così ci eravamo tratti dal pantano per mezzo di buon senso comune e di accettabili compromessi fino a raggiungere qualcosa che si avvicinava ad un successo.
Ma io non ero soddisfatto. Non vedevo perché il progetto e l'implementazione di un sistema operativo dovessero essere molto più difficili di quelli di un compilatore.
Questa è la ragione per cui ho dedicato le mie ricerche successive ai problemi della programmazione parallela ed ai costrutti dei linguaggi di programmazione che potrebbero aiutare nella strutturazione chiara dei sistemi operativi – costrutti quali i monitor ed i processi comunicanti.
Mentre lavoravo alla Elliott divenni molto interessato nelle tecniche per la definizione formale dei linguaggi di programmazione.
A quel tempo Peter Landin e Christopher Strachey proposero di definire un linguaggio di programmazione con una semplice notazione funzionale, che specificava l'effetto di ciascun comando in una macchina astratta definita matematicamente.
Un tale proposito non era però per me soddisfacente, perché avvertivo che una tale definizione

doveva necessariamente incorporare un buon numero di decisioni di rappresentazione largamente arbitrarie, e non sarebbe stata in linea di principio molto più semplice dell'implementazione del linguaggio per una macchina reale. Come alternativa, Io proposi che la definizione di un linguaggio di programmazione dovesse essere formalizzata come un insieme di assiomi che descrivessero le proprietà desiderate dei programmi scritti in quel linguaggio.
Sentivo che assiomi formulati con cura avrebbero lasciato all'implementazione la libertà necessaria per praticamente costruire il linguaggio per macchine differenti, e allo stesso tempo consentire al programmatore di provare la correttezza dei suoi programmi.
Ma non sapevo vedere come in realtà farlo. Pensavo che sarebbe stato necessario un lungo lavoro di ricerca per sviluppare e applicare le tecniche necessarie, e che una università sarebbe stata un posto migliore rispetto all'industria per condurre una tale ricerca.
Così feci domanda per una cattedra in *Computer Science* alla *Queen's University* di Belfast, dove avrei passato nove anni felici e produttivi. Nell'ottobre del 1968, quando spacchettai le mie carte nella mia nuova casa di Belfast, mi imbattei in una oscura versione preliminare di un articolo di Bob Floyd dal titolo "*Assigning Meanings to Programs.*". Fu davvero un colpo di fortuna. Finalmente potevo intravvedere un modo di ottenre ciò che speravo dalle mie ricerche. E così scrissi il mio primo articolo sull'approccio assiomatico alla programmazione dei computer, pubblicato nelle *Communications of the ACM* nell' Ottobre del 1969.
Proprio recentemente ho scoperto che uno dei primi sostenitori del metodo assertivo per provare la correttezza dei programmi fu nientaltri che lo stesso Alan Turing.
Il 24 giugno 1950, ad una conferenza a Cambridge, egli presentò una breve esposizione intitolata "*Checking a Large Routine*" che spiega l'idea con grande chiarezza.
"How can one check a large routine in the sense of making sure that it's right? In order that the man who checks may not have too difficult a task, the programmer should make a number of definite *assertions* which can be checked individually, and from which the correctness of the whole program easily follows."["Come si può verificare una routine grande, nel senso di assicurarsi che essa è realmente corretta? Affinché la persona che compie la verifica non abbia un compito troppo difficile, il programmatore dovrebbe enunciare un certo numero di asserzioni ben definite, che possano essere verificate individualmente, e da cui la correttezza dell'intero programma possa seguire facilmente"]
Si consideri l'analogia con il controllo di una operazione di somma. Se la somma è data direttamente come una singola colonna di cifre, [con il risultato sotto], uno deve controllare l'intera colonna in una sola sessione. Ma se sono dati invece i risultati delle somme per varie colonne (in cui sono ripartiti tutti i numeri da sommare), [con i riporti da addizionare separatamente], il lavoro di chi fa il controllo è molto più facile, essendo suddiviso nel controllo delle varie asserzioni [e cioè la verifica della correttezza della somma di ciascuna colonna] più quello di una somma molto più corta [dei singoli riporti nel totale]. Questo principio può essere applicato al controllo di correttezza di una routine molto grande, ma qui illustreremo il metodo per mezzo di una routine abbastanza piccola, cioè quella che permette di ottenere il valore di n fattoriale senza far uso di un elemento moltiplicatore.
Sfortunatamente non vi è un sistema di codifica noto in modo a sufficienza generale per giustificare di esplicitare questa routine completamente, ma un diagramma di flusso sarà sufficiente a scopo di illustrazione.
Questo mi riporta indietro al tema principale della mia esposizione, e cioè il progetto dei linguaggi di programmazione.
Durante il periodo da agosto 1962 ad ottobre 1966 partecipai a tutte le riunioni del gruppo di lavoro IFIP ALGOL. Dopo il completamento delle nostre fatiche sul sottoinsieme IFIP ALGOL iniziammo la progettazione di ALGOL X, che si intendeva fosse il successore di ALGOL 60.
Furono fatti nuovi suggerimenti, e nuove caratteristiche furono previste, e nel maggio 1965 fu affidato a Niklaus Wirth di riunirli insieme nel progetto di un unico linguaggio.
Ero molto contento della sua bozza di progetto, che evitava tutti i difetti noti di ALGOL 60, e comprendeva varie nuove caratteristiche, le quali, tutte, potevano essere implementate in modo

