CAPITOLO 8

Affidabilità del software

1 Introduzione

Il software gioca un ruolo importante nei dispositivi e nei sistemi sia in termini di rilevanza tecnica che in termini di costo di sviluppo. Diversamente dall'hardware, il software non si rompe né si usura, comunque, si possono verificare degli errori in quanto non risponde come dovrebbe alle esigenze per le quali è stato progettato. Gli errori software sono essenzialmente dovuti a difetti che si verificano casualmente nel tempo. Problemi di software sono problemi di qualità che devono essere risolti con strumenti che assicurano la qualità (quality assurance tools) come la prevenzione dei difetti, il testing, o l'utilizzo di strumenti che verificano la qualità dei dati.

Nel definire l'affidabilità del software è necessario tener conto che essa difficilmente si può definire

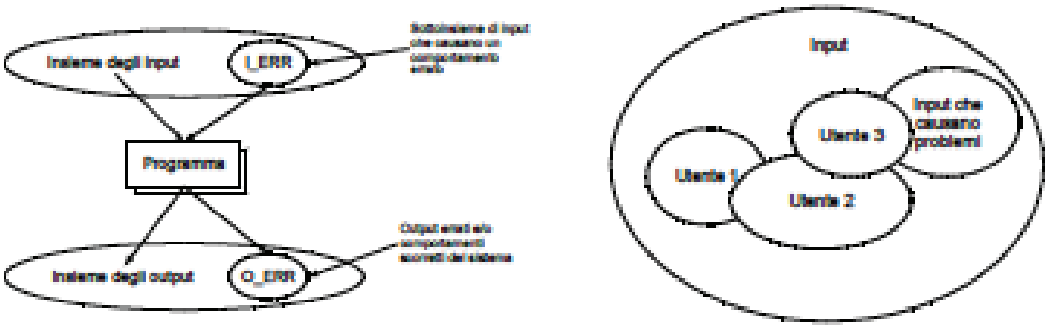

Figure 1: Mappatura input/output e percezione dell'affidabilità.

in modo oggettivo soprattutto perché le misure citate fuori dal contesto non sono significative. In altri termini, è necessario richiedere il profilo di utilizzo del sistema in quanto può essere ragionevole quantificare l'affidabilità di un sistema che controlla sempre lo stesso hardware, mentre non è significativo specificare l'affidabilità di un sistema interattivo che viene usato in modi diversi a meno che non si specificano i diversi modi di utilizzo.

L'affidabilità misura sempre la probabilità che il sistema funzioni senza errori per un dato intervallo di tempo sulla base del sistema, dell'ambiente e in funzione dello scopo. Utenti diversi possono percepire una diversa affidabilità del sistema.
Un fallimento corrisponde ad un comportamento run-time inaspettato (e errato) osservato da un utente del sistema.
Un fault è una caratteristica del software che causa il fallimento.
Osserviamo esplicitamente che i fault non sempre generano fallimenti ma solo quando viene utilizzata la componente errata del sistema.

Attribute	Definition
Compatibility	Degree to which two or more software modules or packages can perform their required functions while sharing the same hardware or software environment
Completeness	Degree to which a software module or package possesses the functions necessary and sufficient to satisfy user needs
Consistency	Degree of uniformity, standardization, and freedom from contradiction within the documentation or parts of a software package
Defect Freedom (Reliability)	Degree to which a software package can execute its required function without causing system failures
Defect Tolerance (Robustness)	Degree to which a software module or package can function correctly in the presence of invalid inputs or highly stressed environmental conditions
Documentation	Totality of documents necessary to describe, design, test, install, and maintain a software package
Efficiency	Degree to which a software module or package performs its required function with minimum consumption of resources (hardware and/or software)
Flexibility	Degree to which a software module or package can be modified for use in applications or environments other than those for which it was designed
Integrity	Degree to which a software package prevents unauthorized access to or modification of computer programs or data
Maintainability	Degree to which a software module or package can be easily modified to correct faults, improve the performance, or other attributes
Portability	Degree to which a software package can be transferred from one hardware or software environment to another
Reusability	Degree to which a software module can be used in another program
Simplicity	Degree to which a software module or package has been conceived and implemented in a straightforward and easily understandable way
Testability	Degree to which a software module or package facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met
Usability	Degree to which a user can learn to operate, prepare inputs for, and interpret outputs of a software package

Software module is used here also for *software element*

Figure 2: Principali attributi della qualità del software.

È necessario inoltre considerare le conseguenze dei fallimenti perché non tutti i fallimenti sono ugualmente gravi.
Un software è percepito come inaffidabile se si verificano fallimenti gravi.
In Figura 1 è schematizzata la corrispondenza tra input e output di un programma e la relativa percezione dell'affidabilità.

L'affidabilità migliora quando si rimuovono i fault che compaiono nelle parti del sistema che sono più frequentemente usate. Tenendo presente che man mano che aumenta l'affidabilità, l'efficienza tende a diminuire si può dedurre che per rendere un sistema più affidabile è importante rimuovere i difetti che causano le conseguenze più serie anche alla luce del fatto che la rimozione dell'$x\%$ di tutti i fault dal software non necessariamente comporta un miglioramento pari all'$x\%$ dell'affidabilità complessiva. Può accadere che la rimozione del 60% dei difetti migliori, ad esempio, l'affidabilità solo del 3%, così che diventa importante rimuovere i difetti che causano le conseguenze più serie. Spesso è necessario inserire codice ridondante che effettui controlli a fissati tempi di esecuzione, comportando eventuali rallentamenti.

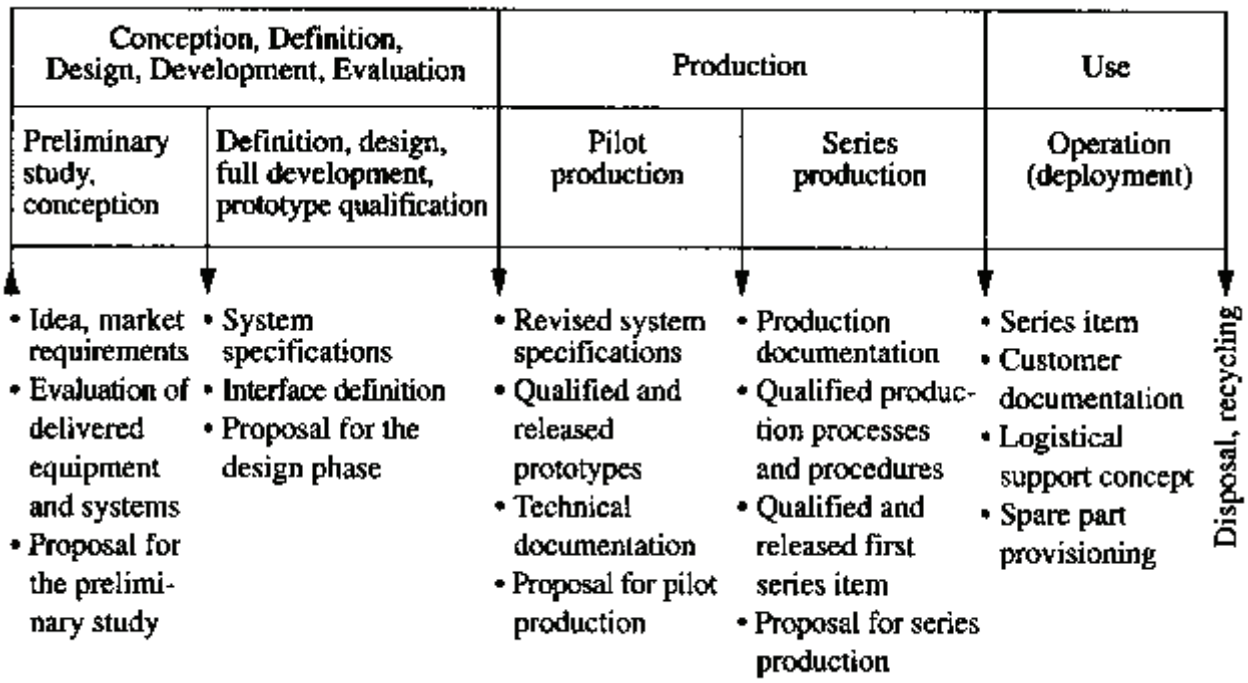

Figure 3: Fasi del ciclo di vita di sistemi complessi.

