RIVISTA TECNICA SELENIA

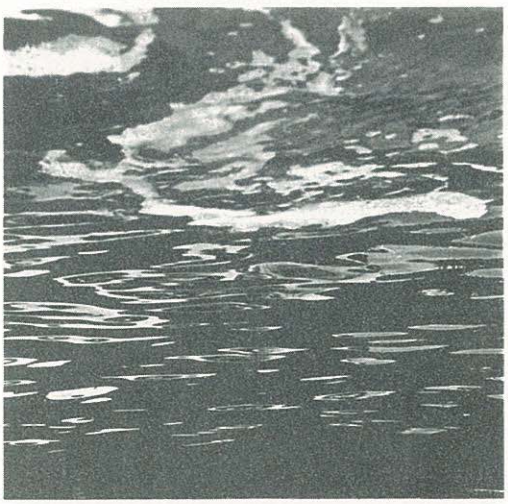
edita dalla **SELENIA, industrie elettroniche associate S.p.A.**

Volume 3 n. 2 1976

RIVISTA TECNICA **SELENIA**

Vol. 3, N. 2, 1976
Periodico trimestrale a cura della
SELENIA, Industrie Elettroniche Associate S.p.A.

SOMMARIO



Edizione della Selenia,
Industrie Elettroniche Associate S.p.A.
Via Tiburtina km 12,400 – 00131 Roma

L'AFFIDABILITÀ DEL SOFTWARE

ROBERTO SOMMA
Direzione Controllo di Qualità

SOMMARIO

Dato il crescente uso di elaboratori nella gestione di sistemi, occorre estendere i concetti dell'Affidabilità al loro software.

Questo articolo ha lo scopo di fare un punto della situazione attuale nei riguardi di tale problema. Viene pertanto dapprima esaminato un modello del software dal punto di vista affidabilistico, vengono quindi descritti due metodi alternativi per la valutazione delle costanti di detto modello allo scopo di renderlo applicabile.

1. Introduzione

La complessità delle operazioni richieste ai moderni sistemi elettronici e principalmente il tempo disponibile per l'effettuazione di tali operazioni, che diviene sempre più breve a causa della velocità dei processi da seguire, impongono sempre più frequentemente la necessità di gestire l'intero sistema tramite un elaboratore.

Tale ulteriore complicazione ha ovviamente una conseguenza anche sulla Affidabilità operativa del sistema, in quanto nel suo calcolo occorrerà tener conto, oltre che della tradizionale affidabilità dell'hardware, dell'affidabilità del software che può definirsi concettualmente come capacità di non avere errori durante una certa missione.

Data la conoscenza assai buona e diffusa della teoria dell'Affidabilità nel caso di hardware, sorge spontanea l'idea di affrontare il problema relativo al software in maniera formalmente identica; appare pertanto naturale definire le caratteristiche di affidabilità del software in termini di successo e insuccesso, cioè in maniera del tutto analoga a quanto fatto nell'hardware. Ad esempio, in questa prospettiva, una caratteristica dell'affidabilità del software è la sua probabilità di successo, definita come la probabilità che il software operi correttamente durante un assegnato tempo di missione quando usato entro i limiti di specifica sulla macchina per la quale è stato progettato.

Se si assume l'ipotesi di indipendenza tra malfunzionamenti di hardware e di software, e se si indicano con R_H (t) ed R_S (t) le probabilità di corretto funzionamento rispettivamente per l'hardware e per il software, si può affermare che la probabilità di corretto funzionamento del sistema è data da:

$$(1) \quad R(t) = R_H(t) \cdot R_S(t)$$

Naturalmente, oltre alla probabilità di successo, altre grandezze possono usarsi per caratterizzare il grado di affidabilità di un sistema, ad esempio, il tempo medio tra errori (MTBE) e la disponibilità (A); tutte quelle grandezze, cioè, le quali sono servite a tale scopo nel caso tradizionale in cui si considerava il solo hardware.

2. Guasto ed errore

Alla luce di quanto precisato nell'introduzione, appare naturale iniziare lo studio dell'affidabilità del software, come già fatto per l'hardware, mettendo a fuoco la causa di malfunzionamento, cioè quello che nel software deve considerarsi come l'analogo del guasto di hardware; appunto dall'esame delle cause di malfunzionamento, infatti, appare la differenza sostanziale tra malfunzionamento di software e malfunzionamento di hardware.