semplice ed efficiente, ed usate in modo sicuro e conveniente. La descrizione del linguaggio non era però ancora completa. Io lavorai duramente nel preparare suggerimenti per il suo miglioramento, e altrettanto fecero molti altri membri del nostro gruppo. Al momento della riunione successiva a St. Pierre de Chartreuse, in Francia nell'ottobre del 1965, avevamo una bozza di un progetto di linguaggio eccellente e realistico, che fu pubblicata in giugno 1966 come “A Contribution to the Development of ALGOL," nelle *Communications of the ACM*.
Tale progetto fu implementato sull'IBM3 60 e gli fu dato il nome di ALGOL W dai suoi molti soddisfatti utilizzatori . Esso fu non soltanto un degno successore dell' ALGOL 60 ma anche un apprezzabile predecessore del PASCAL.
In quella stessa riunione, il comitato ALGOL aveva posizionato, prima di esso, un breve, incompleto e piuttosto incomprensibile documento, che descriveva un differente, più ambizioso, e per me un molto meno attrattivo linguaggio. Fui molto sorpreso quando il gruppo di lavoro, che consisteva di tutti gli esperti di linguaggi di programmazione più noti in campo internazionale, decise di mettere da parte la bozza commissionata, su cui avevamo tutti lavorato, per ingoiare in un boccone un'esca così poco attraente.
Questo accadeva esattamente una settimana dopo la nostra investigazione sul progetto software 503 Mark II.
Cercai disperatamente di dare avvertimenti contrari all'oscurità, alla complessità ed alla troppo grande ambizione del nuovo progetto, ma i miei avvertimenti rimasero inascoltati.
La mia conclusione è che vi sono due modi di costruire un progetto software.
Un modo è di farlo così semplice in modo che *ovviamente* esso non ha manchevolezze;
Un altro modo è quello di farlo a tal punto complicato cosicché esso non abbia *ovvie* manchevolezze.
Il primo metodo è di gran lunga più difficile. Esso richiede la stessa abilità, la stessa dedizione ed anche la stessa ispirazione della scoperta delle leggi fisiche semplici che sono alla base dei complessi fenomeni della natura. Esso richiede anche la disponibilità ad accettare obiettivi limitati da vincoli fisici, logici e tecnologici, e ad accettare un compromesso nel caso che più obiettivi fra loro in conflitto non si possano raggiungere. Nessun comitato farà mai ciò fino a quando non sarà troppo tardi.
E così fu per il comitato ALGOL. Chiaramente la bozza di progetto che esso preferì non era ancora completa. Così una nuova e finale bozza del nuovo progetto del linguaggio ALGOL fu promessa entro il termine di tre mesi, ed essa avrebbe dovuto essere sottoposta all'esame di un sottogruppo di quattro membri, fra i quali ero compreso anch'io. I tre mesi trascorsero e terminarono, senza che una sola parola accennasse alla nuova bozza. Dopo sei mesi il sottogruppo si riunì in Olanda. Di fronte a noi avevamo ora un documento più lungo e con più pagine, pieno di errori corretti all'ultimo minuto, che descriveva ancora un altro diverso (pa per me ugualmente non attraente) linguaggio. Nicklaus Wirth ed io dedicammo un pò di tempo a far sì che fossero rimosse alcune delle manchevolezze presenti nel progetto e nella sua descrizione, ma invano. La completa e finale bozza del progetto fu promessa per la successiva riunione dell'intero comitato ALGOL che doveva tenersi dopo tre mesi.
Anche questi tre mesi passarono, e nessuna parola della nuova bozza apparve. Dopo sei mesi, nell'ottobre del 1966, il gruppo di lavoro ALGOL si riunì a Varsavia. Davanti a sè aveva un documento ancora più lungo e più spesso, pieno di errori corretti all'ultimo minuto, che descriveva in un modo ugualmente oscuro un ancora differente e per me ugualmente non attraente linguaggio. Gli esperti nel gruppo non riuscivano a vedere i difetti presenti nel progetto, ed essi fermamente risolsero di approvare e adottare la bozza, stimando che essa sarebbe stata completata entro tre mesi. Invano li avvertii che non lo sarebbe stata, Invano li stimolai a rimuovere alcuni degli errori tecnici nel linguaggio, la predominanza dei riferimenti, le conversioni di tipo per default. Ben lontano dal desiderare la semplificazione del linguaggio, il gruppo di lavoro in realtà chiese agli autori di includere caratteristiche ancora più complesse, quali l'*overloading* degli operatori e la gestione della concorrenza.
Quando la progettazione di qualsiasi nuovo linguaggio sta avvicinandosi al suo completamento, c'è