Per strumenti e sistemi che esibiscono alta affidabilità o che richiedono un certo livello di sicurezza si dovrebbe sviluppare software "tollerante ai difetti" abile cioè a continuare le operazioni anche in presenza di difetti. A questo scopo è necessario fare alcune considerazioni sulla ridondanza "in time domain" (in questa classe rientrano i protocolli di trasmissione e la verifica ciclica della ridondanza) o nel dominio spaziale (ad esempio, codici a correzione di errore, processi paralleli, ...) o una combinazione dei due tipi di ridondanza.

Se nella realizzazione della funzione richiesta, l'interazione tra hardware e software a livello di sistema è alta (embedded software = software radicato) allora la ridondanza dovrebbe coprire i difetti e gli errori hardware creando quindi un sistema tollerante agli errori. In questo contesto ci si dovrebbe concentrare sugli aspetti cause-effetti, riguardanti la criticità dell'hardware, degli errori software dal punto di vista del system level includendo l'hardware, il software, i fattori umani così come il supporto logistico.

Una prima differenza tra l'affidabilità dell'hardware e l'affidabilità del software compare nel ciclo di vita (cf. Tabelle in Figure 2-3). Infatti, in contrasto con l'hardware, nel ciclo di vita del software non compare la fase di produzione visto che il software può essere copiato senza errori.

Una seconda differenza è data dagli attributi di qualità. Non tutti gli attributi della Tabella in Figura 2 possono essere realizzati allo stesso tempo, ma è necessario stabilire una lista di priorità seguita da tutti "gli ingegneri" coinvolti nel progetto. Un'altra difficoltà risiede nella valutazione quantitativa degli attributi della qualità del software nel senso della definizione delle metriche di qualità del software.

Dalle considerazioni fatte possiamo definire la qualità del software come il grado con cui un pacchetto software mantiene una combinazione di attributi di qualità. Ciò consente una valutazione oggettiva del livello di qualità raggiunto. Inoltre, visto che solo un numero limitato di attributi può essere ragionevolmente ben soddisfatto da uno specifico pacchetto software, ne segue che il principale scopo da raggiungere nell'assicurare la qualità del software è massimizzare la parte comune di attributi di qualità necessari, specificati e realizzati, ovviamente se ciò è supportato da un adeguato insieme di metriche della qualità del software.

Le attività specificate devono essere eseguite durante tutte le fasi del ciclo di vita del software. Molte di queste attività devono essere derivate assicurando la qualità dell'hardware, in particolare dalle azioni orientate a prevenire i difetti, configuration management, testing e azioni correttive. Comunque, le attività orientate alla verifica della qualità del software dovrebbero essere più intensive e caratterizzate da feedback più brevi rispetto a ciò che accade all'hardware.

2 Linee guida per la prevenzione dei difetti software

Cominciamo con l'analizzare le principali cause di difetti.

1. Durante la fase di definizione:

   malintesi nel problema di definizione

   vincoli sulle prestazioni della CPU, taglia della memoria, tempo di computazione, funzioni I/O o altro,

   non esatta specificazione dell'interfaccia, poca attenzione alle necessità dell'utente.

2. Durante la fase di progettazione, codifica e test:

   mancanza di accuratezza nella specificazione dei dettagli,

malintesi nella specificazione dei dettagli,

inconsistenza nelle procedure e negli algoritmi,

problemi di sincronizzazione,

errori di conversione dei dati,

struttura complessa del software o alta dipendenza tra i moduli del software.

3. Durante la fase di integrazione, validazione, e istallazione:

   alta interazione tra i moduli del software,

   errori durante la fase di modifica o correzione,

   documentazione non chiara o incompleta,

   cambiamenti nell'ambiente hardware o software, eccedenza di risorse importanti, quali memoria dinamica, disco,...

I difetti sono generalmente causati da errori umani compiuti da chi sviluppa il software o da chi lo usa. La loro rilevazione e rimozione diventa più costosa via via che il ciclo di vita progredisce. Inoltre, considerando che la maggior parte dei difetti può non venire rilevata per un lungo tempo dopo che l'istallazione del software è stata fatta, perché rilevata solo da particolari combinazioni di dati e stati del sistema, si rende necessario prevenire i difetti attraverso un'appropriata assicurazione della qualità del software. In questa direzione possono essere utili alcune linee guida:

1. Fissare regole e procedure scritte e seguirle durante lo sviluppo del software,
2. Dettagliare le specifiche e le interfacce con la maggior cura possibile preferibilmente prima di cominciare la fase di codifica,
3. Dare priorità alla programmazione ad oggetti,
4. Usare adeguatamente linguaggi ad alto livello (l'assembly solo quando il problema non può essere risolto in altro modo),
5. Partizionare il software in moduli indipendenti che dovrebbero essere testati individualmente, sviluppati in modo top down e integrati seguendo una linea botton up,
6. Tenere in considerazione tutti i vincoli forniti dalle funzioni I/O,
7. Sviluppare software in grado di proteggere se stesso,
8. Considerare aspetti di testing/testability più semplici possibili,
9. Migliorare la comprensione del software aggiungendo opportuni commenti,
10. Documentare il software con cura.

3 Linee guida per il software testing

La progettazione del software testing è generalmente un lavoro difficile, infatti anche piccoli programmi possono avere un numero estremamente grande di stati che rende impossibile un test completo. Pertanto, si rende necessaria una strategia di test. Il problema è noto anche per l'hardware.

La principale linea guida che si applica sia al software che all'hardware consiste nel partizionare i termini in moduli indipendenti che possono essere testati indipendentemente e integrati bottom-up per costituire il sistema. Le seguenti linee guida possono essere utili per definire una strategia di test per il software usato in apparecchiature complesse e nei sistemi:

1. programmare i tests presto durante la fase di progettazione e di codifica,
2. utilizzare tools appropriati,
3. eseguire test prima a livello di modulo,
4. integrare e testare successivamente i moduli bottom-up a livello di sistema,
5. testare con cura tutti i cammini sospetti e le parti di software che se non corrette potrebbero causare maggiori guasti al sistema,
6. per tutti i difetti che sono stati individuati introdurre opportune modifiche e sforzi di debugging,
7. testare il software completo sull'hardware finale e con l'ambiente di sviluppo.

Il testing è l'unica possibilità pratica di trovare e quindi eliminare i difetti.

4 Modelli di crescita dell'affidabilità del software

A partire dall'inizio degli anni '70 sono stati proposti molti modelli per descrivere l'occorrenza di difetti nel software. Questi modelli sono usati per studiare, predire, prendere decisioni durante la fase di progettazione del software, per individuare e/o stimare i tempi di rilascio e per ottimizzare l'allocazione delle risorse per sistemi software modulari.