Nel caso dell'hardware è ben noto che la causa di malfunzionamento è il guasto del componente, inteso come variazione delle sue caratteristiche elettriche oltre le tolleranze accettabili, o per slittamento progressivo o in maniera catastrofica (improvvisa e totale). In tal caso è noto che per i singoli componenti può definirsi una funzione di rischio z (t), dalla quale è possibile derivare tutte le caratteristiche di affidabilità.

Nel caso del software, definito come la totalità dei programmi atti a gestire il sistema fisico per la realizzazione delle funzioni previste e la totalità delle descrizioni dei dati da manipolare ed elaborare, appare evidente che non si può parlare di guasto nel senso suddetto, ma più propriamente di errore come causa di malfunzionamento.

Tale distinzione, peraltro, non è soltanto di terminologia, ma implica una fondamentale differenza sia tra le due cause di malfunzionamento sia tra i loro effetti.

L'esame delle cause mostra che il guasto di un componente giuoca un ruolo attivo, nel senso che determinati meccanismi di guasto portano fuori uso il componente inizialmente sano; per quanto riguarda l'errore non si può parlare di un meccanismo fisico che lo generi, in quanto l'evento "errore di software" è il verificarsi della scoperta di un qualcosa che era già contenuta originariamente nel sistema, tale evento è quindi di natura passiva.

Per quanto riguarda gli effetti, dall'esame delle cause appare evidente che il guasto di un componente altera il sistema, poichè esso, una volta verificatosi il guasto, non è più quello di partenza, ma può tornare tale effettuando la sostituzione del componente guasto con uno funzionante; mentre la rilevazione di un errore di software non altera il sistema, che continua ad essere quello iniziale; è piuttosto la rimozione delle cause di errore che conduce ad un sistema diverso da quello di partenza.

Le precisazioni di cui sopra permettono di concludere che, pur apparendo l'errore di software l'analogo del guasto di hardware, occorre elaborare per esso un modello che tenga conto delle differenze sottolineate.

La prima considerazione da fare nell'elaborazione di un modello di errore, riguarda la sua natura deterministica. Un

certo errore di software infatti viene evidenziato ogni qualvolta si presenta una ben precisa combinazione di istruzioni e valori dei dati, è quindi un fatto deterministico ove accada che si presentino le condizioni atte ad evidenziarlo. Poichè ai fini pratici si è interessati all'evento "rilevazione dell'errore", pare opportuno considerare l'insieme dei due eventi "presentarsi della combinazione di cause evidenzianti l'errore" e "verificarsi dell'errore" e notare che mentre il secondo è per sua natura deterministico, il primo, almeno per grossi sistemi elaboratori esplicanti una molteplicità di funzioni su molti dati, è di fatto considerabile come aleatorio; l'evento globale può quindi considerarsi di natura probabilistica.

Tale situazione può chiarirsi con la seguente analogia molto semplice; si consideri un ponte di tavole ed una persona che debba attraversarlo e riferiamo a questo sistema, per analogia, il caso dell'hardware e quello del software.

Nell'analogia con l'hardware il sistema ponte è formato dai componenti tavole ed è sottoposto alla sollecitazione peso della persona. Può accadere che una tavola, la cui capacità di resistenza è diminuita per una qualsiasi ragione, ceda sotto il peso della persona che attraversa, la quale quindi non può portare a termine con successo la "missione" attraversamento del ponte. Da quel momento in poi il ponte è un sistema inefficiente e può essere ripristinato tramite sostituzione della tavola rotta.

Nell'analogia col software la presenza di errori è rappresentata dalla presenza di un certo numero di tavole rotte non discernibili, la cui collocazione è inizialmente ignota, è indubbio che se la persona passa su una di tali tavole, si avrà l'insuccesso della missione e ciò in maniera deterministica, ma, se il ponte è abbastanza largo (software di grosse dimensioni) in modo che non necessariamente la persona che compie l'attraversamento sarà costretta ad interessare le tavole rotte, la missione avrà successo o no a seconda che non siano o siano interessate le tavole rotte e questo secondo evento è di natura probabilistica data l'ipotesi di ignoranza iniziale sulla loro posizione.

3. Modello dell'errore ed affidabilità

Un errore di software può essere definito come quell'evento che rende necessario apportare cambiamenti al codice di programma (riguardanti un solo carattere o molte righe) allo scopo di soddisfare le richieste di specifica del sistema.