sempre una pazza corsa ad ottenere l'aggiunta di nuove caratteristiche prima della standardizzazione. La corsa è pazza davvero, perché conduce ad un trabocchetto da cui non vi è modo di trarsi fuori.
Una caratteristica che venga omessa può sempre essere aggiunta in un secondo momento, quando la struttura del suo progetto e le sue implicazioni sono completamente e ben comprese. Una caratteristica che sia inclusa prima di essere stata pienamente compresa non può mai essere più tardi rimossa .
Alla fine, in dicembre del 1968, con un senso di cupa depressione, partecipai alla riunione di Monaco di Baviera in cui il nostro mostro dalla lunga gestazione doveva venire alla luce e ricevere il nome di ALGOL 68. A quel momento un buon numero di altri membri del gruppo aveva perso l'iniziale illusione, ma ormai era troppo tardi: il comitato era ormai riempito di sostenitori del linguaggio, che fu inviato per la promulgazione ufficiale da parte dei comitati superiri dell' IFIP. Il meglio che potemmo fare fu inviare assieme ad esso una relazione di minoranza che riportava il nostro ben ponderato punto di vista che " … quale strumento per la *affidabile creazione* di programmi sofisticati il linguaggio era un fallimento. " Questa relazione fu più tardi soppressa da parte dell' IFIP, un atto che mi ricorda i versi di *Hilaire Belloc*,

But scientists, who ought to know	[*Ma gli scienziati, che dovrebbero saperlo,*
Assure us that it must be so.	*Ci assicurano che deve essere così.*
Oh, let us never, never doubt	*Oh, mai, mai dubitiamo*
What nobody is sure about.	*Di ciò di cui nessuno è sicuro*.]