L'occorrenza di difetti software può generare un fallimento a livello di sistema che si presenta in modo spesso casuale nel tempo. Per questo motivo i modelli formulati sono quasi sempre simili a quelli realizzati per i fallimenti hardware. Infatti, anche per il software si introduce il concetto di software failure rate. Questo approccio può essere utile per investigare la crescita della qualità del software durante la validazione e l'istallazione come per i modelli di crescita di affidabilità sviluppati nell'anni '60.

L'obiettivo dovrebbe essere lo sviluppo di software libero da difetti, questo fa sì che ci si focalizzi sulla prevenzione del difetto piuttosto che sul modellare il difetto stesso.

Durante la fase di testing si usano casi di test progettati specificamente per testare il software. L'esecuzione di ogni caso è vista come un "software run" che richiede un certo periodo di tempo e l'uscita di tale software run può essere o un successo o un insuccesso.

Molto spesso si assume che i fallimenti siano indipendenti, ciò significa che un software run è indipendente dall'output di un precedente software run. Questo può essere vero se i casi di test sono scelti dallo spazio degli input con meccanismi casuali così che il software è testato in condizioni omogenee.
Uno scenario più realistico prevede che per aumentare l'affidabilità del test dopo aver osservato un fallimento software si testi una serie di casi correlati che hanno input simili; ciò comporta ovviamente la perdita di indipendenza con gli output successivi.
Pertanto, l'assunzione di indipendenza dei software runs in alcune applicazioni può non essere valida a causa di condizioni operazionali e di vincoli.

Generalmente, nel formulare un modello di software reliability è necessario tener conto di alcuni aspetti quali:

- l'uscita del software run corrente dipende dalle uscite dei software runs precedenti;
- gli errori che causano un fallimento sono rilevati e rimossi istantaneamente non appena si verifica il relativo fallimento;
- la distribuzione del tempo di esecuzione non cambia durante la fase di testing;
- dopo che si è verificato un fallimento, cambiano le probabilità che il prossimo software run fallisca o meno.

Per certificare un software è auspicabile avere una stima degli errori che rimangono nel sistema software, cosa questa difficile da stabilire senza conoscere quali sono gli errori iniziali.

Nell'ambito dell'affidabilità del software, negli ultimi 40-50 anni sono stati proposti molti modelli statistici e sono state sviluppate svariate tecniche per stimare e predire sia l'affidabilità che il numero di errori residui nel software.
Dai dati statistici sugli errori di programmazione è emerso che in media probabilmente ci sono 8 errori per 1000 statments di programma.

Esistono due categorie di modelli di software reliability: modelli deterministici e modelli probabilistici.
Alcuni modelli probabilistici proposti recentemente sono basati sulla teoria delle code, immaginando che il rilevamento di un fallimento corrisponda all'arrivo di un elemento nella coda, questo arrivo lascerà la coda quando verrà corretto l'errore che ha causato il fallimento.

5 Modelli deterministici di crescita dell'affidabilità del software

I modelli deterministici analizzano il contenuto del programma senza introdurre eventi casuali. Esistono due modelli di tipo deterministico particolarmente noti

- la *metrica di Halsted* usata per stimare il numero di errori nel programma,
- la *metrica di McCabe* (complessità ciclomatica) usata per determinare un upper bound al numero rimanente di difetti software.

Generalmente questi modelli rappresentano un approccio quantitativo crescente alla misura del software.

5.1 Metrica del software di Halsted (1977)

In questo caso si usa il numero di operatori e di operandi distinti di un programma per determinare espressioni sulla lunghezza complessiva del programma, sul volume e sul numero di difetti che restano nel programma. In particolare, denotiamo con

n_1 il numero di operatori unici e distinti che compaiono nel programma, gli operatori uguali sono contati una volta sola

n_2 il numero di operandi unici e distinti che compaiono nel programma, gli operandi uguali sono contati una volta sola

N_1 il numero totale di operatori nel programma

N_2 il numero totale di operandi nel programma

N la lunghezza del programma

V il volume del programma

E il numero di errori nel programma

I il numero di istruzioni macchina.

Halsted nel 1977 mostrò che

$$N = N_1 + N_2, \qquad V = N \log_2(n_1 + n_2), \qquad N_i = n_i \log_2 n_i \quad (i = 1, 2)$$

e propose due formule empiriche per stimare il numero di difetti rimanenti nel programma, denotato con E, a partire dal volume, indicato con V:

Formula 1 : $$\widehat{E} = \frac{V}{3000}$$

Formula 2 : $$\widehat{E} = \frac{A}{3000} \qquad \text{dove} \qquad A = \left(\frac{V}{\dfrac{2n_2}{n_1 N_2}} \right)^{2/3} \equiv \left(\frac{V n_1 N_2}{2n_2} \right)^{2/3}.$$

Esempio 1 Consideriamo il programma di ordinamento degli elementi di un array per scambio (in Fortran):

Interchange sort program

Algoritmo InsertionSort(X)	SUBROUTINE SORT (X,N)
Input:Un array X di N elementi confrontabili	DIMENSION X(N)
Output: L'array X avente gli elementi riordinati	IF (N.LT.2) RETURN
for $L = 2$ to N	DO 20 L = 2,N
for $J = 1$ to N	DO 10 J = 1,L
if $(X(L) < X(J)$	IF (X(L).GE.X(J)) GOTO 10
$\{CUR = X(L)$	CUR = X(L)
$X(L) = X(J)$	X(L) = X(J)
$X(J) = CUR\}$	X(J) = CUR
	10 CONTINUE
	20 CONTINUE
	RETURN
	END

Table 1: Ad illustrazione dell'Esempio 1.

In Tabella 2 sono riportati i valori di n_1, n_2, N_1, N_2. In questo caso risulta che

$$N = N_1 + N_2 = 50, \qquad V = N \log_2(n_1 + n_2) = 50 \log_2 17 = 204.$$

Facendo uso del modello 1 si ha

$$\widehat{E} = \frac{V}{3000} = \frac{204}{3000} = 0.068.$$

Operatori	**numero**	**Operandi**	**numero**
=	5	X	6
IF	2	L	5
DO	2	J	4
,	2	N	2
End programma	1	2	2
.LT.	1	SAVE	2
.GE.	1	1	1
GOTO	1		
operazioni su array	6		
End of statement	7		
$n_1 = 10$	$N_1 = 28$	$n_2 = 7$	$N_2 = 22$

Table 2: Ad illustrazione dell'Esempio 1.

Notiamo che il volume per questo programma nella versione in linguaggio assembly dovrebbe essere 328, infatti serve uno sforzo maggiore per specificare un programma in linguaggio assembly. Utilizzando la Formula 2 si ha:

$$\widehat{E} = \frac{A}{3000} = \left(\frac{V}{\frac{2n_2}{n_1 N_2}}\right)^{2/3} \frac{1}{3000} = \left(\frac{204 \cdot 10 \cdot 22}{2 \cdot 7}\right)^{2/3} \frac{1}{3000} = \frac{(3205.714)^{2/3}}{3000} = \frac{217.4118}{3000} = 0.07247.$$

Le metriche di Halsted misurano lo "sforzo" del programma, pertanto, il numero di errori rimanenti deve essere in qualche modo proporzionale a $\widehat{E}$.

5.2 Complessità ciclomatica - Metrica di McCabe (1976)

La metrica di McCabe è una misura di complessità basata sulla rappresentazione del flusso di controllo di un programma e fornisce una misura della complessità logica del programma. Si può applicare a singoli moduli, metodi, classi, funzioni; misura direttamente il numero di cammini linearmente indipendenti nel grafo di controllo del flusso del programma.