La correzione degli errori di software avviene durante la fase di debugging del programma. E' chiaro che, mentre per programmi per i quali è sempre possibile definire con esattezza le grandezze di ingresso e di uscita è anche possibile effettuare prove esaustive che conducono alla rimozione totale degli errori (è ad esempio il caso di programmi di tipo matematico), lo stesso non accade per quei programmi per i quali non sono possibili queste definizioni come è ad esempio il caso di software per il controllo di un sistema in tempo reale; in tali sistemi infatti gli ingressi e le uscite sono dipendenti dal tempo e risulta in tal modo praticamente impossibile prevedere tutte le loro combinazioni.

E' inevitabile, alla luce di quanto detto, che, al momento del rilascio in operazione del sistema, siano presenti un certo numero di errori residui che condurranno al malfunzionamento del sistema ogni qualvolta si verifichino quelle particolari situazioni (non previste nella precedente fase di debugging) atte ad evidenziarli; è pertanto evidente che la frequenza di errore e quindi l'affidabilità del sistema dipende dal numero di tali errori residui, per i quali quindi occorre trovare un modello e la sua connessione con le caratteristiche di affidabilità.

Il punto di partenza per la costruzione del modello deve necessariamente partire dalle uniche informazioni disponibili sul software, cioè dalla registrazione degli errori rivelati e corretti durante la fase di debugging. Tali errori possono essere riportati in un diagramma (Fig. 1) in funzione del tempo di debugging.

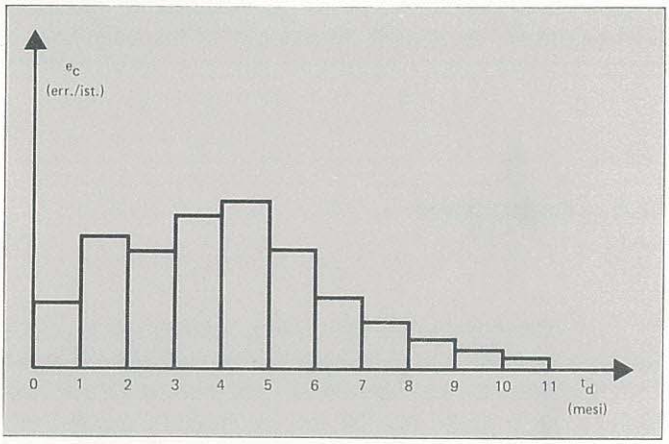


Fig. 1 Errori rilevati e corretti durante la fase di debugging

Si noti che sull'asse delle ordinate di Fig. 1 non compare il numero assoluto di errori corretti, bensì il rapporto tra errori corretti e numero totale di istruzioni. Tale normalizzazione ha lo scopo di permettere la costruzione di un modello valido indipendentemente dalle dimensioni del software (numero di possibili istruzioni).

L'andamento mostrato in Fig. 1, presenta una caratteristica comune ad ogni tipo di software, rappresentata dall'andamento definitivamente decrescente dopo un primo periodo ad andamento qualunque. Se si considera ora la curva relativa alla funzione cumulativa dell'istogramma di Fig. 1, tale caratteristica dell'istogramma si traduce in un andamento asintotico preceduto da un periodo a pendenza variabile (Fig. 2).

Poichè, come detto all'inizio di questo paragrafo, la frequenza di errore, e quindi l'affidabilità del software, è legata al numero di errori presenti nel software all'atto del rilascio in operazione, è opportuno riportare la curva che rappresenta l'andamento degli errori residui in funzione del tempo di debugging, tale curva partirà dal valore cui tende asintoticamente la curva di Fig. 2, valore che si ipotizza costante e che rappresenta il numero totale di errori presenti inizialmente nel programma.

Tale curva è riportata in Fig. 3.