Non partecipai più ad alcuna successiva riunione di quel gruppo di lavoro. Mi fa piacere anche riportare che il gruppo presto venne ad accorgersi che vi era qualcosa di sbagliato nel loro linguaggio e nella sua descrizione; essi faticarono duramente ancora per sei anni per produrre una descrizione rivista del linguaggio. Si tratta in questo caso di un notevole miglioramento, ma ho paura che, almeno nel mio modo di vedere le cose, la revisione non elimini i problemi tecnici presenti nei fondamenti dell'intero progetto, e nemmeno inizi ad inquadrare il problema della sua inaffrontabile complessità.
I programmatori sono sempre circondati dalla complessità; non possiamo evitarla. Le nostre applicazioni sono complesse, poiché abbiamo l'ambizione di usare i nostri computers in modi sempre più sofisticati. La programmazione è complessa a causa del grande numero di obiettivi in conflitto fra loro per ciascuno dei nostri progetti di programmazione. Se il nostro strumento fondamentale, il linguaggio nel quale progettiamo e codifichiamo i nostri programmi, è anch'esso complicato, il linguaggio stesso diventa parte del problema anziché parte della sua soluzione.
Ora vorrei parlarvi di ancora un altro progetto di linguaggio troppo ambizioso.
Tra il 1965 ed il 1970 fui membro e poi anche presidente del Comitato Tecnico no. 10 della European Computer Manufacturers Association. Fummo incaricati dapprima di una indagine conoscitiva esterna e poi del processo di standardizzazione di un linguaggio che in qualche modo fosse la sintesi finale di tutti i linguaggi, progettato in modo da soddisfare le esigenze di tutti i tipi di applicazioni per computer, sia commerciali che scientifiche, dal più importante costruttore di computer di sempre. Io avevo studiato con interesse e sorpresa, ed anche con un tocco di divertimento, i quattro documenti iniziali che descrivevano un linguaggio denominato NPL apparsi fra il 1 marzo ed il 30 novembre 1964. Ciascuno , nelle proprie speculazioni e desideri , era più ambizioso e assurdo del precedente. Poi il linguaggio cominciò ad essere implementato, ed apparì una nuova serie di documenti, con intervalli regolari di sei mesi, ciascuno dei quali descriveva la versione finale provvisoria congelata del linguaggio, sotto il suo nome finale congelato PL/1.
Ma, per me, ciascuna revisione del documento semplicemente mostrava quanto distante dallo stato iniziale l' implementazione ad alto livello era progredita. Quelle parti del linguaggio che non erano già state implementate erano ancora descritte in prosa fiorita dal libero fluire, che prometteva inalterata delizia. Nelle parti che *erano* state implementate, i fiori erano appassiti; essi erano soffocati da un rigoglioso sottobosco di note a piè di pagina che ponevano arbitrarie e spiacevoli restrizioni sull'uso di ciascuna caratteristica, e che caricavano sul programmatore la responsabilità di controllare i complessi e non attesi effetti collaterali e gli effetti della interazione con tutte le altre

caratteristiche del linguaggio.
Infine, l' 11 marzo 1968, la descrizione del linguaggio fu nobilmente presentata al mondo in attesa come quella di un degno candidato per la standardizzazione. Ma non lo era. Essa era già stata sottoposta a circa settemila correzioni e modifiche per mano dei suoi iniziali progettisti. Altre dodici edizioni furono necessarie prima che essa fosse finalmente pubblicata come uno standard nel 1976. Temo che ciò accadde non perché ciascuno di coloro che rano coivolti fosse soddisfatto del progetto complessivo, ma invece perché tutti erano profondamente annoiati e disillusi.
Per tutto il tempo in cui io fui coinvolto in questo progetto, proposi l'urgenza che il linguaggio fosse semplificato, se necessario suddividendolo in sottoinsiemi, in modo che il programmatore professionale fosse in grado di comprenderlo pienamente, e di assumersi la responsabilità per la correttezza e la determinazione ragionevole del costo dei suoi programmi. Sollecitai inoltre che le caratteristiche potenzialmente pericolose quali la presenza dei *defaults* e le ON-*conditions* fossero eliminate.
Ero certo che sarebbe stato impossibile scrivere un compilatore completamente affidabile quando la correttezza di ciascuna parte del programma fosse dipendente dal verificare che ogni altra parte del programma avesse evitato tutte le trappole ed i trabocchetti del linguaggio.
Dapprima sperai che un tale non del tutto tecnicamente solido progetto avrebbe collassato, ma presto mi resi conto che esso era condannato al successo,. Praticamente qualsiasi cosa nel software può essere implementata, venduta ed anche usata se è data una sufficiente determinazione.
Non vi è nulla che un semplice scienziato possa dire che costituisca un ostacolo contrapposto al flusso di cento milioni di dollari.
Ma vi è una qualità che non può essere comperata in questo modo, ed è la affidabilità. Il prezzo della affidabilità sta nel perseguire la massima semplicità. Ed è un prezzo che il molto ricco trova assai difficile da pagare.
Tutto questo accadde molto tempo fa. Può essere ciò considerato rilevante in una conferenza dedicata ad una anteprima della *Computer Age* che ci sta davanti? La mia più grande paura è che lo può essere. Gli errori che sono stati compiuti negli ultimi venti anni stanno venendo ripetuti oggi ed anche su una scala più ampia. Mi riferisco alla intrapresa della progettazione di un linguaggio che ha generato documenti con i nomi *strawman*, *woodenman*, *tinman*, *ironman*,*steelman*, *green* ed infine ora ADA. Questo progetto ha avuto origine ed è stato sponsorizzato da una delle più potenti organizzazioni del mondo, il Dipartimento della Difesa degli Stati Uniti. Quindi ad esso sono assicurate influenza ed attenzione in modo completamente indipendente dai suoi meriti tecnici ed i suoi errori e manchevolezze ci pongono una sfida con pericoli molto maggiori. Infatti nessuna delle evidenze disponibili fino ad ora può ispirare la fiducia che questo linguaggio ha permesso di evitare qualcuno dei problemi che hanno afflitto altri progetti di linguaggi complessi nel passato.
Io ho fornito la mia miglior possibile consulenza a questo progetto fin dal 1975. All'inizio avevo grandi speranze. Gli obiettivi originali del linguaggio includevano l'affidabilità la leggibilità dei programmi, la formalizzazione della definizione del linguaggio, e anche la semplicità.
Gradualmente però questi obiettivi sono stati sacrificati in favore della potenza, che si suppone ottenuta con una pletora di caratteristiche e di convenzioni di notazione, molte delle quali non necessarie, ed alcune di esse, quali la gestione delle eccezioni, anche pericolose. Viviamo nuovamente la storia del progetto dell'auto a motore. Gli accessori e i lustrini prevalgono sulle preoccupazioni fondamentali che sono quelle della sicurezza e dell'economia.
Non è comunque troppo tardi! Credo che con una potatura attenta del linguaggio ADA sia ancora possibile selezionare un sottoinsieme molto potente che sarebbe affidabile ed efficiente nell'implementazione, e sicuro ed economico nell'uso.
Gli sponsor del linguaggio hanno dichiarato in modo inequivoco, tuttavia, che non vi saranno sottoinsiemi. Questo è il paradosso più strano dell'intero strano progetto. Se si vuole un linguaggio non partizionato in sottoinsiemi, è necessario farlo *piccolo*. Vi si includono solo quelle caratteristiche che si sa sono necessarie per *ogni* singola applicazione del linguaggio e che si sa sono appropriate per *ogni* singola configurazione hardware su cui il linguaggio è implementato. Poi estensioni possono essere progettate in modo speciale quando necessario per particolari