Ricordiamo che in un grafo fortemente connesso ogni coppia di nodi è congiunta da un cammino e il numero ciclomatico è il massimo numero di cammini (circuiti) linearmente indipendenti; questi cammini formano una base per tutti i circuiti del grafo in quanto ogni cammino nel grafo si può esprimere come combinazione lineare dei circuiti.
Nel nostro caso i nodi del grafo corrispondono a gruppi indivisibili di istruzioni mentre gli archi (orientati) connettono due nodi, diciamo A e B, se il gruppo di istruzioni rappresentato da B può essere eseguito subito dopo il gruppo rappresentato da A; la notazione usata in tal caso è la seguente: $A \to B$.

Il numero ciclomatico associato al grafo $G(n, e)$ caratterizzato da n nodi e e rami è

$$V(G) = e - n + 2p,$$

dove p rappresenta il numero di componenti connesse e generalmente vale 1.

In alternativa,

$$V(G) = \pi + 1,$$

dove π è il numero di nodi che rappresentano decisioni o ramificazioni[1] nel programma.

McCabe notò che, se usato nel contesto di un cammino base, $V(G)$ fornisce un upper bound al numero di cammini indipendenti nell'insieme di base del programma. Inoltre, $V(G)$ dà anche un limite superiore al numero di test da effettuare per assicurarsi che tutti gli statements del programma sono stati eseguiti almeno una volta.

In quest'ottica, la complessità ciclomatica stima quanto sia difficile testare tutti i cammini del programma fornendo, in tal modo, anche informazioni su quanto sia difficile soddisfare le specifiche richieste.

6 Modelli probabilistici

I modelli probabilistici descrivono l'occorrenza di fallimenti e la rimozione degli errori come eventi probabilistici. Questi modelli sono classificati in diversi gruppi:

- Error seeding
- Failure rate
- Curve fitting

[1] Per ramificazioni si intende un IF, WHILE, REPEAT, CASE, generalmente si esclude il GOTO.

- Reliability growth
- Markov structure
- Time series
- Modelli basati sul processo di Poisson non omogeneo.

6.1 Modelli di error seeding

Questi modelli stimano il numero di errori presenti nel programma utilizzando tecniche di campionamento multistage. L'error seeding è un processo in cui si aggiungono casualmente errori al codice sorgente allo scopo di valutare la quantità di errori residui dopo la fase di test. In questa classe di modelli gli errori sono divisi in due gruppi: *errori indigeni* e *errori indotti*.

Modello di Mill (1970) Assumiamo che gli errori indigeni e gli errori indotti siano ugualmente probabili e denotiamo con

N il numero di errori indigeni

n_1 il numero di errori indotti

r il numero totale di errori rimossi durante il debugging

k il numero di errori indotti rimossi

Osservando che $r-k$ rappresenta il numero totale di errori indigeni rimossi, si può valutare la probabilità di rimuovere r errori di cui k indotti e $r-k$ indigeni:

$$P(k; N, n_1, r) = \frac{\binom{n_1}{k}\binom{N}{r-k}}{\binom{N+n_1}{r}}. \tag{1}$$

In questo modo lo stimatore di massima verosimiglianza di N è

$$\widehat{N} = \lfloor N_0 \rfloor \qquad \text{con} \qquad N_0 = \frac{n_1(r-k)}{k} - 1, \tag{2}$$

se N_0 è intero sia N_0 che $N_0 + 1$ forniscono gli stimatori di massima verosimiglianza.

Esempio 2 Supponiamo che

- sia $n_1 = 10$ il numero di errori indotti

- sia $r = 15$ il numero totale di errori rimossi durante il debugging

- sia $k = 6$ il numero di errori indotti rimossi.

Dalla (2) si ha che

$$N_0 = \frac{n_1(r-k)}{k} - 1 = \frac{10 \cdot (15-6)}{6} - 1 = \frac{90}{6} - 1 = 15 - 1 = 14.$$

Visto che N_0 è intero allora sia 14 che 15 sono gli stimatori di massima verosimiglianza del numero di errori indigeni.

Un altro modo realistico per stimare gli errori che rimangono in un programma è basato sul lavoro di due programmatori che testano il programma su insiemi indipendenti di casi test.

Sia N il numero iniziale di errori presenti nel programma; il primo programmatore rileva n_1 errori e non li rimuove tutti, il secondo programmatore rileva r errori dallo stesso programma.
Supponiamo che k errori siano rilevati da entrambi i programmatori. Se tutti gli errori hanno la stessa probabilità di essere rilevati allora la frazione di errori rilevata dal primo programmatore (o dal secondo), cioè k rispetto ad una selezione casuale di errori r, è pari alla frazione n_1/N.
In altri termini, risulta:

$$\frac{k}{r} = \frac{n_1}{N}$$

da cui si può fornire una stima del numero di errori inizialmente presenti nel programma:

$$\widehat{N} = \frac{n_1 r}{k} \tag{3}$$

La probabilità che esattamente N errori siano inizialmente presenti con k errori comuni e r errori rilevati dal secondo programmatore è ottenibile facendo uso della distribuzione ipergeometrica data in (1)[2], da cui si ricava lo stimatore di massima verosimiglianza per N

$$\widehat{N} = \frac{n_1 r}{k}$$

che coincide con quello fornito in (3).

Esempio 3 Se $n_1 = 10, r = 15$ e $k = 6$ si ha

$$\widehat{N} = \frac{n_1 r}{k} = \frac{10 \cdot 15}{6} = 25.$$

Osserviamo esplicitamente che, rispetto all'Esempio 2 la stima di N ci ha fornito la somma $n_1 + r$.

Modello di Cai (1998) Questo modello è utilizzato per stimare il numero di difetti che restano nel software. Supponiamo che il software sia diviso in *Parte 0* e *Parte 1*. Inoltre, assumiamo che

1. Nel software rimangono N difetti di cui N_0 nella *Parte 0* e N_1 nella *Parte 1*, naturalmente $N_0 + N_1 = N$
2. Ogni difetto restante ha la stessa probabilità di essere rilevato
3. Dove rilevati i difetti sono rimossi

[2]Relativamente al secondo programmatore risulta $\frac{\binom{N-k}{r-k}\binom{k}{k}}{\binom{N}{r}}$

4. Si rimuove un difetto alla volta e ogni rimozione non introduce altri difetti

5. Restano n difetti da rimuovere.

Sia t_i l'istante in cui si rimuove il difetto i-esimo e sia Y_i la seguente funzione indicatrice:

$$Y_i = \begin{cases} 0 & \text{se il difetto } i\text{-esimo è nella Parte 0} \\ 1 & \text{se il difetto } i\text{-esimo è nella Parte 1.} \end{cases}$$

Denotiamo con $N_j(i)$ il numero di difetti che restano nella *Parte j* nell'intervallo di tempo (t_i, t_{i+1}) e assumiamo che $Y_0 = y_0 = 0$ e che $Y_i = y_i$ che rappresenta il valore osservato. Risulta che

$$N_0(i) = N_0 - i + \sum_{j=0}^{i} y_j, \qquad N_1(i) = N_1 - \sum_{j=0}^{i} y_j.$$

Pertanto, denotando con

$$p_j(i) = \mathbb{P}(rilevare\ un\ difetto\ in\ (t_i, t_{i+1})\ nella\ parte\ P_j) = \frac{N_j(i)}{N_0(i) + N_1(i)},$$

si ha:

$$p_0(i) = \frac{N_0 - i + \sum_{j=0}^{i} y_j}{N_0 + N_1 - i}, \qquad p_1(i) = \frac{N_1 - \sum_{j=0}^{i} y_j}{N_0 + N_1 - i}.$$

Osservazione 1 Notiamo che $p_0(i)$ e $p_1(i)$ definiscono la funzione di verosimiglianza.