Si noti che ritenere costane il valore asintotico equivale

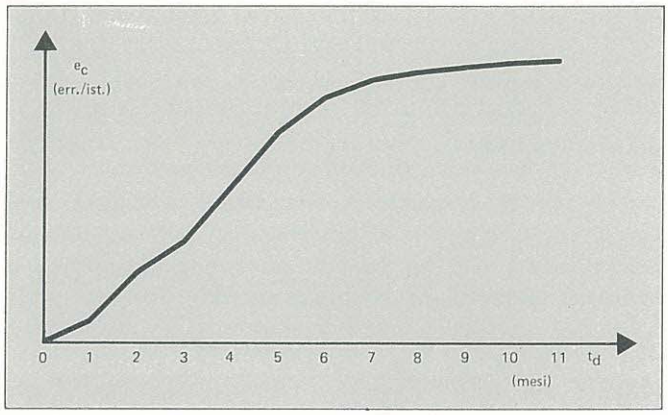


Fig. 2 Andamento cumulativo degli errori corretti

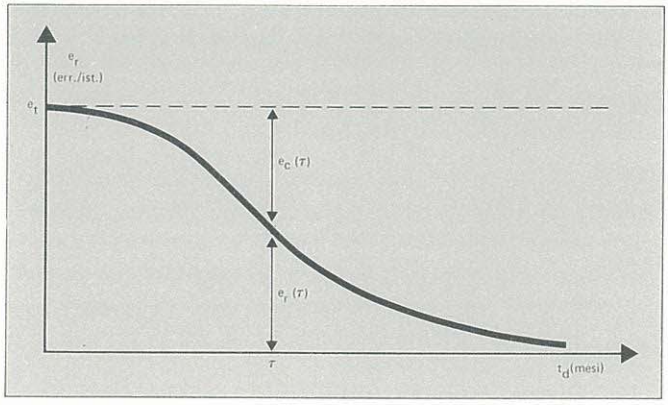


Fig. 3 Andamento degli errori/istruzione residui

ad assumere che il processo di correzione degli errori non influenza negativamente il numero di errori presenti nel software all'istante iniziale.

Se con E si indica il numero totale di errori e con I il numero totale di istruzioni, si ha ovviamente:

$$(2) \quad e_t = \frac{E}{I}$$

Poichè è inevitabile la troncatura del periodo di debugging, è parimenti inevitabile la presenza di un certo numero di errori residui all'atto del rilascio in operazione del sistema, esso sarà ovviamente espresso da:

$$(3) \quad e_r(\tau) = e_t - e_c(\tau) = \frac{E}{I} - e_c(\tau)$$

dove τ è la durata del debugging.

Nell'ipotesi che dal momento del rilascio non si intervenga più per rimuovere eventuali altri errori rilevati, il numero di errori presenti si manterrà costante da quel momento in poi.

Una tale situazione può rappresentarsi graficamente con una curva come quella di Fig. 4.

Questa curva ha andamento simile a quella a "vasca da bagno" dei componenti elettronici, a parte l'ovvia mancanza del periodo di invecchiamento.

Dimensionalmente però l'asse delle ordinate della curva relativa ai componenti elettronici ha il rapporto tra i guasti

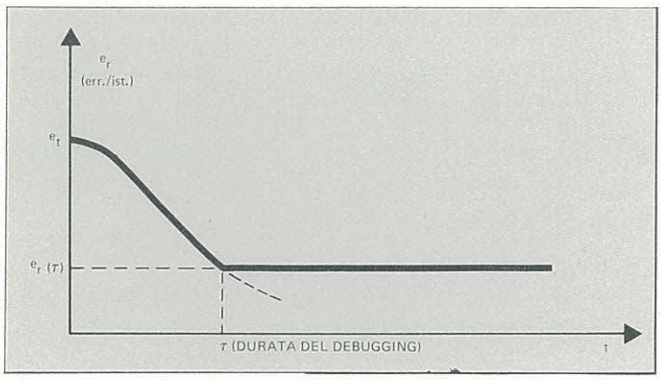


Fig. 4 Andamento degli errori residui con debugging di durata τ

ed il tempo, mentre in Fig. 4 e_r ha le dimensioni di errori/istruzione.

Allo scopo di riportare tali dimensioni a quelle di errori/secondo occorre che il legame tra λ_E ed $e_r(\tau)$ sia di tipo moltiplicativo con costante avente le dimensioni di istruzioni/secondo; indicando con K tale costante si ha:

$$(4) \quad \lambda_E = K e_r(\tau) \qquad \text{(errori/secondo)}$$

Poichè il modello di λ_E ottenuto nel modo descritto è perfettamente analogo a quello sviluppato per i componenti elettronici, la possibilità di non incappare in un errore durante una missione di durata t sarà data da:

$$(5) \quad R_S(t) = \exp(-\lambda_E t)$$

dove:

$\lambda_E = K e_r(\tau)$ = Frequenza di errore

K = Costante che lega la λ_E ad $e_r(\tau)$

$e_r(\tau)$ = Errori/istruzione residui dopo un debugging di durata τ.