dispositivi hardware e per particolari applicazioni.
Questa è la grande forza del PASCAL, e cioè che vi sono in esso così poche caratteristiche non necessarie, e praticamente nessuna necessità di sottoinsiemi.
Ed è per questo che il linguaggio è abbastanza forte da poter supportare estensioni specializzate - Concurrent PASCAL per il lavoro *real-time*, PASCAL PLUS per la simulazione di eventi discreti, UCSD PASCAL per *workstations* a microprocessore.
Se solo potessimo apprendere le lezioni giuste dai successi del passato non avremmo necessità di impararle dai nostri fallimenti.
E così la parte migliore dei miei consigli agli originatori e progettisti di ADA è stata del tutto ignorata. In quest'ultimo tentativo io mi appello a voi, rappresentanti della programmazione come professione negli Stati Uniti, e cittadini che hanno a cuore il *welfare* e la sicurezza del vostro paese e dell'umanità: non permettete che questo linguaggio, nel suo stato presente, sia usato in applicazioni in cui la affidabilità è un fattore critico, e cioè centrali nucleari di generazione di potenza, missili vettori balistici, sistemi di preavviso e di allarme remoto, sistemi di difesa anti-missili balistici.
Il prossimo razzo che va fuori rotta come risultato di un errore nel linguaggio di programmazione può non essere un razzo spaziale in missione di esplorazione, in viaggio senza conseguenze pericolose verso Venere, ma può essere una testata nucleare da guerra che esplode sopra una delle nostre città.
Un inaffidabile linguaggio di programmazione che produce programmi inaffidabili costituisce un rischio molto più grande per il nostro ambiente e per la nostra società di quanto lo siano automobili non sicure, pesticidi tossici, o incidenti presso centrali di potenza nucleari.
Siate vigilanti per ridurre questo rischio, e per non accrescerlo.
Permettetemi ora di non terminare su questa nota triste.
Che il proprio migliore avvertimento sia ignorato è il fato comune di tutti quelli che assumono il ruolo di consulenti, fin da quando Cassandra sottolineò i pericoli del portare un cavallo di legno entro le mura di Troia.
Questo mi richiama alla mente una storia che ero abituato a sentire nella mia fanciullezza.
Per quello che mi ricordo, il suo titolo era:

I vestiti Vecchi dell'Imperatore.