Dimostrazione Sia $L(y_1, y_2, \ldots, y_n)$ la funzione di verosimiglianza. Risulta che

$$\begin{aligned} L(y_1, y_2, \ldots, y_n) &\equiv \mathbb{P}(Y_1 = y_1, Y_2 = y_2, \ldots, Y_n = y_n) \\ &= \prod_{i=1}^{n} \mathbb{P}[Y_i = y_i | Y_1 = y_1, Y_2 = y_2, \ldots, Y_{i-1} = y_{i-1}], \end{aligned}$$

dove

$$\mathbb{P}[Y_i = y_i | Y_1 = y_1, Y_2 = y_2, \ldots, Y_{i-1} = y_{i-1}] = \begin{cases} p_0(i-1) & y_i = 0 \\ p_1(i-1) & y_i = 1. \end{cases}$$

Quindi,

$$L(y_1, y_2, \ldots, y_n) = \prod_{i=1}^{n} [p_0(i-1)]^{1-y_i} [p_1(i-1)]^{y_i}.$$

Pertanto, la funzione di verosimiglianza è definita in termini di $p_0(i)$ e $p_1(i)$. Inoltre risulta

$$\ln L(y_1, y_2, \ldots, y_n) = \sum_{i=1}^{n} (1 - y_i) \ln[p_0(i-1)] + \sum_{i=1}^{n} y_i \ln[p_1(i-1)].$$

Dalla funzione di verosimiglianza si possono determinare gli stimatori di massima verosimiglianza di N_0 e N_1 risolvendo le equazioni

$$\sum_{i=1}^{n} \frac{1 - y_i}{N_0 - i + 1 + \sum_{j=0}^{i-1} y_i} = \frac{1}{N_0 + N_1 - i + 1}, \qquad \sum_{i=1}^{n} \frac{y_i}{N_1 - \sum_{j=0}^{i-1} y_i} = \frac{1}{N_0 + N_1 - i + 1}.$$

Modello di distribuzione ipergeometrico (Tohma *et al.* 1991) Questo modello stima il numero di faults inizialmente presenti nel programma all'inizio della fase di test o del processo di debugging facendo uso della distribuzione ipergeometrica.
Indichiamo con C_{i-1} il numero complessivo di errori già rilevati durante le istanze di test $t_1, t_2, \ldots, t_{i-1}$ e sia N_i il numero di errori rilevati durante l'istanza t_i. Assumiamo che

1. All'inizio della fase di test il programma contiene m faults
2. Un test è definito come un numero di istanze di test costituite da dati input e dati output. In altri termini, un'istanza del test (test instance) è l'insieme di operazioni di test eseguite in un giorno, una settimana,... Le test instance sono denotate con t_i per $i = 1, 2, \ldots, n$
3. Tra un'istanza e l'altra del test non sono rimossi i fault rilevati.

Sia C_{i-1} il numero di faults contati fino all'istanza t_{i-1}, W_i il numero di fault evidenziati fino all'istanza t_i e sia $N_i = n_i$ il numero di faults nuovi rilevati durante l'istanza osservata di t_i, risulta che $n_i \leq W_i$. Infatti, alcuni dei W_i fault potrebbero essere stati già contati in C_{i-1}. Ogni fault può essere classificato in "nuovo" o "riscontrato nuovamente". Se il numero di fault nuovi N_i segue la distribuzione ipergeometrica si ha che

$$\mathbb{P}(N_i = n_i) = \frac{\binom{m - C_{i-1}}{n_i}\binom{C_{i-1}}{W_i - n_i}}{\binom{m}{W_i}},$$

dove m è il numero di falts inizialmente presenti nel programma ed inoltre

$$C_{i-1} = \sum_{k=1}^{i-1} n_k, \qquad C_0 = 0, \qquad n_0 = 0.$$

Affinché la precedente relazione sia consistente deve risultare che

$$0 \leq n_i \leq m - C_{i-1} \quad \text{e} \quad 0 \leq W_i - n_i \leq C_{i-1},$$

ossia

$$W_i - C_{i-1} \leq n_i \leq W_i.$$

Pertanto, deve risultare

$$\max(0, W_i - C_{i-1}) \leq n_i \leq \min(W_i, m - C_{i-1}).$$

Inoltre,

$$\mathbb{E}(N_i) = \frac{(m - C_{i-1})W_i}{m}, \qquad \mathbb{E}(C_i) = m\left[1 - \prod_{j=1}^{i}\left(1 - \frac{W_j}{n_j}\right)\right].$$

6.2 Modelli di failure rate

Questo gruppo di modelli è usato per studiare il tasso di fallimento per fault e gli intervalli di tempo tra un fallimento e l'altro. In particolare, questi modelli permettono di analizzare come il failure rate varia nel tempo in base al numero di faults rimanenti. In questo gruppo sono inclusi i seguenti modelli

- Modello di Jelinski-Moranda (1972)
- Modello di Schick-Wolverton (1978)
- Modello di Schick-Wolverton modificato (1977)
- Modello Goel-Okumoto Imperfect Debugging (1979)
- Modello di Jelinski Moranda (1979)
- Modello di Moranda (1975)
- Modello di Poisson con distribuzione binomiale negativa

Modello di Jelinski-Moranda (1972) Denotiamo con

- Φ una costante di proporzionalità,

- N il numero iniziale di faults,

- t_i l'intervallo di tempo tra l'$(i-1)$-esimo e lo i-esimo fallimento.

Il failure rate è definito al seguente modo:

$$\lambda(t_i) = \Phi[N - (i-1)].$$

Pertanto, inizialmente risulta:

$$\lambda(t_1) = \Phi N, \quad \lambda(t_2) = \Phi(N-1), \ \ldots$$

Inoltre, la funzione densità di probabilità, la funzione di distribuzione e l'affidabilità in uno specifico intervallo di tempo risultano essere:

$$\begin{aligned}
f(t_i) &= \lambda(t_i) \exp\left\{-\int_0^{t_i} \lambda(x)\, dx\right\} = \Phi[N-(i-1)] \exp\left\{-\Phi[N-(i-1)]\, t_i\right\}, \\
F(t_i) &= 1 - \exp\left\{-\Phi[N-(i-1)]\, t_i\right\}, \\
R(t_i) &= 1 - F(t_i) = \exp\left\{-\Phi[N-(i-1)]\, t_i\right\}.
\end{aligned}$$

Queste funzioni risultano costanti nell'intervallo di tempo tra un fallimento e l'altro.
Il metodo di massima verosimiglianza può essere utilizzato per stimare N e Φ. In particolare, la funzione di verosimiglianza è

$$L(N, \Phi) = \Phi^n \prod_{i=1}^{n} [N-(i-1)] \exp\left\{-\Phi \sum_{i=1}^{n} [N-(i-1)]\, t_i\right\},$$

da cui segue che

$$\ln L(N,\Phi) = n\ln\Phi + \sum_{i=1}^{n}\ln[N-(i-1)] - \Phi\sum_{i=1}^{n}[N-(i-1)]t_i.$$

Pertanto,

$$\frac{\partial}{\partial N}\ln L(N,\Phi) = \sum_{i=1}^{n}\frac{1}{N-(i-1)} - \Phi\sum_{i=1}^{n}t_i = 0$$
$$\frac{\partial}{\partial \Phi}\ln L(N,\Phi) = \frac{n}{\Phi} - \sum_{i=1}^{n}[N-(i-1)]\,t_i = 0$$

da cui risulta che

$$\Phi = \frac{\displaystyle\sum_{i=1}^{n}\frac{1}{N-(i-1)}}{\displaystyle\sum_{i=1}^{n}t_i}, \qquad n\sum_{i=1}^{n}t_i = \left[\sum_{i=1}^{n}[N-(i-1)t_i]\right]\left[\sum_{i=1}^{n}\frac{1}{N-(i-1)}\right]$$