Ancora, in maniera analoga all'hardware, il tempo medio tra errori (MTBE) è fornito da:

$$(6) \quad \text{MTBE} = \int_0^\infty R_S(t)\,dt = \frac{1}{\lambda_E} = \frac{1}{K\, e_r(\tau)}$$

La pratica applicazione del modello esposto richiede la stima dei due parametri incogniti K ed $e_r(\tau)$ che compaiono nell'espressione (4), che si può anche scrivere, sostituendo la (3) nella (4):

$$(7) \quad \lambda_E = K\left[\frac{E}{I} - e_c(\tau)\right]$$

dal cui esame si ha che alcune grandezze sono note, mentre per altre occorre applicare un procedimento di stima. Più precisamente le grandezze note sono:

I = Numero totale di istruzioni

$e_r(\tau)$ = Errori corretti con un debugging di durata τ;

mentre le grandezze da stimare sono:

K = costante di proporzionalità tra errori residui e frequenza di errore;

E = numero di errori inizialmene presenti nel software.

4. Stima dei parametri del modello

La stima dei parametri incogniti K ed E che compaiono nella (7), può effettuarsi con uno dei metodi noti; qui di seguito ne verranno applicati due, quello dei momenti e quello della massima verosimiglianza.

4.1. METODO DEI MOMENTI

La stima secondo questo metodo si effettua col seguente procedimento.

Se si effettuano n runs, accade che r di essi saranno coronati da successo (non evidenzieranno cioè errori durante la loro esecuzione), mentre i restanti (n-r) evidenzieranno un errore prima del loro termine.

Se con t_s (s = 1, 2,. . . . . r) si indicano le durate degli r runs coronati da successo e con t_e (e = 1, 2,. . . . . n-r) i tempi trascorsi dall'inizio dei runs evidenzianti errori e l'istante in cui si presenta il primo errore per ciascuno di essi, si ha, ovviamente, che il tempo di corretto funzionamento è dato dalla somma dei t_s e dei t_e, cioè:

$$(8) \quad t_{cf} = \sum_{s=1}^{r} t_s + \sum_{e=1}^{n-r} t_e$$

Poichè i t_e sono calcolati al presentarsi del primo errore, si ha che gli errori che si avranno in totale nel tempo t_{cf} sono proprio (n-r), tanti cioè quanti i runs per i quali si ha insuccesso. Le due grandezze numero di errori e tempo in cui essi si presentano (8) permettono una stima della frequenza di errore data da:

$$(9) \quad \hat{\lambda}_E = \frac{n-r}{t_{cf}}$$

Tale procedimento di stima del λ_E, usato ripetutamente durante il tempo di debugging τ permette il calcolo di E e K nel modo descritto nel seguito.

Nella durata di debugging possono pensarsi fissati tanti istanti t_i tali che, chiamato τ_i l'intervallo 0, t_i si abbia:

$$(10) \quad \tau_1 < \tau_2 < \ldots\ldots < \tau_p = \tau$$

Dopo ciascuno di detti intervalli si effettua una registrazione degli errori rilevati e corretti $e_c(\tau_i)$ (i = 1, 2,. . . p); i valori degli istanti t_i non debbono essere scelti a priori e l'unico vincolo ad essi deve essere che:

$$(11) \quad e_c(\tau_1) < e_c(\tau_2) < \ldots\ldots < e_c(\tau_p) = e_c(\tau)$$

Se in ciascuno di questi istanti si interrompe il processo di debugging e si effettua una stima del $\lambda_E(\tau_i)$ secondo il procedimento che ha condotto alla (9) si ottiene la serie di valori stimati fornita da:

$$(12) \quad \hat{\lambda}_E(\tau_i) = \frac{n_i - r_i}{t_{cfi}} \qquad (i = 1, 2, \ldots\ldots, p)$$

dove:

n_i = Numero di runs di prova dopo il periodo i-esimo

r_i = Numero di tali runs coronati da successo

t_{cfi} = Tempo di corretto funzionamento all'i-esimo periodo valutato secondo la (8).

Se ora si considerano due istanti successivi τ_i e τ_{i+1} (i = 1,2,. . ., p-1) e si eguagliano le relative stime all'espressione (7) di λ_E valutata per $e_c(\tau_i)$ ed $e_c(\tau_{i+1})$, si ottiene il sistema di due equazioni seguente:

$$(13) \quad \begin{aligned} \hat{\lambda}_E(\tau_i) &= K\left[\frac{E}{I} - e_c(\tau_i)\right] \\ \hat{\lambda}_E(\tau_{i+1}) &= K\left[\frac{E}{I} - e_c(\tau_{i+1})\right] \end{aligned}$$

Tale sistema è nelle due incognite K ed E e la sua soluzione permette la loro valutazione.