Molti anni fa c'era un imperatore che era così fortemente appassionato del vestire da spendere tutto il suo denaro in abiti. Egli non si curava di soldati o cose di guerra, non partecipava a banchetti e non esercitava la giustizia nella corte.
Di ogni altro re o imperatore si sarebbe potuto dire, “E' seduto in consiglio”, ma di lui veniva sempre detto, “L'imperatore è seduto nel suo guardaroba.” E così infatti egli era.
In una occasione sfortunata era stato indotto ad uscirne fuori, completamente nudo, con suo grande disappunto e imbarazzo, e con il soorpreso divertimento dei suoi sudditi. Egli prese allora la decisione di non lasciare mai il suo trono, e per evitare una eventuale nudità ordinò che ciascuno dei suoi nuovi vestiti dovesse essere semplicemente drappeggiato sopra quello più vecchio.
Il tempo passava piacevolmente nella grande città che era la sua capitale. Ministri e cortigiani, tessitori e sarti, visitatori e sudditi, cucitrici e ricamatrici, andavano e venivano da e per la sala del trono, secondo i loro vari compiti, e tutti invariabilmente esclamavano, “Quale magnifico abbigliamento ha il nostro Imperatore.”
Un giorno al più anziano e fedele Ministro dell'imperatore giunse la notizia di un sarto assai rinomato che insegnava in un antico istituto di superiore arte del cucito, che aveva sviluppato una nuova arte di ricamo astratto, usando punti e cuciture così raffinati che nessuno era in grado di affermare se in realtà essi vi fossero effettivamente o meno.
“Questi devono essere invero splendidi cuciti,” pensò il Ministro. “Se solo possiamo ingaggiare questo sarto per avere dei consigli, porteremo gli ornamenti del nostro Imperatore a tali altezze di ostentazione che tutto il mondo intero lo riconoscerà come il più grande Imperatore che ci sia mai

stato.” E così l'onesto vecchio Ministro si accordò con il maestro sarto, con notevole spesa. Il sarto fu portato nella sala del trono dove egli porse le dovute riverenze al cumulo di belle stoffe che ora completamente ricopriva il trono. Tutti i cortigiani erano in ansiosa attesa dei suoi suggerimenti. Si immagini il loro stupore quando fu noto che il suo consiglio non consisteva nell' aggiungere sofisticazione ed ancora più elaborata opera di ricamo a quella che già esisteva, ma invece fu quello di rimuovere strati di fronzoli e di ricercare semplicità ed eleganza al posto di stravagante elaborazione.
“Questo sarto non è l'esperto che egli sostiene di essere”, essi mormorarono. “Le sue capacità si sono offuscate per la lunga contemplazione entro la sua torre d'avorio, ed egli non capisce più le necessità sartoriali di un Imperatore moderno.” Il sarto argomentò a lungo e con forza per stabilire il buon senso dei suoi suggerimenti, ma non riuscì a farsi ascoltare. Alla fine egli accettò il suo compenso e se ne tornò alla sua torre d'avorio.
Mai fino ad oggi di questa storia è stata detta l'intera verità. E cioè che, un bel mattino, quando l'imperatore si sentì annoiato ed accaldato, egli si districò con attenzione da sotto quella montagna di stoffe, e ora vive felicemente come guardiano di porci in un'altra storia.
Il sarto dal canto suo è canonizzato come santo patrono di tutti i consulenti, perché nonostante gli enormi compensi che e riuscito a ricavarne, non è mai stato in grado di convincere i suoi clienti della sua nascente convinzione che i loro vestiti non contengono l' Imperatore.

1. [http://en.wikipedia.org/wiki/Tony_Hoare].

2. ^ ***a*** ***b*** C.A.R. Hoare (February 1981). "The emperor's old clothes" (PDF). *Communications of the ACM* **24** (2): 5–83. doi:10.1145/358549.358561. ISSN 0001-0782.

9. ^ Knuth, Donald: Structured Programming with Goto Statements. *Computing Surveys* **6**:4 (1974), 261–301.

12. ^ Hoare, Charles Anthony Richard (1980-10-27). "The Emperor's Old Clothes / The 1980 ACM Turing Award Lecture". Association for Computing Machinery. Archived from the original on 2012-02-03.