Modello di Schick-Wolverton (1978) In questo caso il failure rate è definito al seguente modo:

$$\lambda(t_i) = \Phi[N-(i-1)]\,t_i,$$

così che la funzione densità di probabilità, la funzione di distribuzione e l'affidabilità in uno specifico intervallo di tempo risultano essere:

$$f(t_i) = \lambda(t_i)\exp\left\{-\int_0^{t_i}\lambda(x)\,dx\right\} = \Phi[N-(i-1)]\,t_i\exp\left\{-\frac{\Phi(N-i+1)\,t_i^2}{2}\right\},$$
$$F(t_i) = 1-\exp\left\{-\frac{\Phi(N-i+1)\,t_i^2}{2}\right\},$$
$$R(t_i) = 1-F(t_i) = \exp\left\{-\frac{\Phi(N-i+1)\,t_i^2}{2}\right\}.$$

Inoltre, la funzione di verosimiglianza è

$$L(N,\Phi) = \Phi^n\prod_{i=1}^{n}[N-(i-1)]\exp\left\{-\Phi\sum_{i=1}^{n}(N-(i-1))\,t_i\right\},$$

da cui segue che

$$\ln L(N,\Phi) = n\ln\Phi + \sum_{i=1}^{n}\ln[N-(i-1)]i + \sum_{i=1}^{n}\ln t_i - \Phi\sum_{i=1}^{n}[N-(i-1)]\frac{t_i^2}{2}.$$

Pertanto,

$$\frac{\partial}{\partial N}\ln L(N,\Phi) = \sum_{i=1}^{n}\frac{1}{N-(i-1)} - \Phi\sum_{i=1}^{n}\frac{t_i^2}{2} = 0$$
$$\frac{\partial}{\partial \Phi}\ln L(N,\Phi) = \frac{n}{\Phi} - \sum_{i=1}^{n}[N-(i-1)\,\frac{t_i^2}{2} = 0$$

da cui risulta che

$$\Phi = 2\sum_{i=1}^{n} \frac{1}{[N-(i-1)]\,T}, \qquad N = \frac{2n}{\Phi\,T} + \frac{1}{T}\sum_{i=1}^{n}(i-1)t_i^2,$$

dove

$$T = \sum_{i=1}^{n} t_i^2.$$

Modello di Schick-Wolverton modificato (1977) In questo caso si suppone che durante la fase $(i-1)$-esima può essere rilevato più di un errore. Specificamente, denotando con n_{i-1} il numero di errori che si sono verificati nell'intervallo di tempo tra la $(i-1)$-esima e la i-esima fase di test, il failure rate è definito al seguente modo:

$$\lambda(t_i) = \Phi[N - n_{i-1}]\,t_i,$$

così che la funzione densità di probabilità, la funzione di distribuzione e l'affidabilità in uno specifico intervallo di tempo risultano essere:

$$\begin{aligned}
f(t_i) &= \lambda(t_i)\exp\left\{-\int_0^{t_i}\lambda(x)\,dx\right\} = \Phi[N-n_{i-1}]\,t_i\exp\left\{-\frac{\Phi(N-n_{i-1})\,t_i^2}{2}\right\},\\
F(t_i) &= 1-\exp\left\{-\frac{\Phi(N-n_{i-i})\,t_i^2}{2}\right\},\\
R(t_i) &= 1-F(t_i) = \exp\left\{-\frac{\Phi(N-n_{i-1})\,t_i^2}{2}\right\}.
\end{aligned}$$

Goel-Okumoto Imperfect Debugging model (1979) In questo caso si assume che una volta rilevato un errore il fault che lo ha causato possa essere rimosso con probabilità p; ovviamente $1-p$ rappresenta la probabilità che il fault non venga rimosso. Tenendo conto di ciò il failure rate risulta:

$$\lambda(t_i) = \Phi[N-(i-1)\,p].$$

così che la funzione densità di probabilità, la funzione di distribuzione e l'affidabilità in uno specifico intervallo di tempo risultano essere:

$$\begin{aligned}
f(t_i) &= \lambda(t_i)\exp\left\{-\int_0^{t_i}\lambda(x)\,dx\right\} = \Phi[N-(i-1)]p\exp\left\{-\Phi[N-(i-1)\,p]\,t_i\right\}\\
F(t_i) &= 1-\exp\left\{-\Phi[N-(i-1)\,p]\,t_i\right\}\\
R(t_i) &= 1-F(t_i) = \exp\left\{-\Phi[N-(i-1)\,p]\,t_i\right\}.
\end{aligned}$$

Notiamo esplicitamente che

$$\lambda(t_i) = \Phi[N-(i-1)\,p] = p\,\Phi\left[\frac{N}{p}-(i-1)\right] \equiv \widetilde{\Phi}[\widetilde{N}-(i-1)]$$

pertanto, il failure rate di questo modello è della stessa forma del failure rate del modello di Jelinski-Moranda con $\widetilde{\Phi} = p\,\Phi$ e $\widetilde{N} = N/p > N$.

Modello geometrico di Jelinski-Moranda (1979) In questo caso si assume che il failure rate inizialmente sia costante e decresce "geometricamente" al crescere dei tempi di fallimento. In particolare, durante lo i-esimo intervallo di fallimento si assume che

$$\lambda(t_i) = D\,K^{i-1},$$

dove D è il tasso iniziale di fallimento e $K \in (0,1)$ è il parametro della funzione geometrica. La funzione densità di probabilità, la funzione di distribuzione e l'affidabilità in uno specifico intervallo di tempo risultano essere:

$$\begin{aligned}
f(t_i) &= \lambda(t_i)\exp\left\{-\int_0^{t_i}\lambda(x)\,dx\right\} = D\,K^{i-1}\exp\left\{-D\,K^{i-1}t_i\right\},\\
F(t_i) &= 1-\exp\left\{-D\,K^{i-1}t_i\right\},\\
R(t_i) &= 1-F(t_i) = \exp\left\{-D\,K^{i-1}t_i\right\}.
\end{aligned}$$

Se si suppone che durante una singola fase si possono rimuovere più errori allora il failure rate diventa

$$\lambda(t_i) = D\,K^{n_{i-1}},$$

dove n_{i-1} rappresenta il numero totale di errori rimossi fino alla fase di test $(i-1)$-esima. In questo caso risulta:

$$\begin{aligned}
f(t_i) &= \lambda(t_i)\exp\left\{-\int_0^{t_i}\lambda(x)\,dx\right\} = D\,K^{n_{i-1}}\exp\left\{-D\,K^{n_{i-1}}t_i\right\},\\
F(t_i) &= 1-\exp\left\{-D\,K^{n_{i-1}}t_i\right\},\\
R(t_i) &= 1-F(t_i) = \exp\left\{-D\,K^{n_{i-1}}t_i\right\}.
\end{aligned}$$