Facendo il rapporto tra le due equazioni si ha:

$$(13a) \quad \frac{\hat{\lambda}_E(\tau_i)}{\hat{\lambda}_E(\tau_{i+1})} = \frac{E - I\,e_c(\tau_i)}{E - I\,e_c(\tau_{i+1})}$$

quindi:

$$(13b) \quad \begin{aligned} &E\,\hat{\lambda}_E(\tau_i) - \hat{\lambda}_E(\tau_i)\,I\,e_c(\tau_{i+1}) = \\ &= E\,\hat{\lambda}_E(\tau_{i+1}) - \hat{\lambda}_E(\tau_{i+1})\,I\,e_c(\tau_i) \end{aligned}$$

da cui:

$$(13c) \quad \begin{aligned} &E\left[\hat{\lambda}_E(\tau_{i+1}) - \hat{\lambda}_E(\tau_i)\right] = \\ &= I\left[\hat{\lambda}_E(\tau_{i+1})\,e_c(\tau_i) - \hat{\lambda}_E(\tau_i)\,e_c(\tau_{i+1})\right] \end{aligned}$$

in definitiva per la stima di E si ha:

$$(14) \quad \hat{E}_{i+1} = \frac{I\left[\hat{\lambda}_E(\tau_{i+1})\,e_c(\tau_i) - \hat{\lambda}_E(\tau_i)\,e_c(\tau_{i+1})\right]}{\hat{\lambda}_E(\tau_{i+1}) - \hat{\lambda}_E(\tau_i)}$$

La prima delle due equazioni di partenza (13) fornisce:

$$\hat{K}_{i+1} = \frac{\hat{\lambda}_E(\tau_i)}{\frac{\hat{E}_{i+1}}{I} - e_c(\tau_i)}$$

ed effettuando la sostituzione del valore $\hat{E}_{i+1}$ fornito dalla (14) si ottiene:

$$(14a) \quad \begin{aligned} \hat{K}_{i+1} &= \frac{\hat{\lambda}_E(\tau_i)}{\frac{\hat{\lambda}_E(\tau_i)\,e_c(\tau_{i+1}) - \hat{\lambda}_E(\tau_{i+1})\,e_c(\tau_i)}{\hat{\lambda}_E(\tau_i) - \hat{\lambda}_E(\tau_{i+1})} - e_c(\tau_i)} = \\ &= \frac{\hat{\lambda}_E(\tau_i)\left[\hat{\lambda}_E(\tau_i) - \hat{\lambda}_E(\tau_{i+1})\right]}{\hat{\lambda}_E(\tau_i)e_c(\tau_{i+1}) - \hat{\lambda}_E(\tau_{i+1})e_c(\tau_i) - \hat{\lambda}_E(\tau_i)e_c(\tau_i) + \hat{\lambda}_E(\tau_{i+1})e_c(\tau_i)} \end{aligned}$$

ed in definitiva:

$$(15) \quad \hat{K}_{i+1} = \frac{\hat{\lambda}_E(\tau_i) - \hat{\lambda}_E(\tau_{i+1})}{e_c(\tau_{i+1}) - e_c(\tau_i)}$$

Ricordando la definizione di $\hat{\lambda}_E(\tau_i)$ fornita dalla (12) e sostituendo tale espressione nella (14) e (15) si ottengono i valori stimati di E e K in funzione dei dati ottenuti dai

runs di prova, cioè:

$$(16)\quad \hat{E}_{i+1} = \frac{I\left[\frac{n_i - r_i}{t_{cfi}} e_c(\tau_{i+1}) - \frac{n_{i+1} - r_{i+1}}{t_{cf(i+1)}} e_c(\tau_i)\right]}{\frac{n_i - r_i}{t_{cfi}} - \frac{n_{i+1} - r_{i+1}}{t_{cf(i+1)}}}$$

$$(17)\quad \hat{K}_{i+1} = \frac{\frac{n_i - r_i}{t_{cfi}} - \frac{n_{i+1} - r_{i+1}}{t_{cf(i+1)}}}{e_c(\tau_{i+1}) - e_c(\tau_i)}$$

Applicando il descritto procedimento di stima a periodi di debugging terminanti in istanti successivi presi a due a due successivamente, si ottengono due serie di valori per $\hat{E}$ e $\hat{K}$, precisamente:

$$(17a)\quad \begin{matrix} \hat{E}_2, \hat{E}_3, \ldots, \hat{E}_n \\ \hat{K}_2, \hat{K}_3, \ldots, \hat{K}_n \end{matrix}$$

Queste serie di valori possono essere riportate in grafico e, se è vera l'ipotesi di partenza, relativa alla costanza di E e K, essi si disporranno in maniera fluttuante attorno a due linee orizzontali che rappresenteranno per l'appunto le stime migliori di E e K ottenibili col metodo descritto.