Modello geometrico di Moranda (di Poisson) (1975) In questo modello si assume che la durata di una generica fase di test sia costante e pari a T non dipendendo da i. Sia N_i il numero di fallimenti che si sono verificati durante la i-esima fase di test; N_i segue la distribuzione di Poisson con parametro $D\,K^{i-1}$, in altri termini risulta che

$$\mathbb{P}(N_i = n) = e^{-D\,K^{i-1}}\frac{(D\,K^{i-1})^n}{n!}.$$

Pertanto, ricordando la relazione esistente tra la distribuzione di Poisson e la distribuzione esponenziale segue che la funzione densità di probabilità, la funzione di distribuzione e l'affidabilità in uno specifico intervallo di tempo sono:

$$\begin{aligned}
f(t_i) &= \lambda(t_i)\exp\left\{-\int_0^{t_i}\lambda(x)\,dx\right\} = D\,K^{i-1}\exp\left\{-D\,K^{i-1}T\right\},\\
F(t_i) &= 1-\exp\left\{-D\,K^{i-1}T\right\},\\
R(T) &= 1-F(T) = \exp\left\{-D\,K^{i-1}T\right\}.
\end{aligned}$$

Modello di Poisson con distribuzione binomiale negativa Anche in questo modello, come nel precedente (Modello geometrico di Moranda) si suppone che il numero di fallimenti che si sono

verificati durante la i-esima fase di test segue la distribuzione di Poisson, ma il parametro di tale distribuzione in questo caso è una variabile aleatoria che si assume di distribuzione gamma, o meglio di Erlang:

$$f(\lambda) = \frac{1}{\Gamma(m)} k^m \lambda^{m-1} e^{-k\lambda} \qquad \lambda > 0.$$

Per calcolare la probabilità che ci siano n errori in $(0, t)$ osserviamo che

$$\begin{aligned}\mathbb{P}[N(t) = n] &= \int_0^\infty \mathbb{P}[N(t) = n | \Lambda = \lambda]\, d\lambda = \int_0^\infty \frac{(\lambda t)^n}{n!}\, e^{-\lambda t}\, \frac{1}{\Gamma(m)} k^m \lambda^{m-1} e^{-k\lambda}\, d\lambda \\ &= \frac{k^m t^n}{\Gamma(m)}\, \frac{1}{n!}\, \frac{\Gamma(n+m)}{(k+t)^{n+m}} = \frac{k^m t^n}{n!}\, \frac{(n+m-1)\ldots m}{(k+t)^{n+m}} = \binom{n+m-1}{n} \frac{k^m t^n}{(k+t)^{n+m}}\end{aligned}$$

Pertanto, ponendo

$$p = \frac{k}{t+k}, \qquad 1 - p = q = \frac{t}{t+k},$$

segue che

$$\mathbb{P}[N(t) = n] = \binom{n+m-1}{n} p^m (1-p)^n \qquad (n = 0, 1, \ldots),$$

È possibile quindi concludere che la probabilità che ci siano n errori in $(0, t)$ segue la distribuzione binomiale negativa di parametri m e p.

6.3 Curve fitting model

Questi modelli usano l'analisi di regressione statistica per studiare le relazioni tra le complessità del software e il numero di faults presenti nel programma. Il problema consiste nell'individuare le variabili dipendenti e le variabili indipendenti nonché le relazioni funzionali che legano tali variabili usando i classici metodi statistici di regressione lineare, regressione non lineare, analisi di serie temporali. Generalmente le variabili dipendenti rappresentano il numero di errori nel programma; le variabili indipendenti descrivono invece il numero di moduli cambiati durante la fase di manutenzione, ossia durante gli intervalli di tempo tra un fallimento e l'altro. Questa classe comprende

Modelli che stimano gli errori,

Modelli di stima della complessità,

Modelli che stimano il failure rate.

Modelli di stima degli errori Sia N il numero di errori iniziali. Denotiamo con X_i il fattore di errore i-esimo, generalmente si tratta di una metrica di complessità del software o di un fattore ambientale. Si assume che

$$N = \sum_i a_i X_i + \sum_i b_i X_i^2 + \sum_i c_i X_i^3 + \epsilon,$$

dove a_i, b_i, c_i sono i coefficienti del modello e ϵ è un termine di errore. Molti modelli considerano un unico fattore di errore.

Modello di stima della complessità (Belady e Lehman (1976) Questo modello è generalmente utilizzato per software caratterizzati dall'avere molte versioni rilasciate. Siano

- R il numero della sequenza rilasciata,

- E_R il/i fattore/i ambientali che intervengono durante la fase R,

- M_R il numero di moduli al rilascio R,

- I_R l'ampiezza dell'intervallo di tempo tra un rilascio e l'altro,

- D il numero di giorni trascorsi dal primo errore individuato,

- ϵ un termine di errore.

La complessità del software è definita come

$$C_R = a_0 + a_1 R + a_2 E_R + a_3 M_R + a_4 I_R + a_5 D + \epsilon.$$

Modello di stima del failure rate (Miller 1985) Siano $t_1, t_2, \ldots, t_n$ gli istanti in cui si verificano i fallimenti. Una stima del tasso di fallimento durante l'intervallo i-esimo è

$$\widehat{\lambda_i} = \frac{1}{t_{i+1} - t_i}.$$

Se i failure rates sono non-decrescenti allora una stima di λ_i, indicata con λ_i^*, può essere ottenuta col metodo dei minimi quadrati (Miller 1985).

6.4 Modelli markoviani

Un processo markoviano è caratterizzato dal fatto che il comportamento futuro del processo, noto lo stato presente, non dipende dalla storia passata.

Il gruppo di modelli basati sulla proprietà di Markov è usato per studiare l'affidabilità del software e le relazioni esistenti tra i vari moduli. In particolare, si assume che i fallimenti dei moduli sono indipendenti fra loro: tale assunzione è abbastanza adeguata quando si ragiona a livello di moduli perchè questi possono essere progettati, codificati e testati indipendentemente l'uno dall'altro; tuttavia, tale assunzione può non essere vera quando si ragiona a livello di intero sistema.
Questa classe include:

Modello di Markov con debugging imperfetto (Goel 1979b),

Modello Markoviano di Littlewood (1979),

Modello Software safety (Yamada *et al.* 1998).

Modello di Markov con debugging imperfetto (Goel 1979b) In questo caso, le transizioni dallo stato i allo stato j sono regolate dalle seguenti probabilità

$$p_{i,j} = \begin{cases} p & j = i-1 \\ q & j = i \\ 1 & j = i = 0 \\ 0 & altrimenti, \end{cases}$$

dove p denota la probabilità di debugging correttamente eseguito e $q = 1 - p$ è la probabilità di debugging imperfetto.
In questo caso l'affidabilità nel k-esimo intervallo di fallimento è

$$R_k(t) = \sum_{j=0}^{k-1} \binom{k-1}{j} p^{k-j-1} q^j e^{-[N-(k-j-1)]\,\Phi t}.$$

Modello Markoviano di Littlewood (1979) In questo modello si considerano due tipi di fallimenti. Un primo tipo si presenta ad ogni modulo in accordo ad un processo di Poisson ed è caratterizzato dal fatto che ogni integrazione dei moduli comporta l'introduzione di nuovi errori. Il secondo tipo di fallimenti si verifica a livello di interfaccia dei moduli. Denotiamo con

- n il numero di moduli
- $a_{i,j}$ le transizioni tra il modulo i e il modulo j
- λ_i il tasso del processo di Poisson che descrive i fallimenti al modulo i
- $q_{i,j}$ la probabilità che la transizione tra il modulo i e il modulo j fallisca
- π_j la distribuzione limite del processo.

Littlewood mostrò che il processo di fallimento del programma tende asintoticamente ad un processo di Poisson con tasso di fallimento

$$\sum_{i=1}^{n} \pi_i \left(\lambda_i + \sum_{j \neq i}^{n} a_{i,j} q_{i,j} \right)$$

quando λ_i e $q_{i,j}$ tendono a zero.