4.2. CRITERIO DELLA MASSIMA VEROSIMIGLIANZA

Poichè si è ipotizzato un modello Poissoniano si ha che la funzione densità di probabilità è data da:

$$(18)\quad f(t) = \lambda_E e^{-\lambda_E t}$$

e, tenendo conto della (7),

$$(19)\quad f(t) = K\left[\frac{E}{I} - e_c(\tau)\right] \exp\left\{-K\left[\frac{E}{I} - e_c(\tau)\right] t\right\}$$

Il criterio di stima considerato si applica come segue. Si supponga che dopo debugging di durata $\tau_1, \tau_2, \ldots, \tau_p$, si facciano girare rispettivamente $n_1, n_2, \ldots, n_p$ runs di prova ottenendo, per il generico n_i i tempi al guasto $T_{i1}, T_{i2}, \ldots, T_{i\,n_i}$, tali tempi sono da assumere pari alla durata del run se non si presentano errori, pari all'intervallo tra l'inizio del run ed il presentarsi del primo errore nei casi in cui questo avvenga.

Data la struttura esponenziale della (19) conviene utilizzare la funzione di massima verosimiglianza logaritmica, data da:

$$(20)\quad \mathcal{L}(K, E) = \sum_{i=1}^{n_j} \ln f(T_{j,i}) \qquad (j = 1, 2, \ldots, p)$$

In tal caso, tenendo conto della (19) e dei dati descritti, ottenuti da prove, si ha:

$$\begin{aligned}(21)\quad \mathcal{L}(K, E) = {} & \sum_{i=1}^{n_1}\left\{\ln K + \ln\left[\frac{E}{I} - e_c(\tau_1)\right]\right\} - \\ & - \sum_{i=1}^{n_1} K\left[\frac{E}{I} - e_c(\tau_1)\right] T_{1,i} + \\ & + \sum_{i=1}^{n_2}\left\{\ln K + \ln\left[\frac{E}{I} - e_c(\tau_2)\right]\right\} - \\ & - \sum_{i=1}^{n_2} K\left[\frac{E}{I} - e_c(\tau_2)\right] T_{2,i} + \\ & + \ldots\ldots\ldots\ldots\ldots\ldots + \\ & + \sum_{i=1}^{n_p}\left\{\ln K + \ln\left[\frac{E}{I} - e_c(\tau_p)\right]\right\} - \\ & - \sum_{i=1}^{n_p} K\left[\frac{E}{I} - e_c(\tau_p)\right] T_{p,i} = \\ = {} & (n_1 + n_2 + \ldots + n_p)\ln K + n_1 \ln\left[\frac{E}{I} - e_c(\tau_1)\right] + \\ & + n_2 \ln\left[\frac{E}{I} - e_c(\tau_2)\right] + \ldots\ldots\ldots\ldots + \\ & + n_p \ln\left[\frac{E}{I} - e_c(\tau_p)\right] - K\left[\frac{E}{I} - e_c(\tau_1)\right] \sum_{i=1}^{n_1} T_{1,i} - \\ & - K\left[\frac{E}{I} - e_c(\tau_2)\right] \sum_{i=1}^{n_2} T_{2,i} - \ldots\ldots - \\ & - K\left[\frac{E}{I} - e_c(\tau_p)\right] \sum_{i=1}^{n_p} T_{p,i}\end{aligned}$$

Poichè $\sum_{j=1}^{n_j} T_{j,i}$ è pari al tempo totale di corretto funzionamento durante l'esecuzione degli n_j runs di prova, indicando tale tempo con t_{cfj} si ha in definitiva:

$$\begin{aligned}(22)\quad \mathcal{L}(K, E) = {} & \ln K \sum_{j=1}^{p} n_j + \sum_{j=1}^{p} n_j \ln\left[\frac{E}{I} - e_c(\tau_j)\right] - \\ & - \sum_{j=1}^{p} K\left[\frac{E}{I} - e_c(\tau_j)\right] t_{cfj}\end{aligned}$$

Derivando la (22) rispetto a K ed E ed eguagliando tali derivate a zero, si ottengono le due equazioni di massima verosimiglianza che costituiscono il sistema la cui soluzione fornisce le stime $\hat{K}$ ed $\hat{E}$.

Precisamente:

$$(23)\quad \frac{\partial \mathcal{L}}{\partial K} = \frac{1}{K}\sum_{j=1}^{p} n_j - \sum_{j=1}^{p}\left[\frac{E}{I} - e_c(\tau_j)\right] t_{cfj} = 0$$

da cui si ottiene:

$$(24)\quad K = \frac{\sum_{j=1}^{p} n_j}{\sum_{j=1}^{p}\left[\frac{E}{I} - e_c(\tau_j)\right] t_{cfj}}$$

e

$$(25)\quad \frac{\partial \mathcal{L}}{\partial E} = \sum_{j=1}^{p} \frac{n_j}{I\left[\frac{E}{I} - e_c(\tau_j)\right]} - \frac{K}{I}\sum_{j=1}^{p} t_{cfj} = 0$$

da cui si ottiene:

$$(26)\quad K = \frac{\sum_{j=1}^{p} \frac{n_j}{\left[\frac{E}{I} - e_c(\tau_j)\right]}}{\sum_{j=1}^{p} t_{cfj}}$$

Come detto, le stime $\hat{K}$ ed $\hat{E}$ si ottengono risolvendo il sistema formato dalle equazioni (24) e (26).

5. Conclusione

La trattazione esposta nei precedenti paragrafi mostra che l'affidabilità del software dipende, come è del resto ovvio, dal numero di errori presenti all'atto del suo rilascio in operazione; per migliorare l'affidabilità occorre quindi che tale numero sia quanto minore è possibile.

Come mostrato, il detto numero di errori è dato dalla differenza tra quelli inizialmente presenti e quelli rimossi nella fase di debugging pertanto si otterrà un miglioramento dell'affidabilità minimizzando i primi e massimizzando i secondi.

Il primo è un obiettivo da raggiungere durante la fase di progettazione del software ed infatti sono in corso attività atte ad ottenere metodologie di progettazione del software tali da minimizzare il numero di errori inizialmente presenti. Tra tali nuove tecniche di progettazione le principali sono:

– *Progettazione strutturata:* essa agisce al livello di codifica di istruzioni e consiste nell'utilizzazione di istruzioni di controllo di tipi particolari.

– *Aproccio top-down:* esso agisce di progetto e consiste nello sviluppo di programmi per via gerarchica, cioè utilizzando una struttura ad albero.

Il secondo obiettivo si raggiunge ottimizzando il processo di debugging in modo da massimizzare il tasso di rimozione degli errori; anche in tale direzione sono state elaborate delle tecniche adatte, le cui principali sono:

– *Debugging a settori:* consistente nel provare le singole istruzioni separatamente.

– *Programmi di analisi del flusso:* consistente nel provare le vie di flusso, e quindi le istruzioni di controllo, del programma.

– *Test case generator:* consistente in un'analisi automatica del programma mediante la generazione di particolari input che permettono di provare tutte le vie del programma stesso. Tale sistema è limitato dal fatto che esso non sempre è applicabile per programmi complessi.

BIBLIOGRAFIA

[1] *M.L. Shooman* – Probabilistic Models for Software Reliability. Conference on Statistical Methods for the Evaluation of Computer System Performance, Brown Univ., Nov. 1971 – Published in "Probabilistic Models for Software", Freiberger Ed., Academic Press, N.Y., 1972.

[2] *Z. Jelinski & P. Moranda* – Software Reliability Research. Conference on Statistical Methods for the Evaluation of Computer System Performance, Brown Univ., Nov. 1971 – Published in "Probabilistic Models for Software", Freiberger Ed. Academic Press, N.Y., 1972.

[3] *M.L. Shooman* – Operational Testing and Software Reliability Estimation During Program Development. 1973 IEEE Symposium on Computer Software Reliability – New York city, April 30 through May 2, 1973.

[4] *W.H. Mac Williams* – Reliability of Large Real Time Control Software Systems. 1973 IEEE Symposium on Computer Software Reliability – New York city, April 30 through May 2, 1973.

[5] *R.E. Meeker et Al.* – A Study in Software Reliability and Evaluation – Doc. AD-760 144, February 1973, of the University of Texas, Austin, Texas – Distributed by NTIS – U.S. Department of Commerce.