Modello Software safety (Yamada *et al.* 1998) Questo modello descrive il comportamento time-dependent del software basandosi su processi di Markov. Devono essere soddisfatte le seguenti assunzioni:

- Il software può operare in sicurezza o no (a rischio) durante il suo utilizzo. Comunque, gli intervalli di tempo in cui opera il software sono distribuiti esponenzialmente con media $1/\vartheta$ per gli intervalli in cui opera in sicurezza e $1/\eta$ per gli altri
- Quando si verifica un fallimento viene eseguita un'attività di debugging. Il debugging è perfetto con probabilità a e risulta imperfetto con probabilità b

- L'affidabilità del software aumenta nel caso di debugging perfetto. Gli intervalli di tempo che intercorrono tra l'occorrenza di un fallimento e il successivo seguono una distribuzione esponenziale di media $1/\lambda_n$ per $n = 0, 1, \ldots$ dove λ_n denota il numero totale di faults corretti
- La probabilità che due o più fallimeti si verificano simultaneamente risulta trascurabile.

Sia $X(t)$ lo stato del sistema software all'istante t. $X(t)$ è un processo stocastico che assume due valori a seconda se si sta lavorando in sicurezza (W_n) o no. quindi,

$$X(t) = \begin{cases} W_n & \text{se il sistema lavora in sicurezza} \\ U_n & \text{se il sistema non lavora in sicurezza.} \end{cases}$$

Supponiamo che sono stati già corretti n faults, l'intensità di fallimento per l'occorrenza del prossimo faults è

$$\lambda_n = D\, k^n,$$

dove $D > 0$ rappresenta il tasso di fallimento iniziale e $k \in (0, 1)$ è un parametro descrivente la decrescita del tasso di fallimento. La sicurezza del software è definita come la probabilità con cui il sistema non fallisce all'istante t:

$$S(t) \equiv \mathbb{P}(\textit{il sistema non fallisce all'istante } t) = \sum_{n=0}^{\infty} p_n(t),$$

dove

$$p_n(t) = \mathbb{P}[X(t) = W_n] = A_n e^{-(\lambda_n + \vartheta + \eta)t} + \sum_{i=0}^{n} B_{n,i} e^{-a\lambda_i t}$$

con

$$A_n = \frac{-\vartheta \prod_{j=0}^{n-1} a\lambda_j}{\prod_{j=0}^{n} (a\lambda_j - \lambda_n - \vartheta - \eta)}, \qquad B_{n,i} = \frac{(\lambda_n + \eta - a\lambda_i) \prod_{j=0}^{n-1} \lambda_j}{\lambda_n + \eta - a\lambda_i) \prod_{j=0}^{n} (\lambda_j - \lambda_i)}.$$

6.5 Modelli basati sulle serie storiche

Consideriamo il modello ARIMA (Autoregressive Integrated Moving Average) studiato da Box e Jenkins (1994). Denotiamo con

- z_t il numero di fallimenti osservati nella t-esima fase di test, ossia il valore osservato all'istante t,
- δ la media dei valori z_t per $t = 0, 1, \ldots$,
- $a_t = z_t - \widehat{z}_t$ un valore generato da una distribuzione normale di media nulla; questo valore rappresenta un *rumore bianco* durante l'intervallo t-esimo,
- p il numero di ordine dell'autoregressione (AR),

- d la differenza con i dati originali,

- q il numero di ordine della moving average (MA).

Il modello ARIMA esprime il numero di fallimenti osservati nella i-esima fase di test al seguente modo:

$$z_i = \delta + \varphi_1 z_{i-1} + \varphi_2 z_{i-2} + \ldots + \varphi_p z_{i-p} + a_i - \vartheta_1 a_{i-1} - \vartheta_2 a_{i-2} - \ldots - \vartheta_q a_{i-q}.$$

Siano inoltre $y_1, y_2, \ldots, y_n$ numeri interi ciascuno rappresentante il numero di fallimenti (cumulativi) del software registrati alle unità di tempo $1, 2, \ldots, n$.
Se gli n valori osservati non oscillano intorno ad un "valore medio" costante o non oscillano con varianza costante allora la serie temporale è detta non stazionaria. In altri termini, una serie temporale è detta stazionaria se le sue proprietà statistiche rimangono costanti nel tempo.
Per i casi non stazionari si può considerare la differenza (prima o seconda) dei valori e vedere se la serie trasformata è stazionaria. Osserviamo che il numero di fallimenti nella t-esima fase di test è dato da

$$z_t = y_t - y_{t-1}, \qquad t = 2, 3, \ldots, n.$$

La differenza prima di z, denotata con $w_i \equiv z_i - z_{i-1}$, coincide con la differenza seconda di y, infatti si ha:

$$w_i \equiv z_i - z_{i-1} = y_i - y_{i-1} - y_{i-1} - y_{i-2} = y_i - 2y_{i-2} + y_{i-2}. \qquad i = 3, 4, \ldots, n.$$

Pertanto, per il modello ARIMA con parametri $p = 0,\ d = 2$ e $q = 3$, risulta:

$$w_i = \delta + a_i - \vartheta_1 a_{i-1} - \vartheta_2 a_{i-2} - \vartheta_3 a_{i-3}$$

e quindi

$$y_i - 2y_{i-2} + y_{i-2}\delta + a_i - \vartheta_1 a_{i-1} - \vartheta_2 a_{i-2} - \vartheta_3 a_{i-3}.$$

Quest'ultima relazione indica che le differenze seconde $\{w_t\}$ costituiscono una serie di combinazioni lineari di $(a_{i-3}, a_{i-2}, a_{i-1}, a_i)$ pesate con pesi $(-\vartheta_3, -\vartheta_2, -\vartheta_1, 1)$ e possono essere utilizzate per stimare, e quindi per predire, l'affidabiltà del software.

6.6 Modelli basati sul processo di Poisson non omogeneo

In questa categoria rientrano i modelli di software reliability basati sull'ipotesi che il numero di errori che si sono verificati fino all'istante t è descritto da un processo di Poisson $N(t)$ non omogeneo (NHPP). In particolare si assume che

$$\mathbb{P}[N(t) = n] = \frac{[m(t)]^n}{n!}\, e^{-m(t)} \qquad (n = 0, 1, \ldots),$$

dove $m(t)$ è la media del processo $N(t)$. Infatti risulta che

$$\mathbb{E}[N(t)] = \sum_{n=0}^{\infty} n\, \frac{[m(t)]^n}{n!}\, e^{-m(t)} = \sum_{n=1}^{\infty} n\, \frac{[m(t)]^n}{n!}\, e^{-m(t)} = m(t) \sum_{j=0}^{\infty} n\, \frac{[m(t)]^j}{j!}\, e^{-m(t)} \equiv m(t).$$

Pertanto, in questo contesto $m(t)$ rappresenta il numero medio di errori rilevati fino all'istante t.

La funzione $m(t)$ deve essere non decrescente rispetto a t che descrive il tempo di testing. Inoltre, deve risultare

$$\lim_{t\to\infty} m(t) = a,$$

dove a rappresenta il numero totale di errori rilevati. La conoscenza di questo valore limite può aiutare a determinare se il software è pronto per essere rilasciato al cliente e quante altre risorse di testing sono richieste. La conoscenza di a può fornire anche una stima del numero di fallimenti che eventualmente saranno incontrati dal cliente dopo che il software è stato rilasciato.

Lo studio di questa classe di modelli è basato su due questioni: l'analisi del processo di Poisson non omogeneo e l'analisi delle funzioni $m(t)$ adatte a descrivere il numero medio di faults rilevati e corretti fino all'istante t. Cominciamo con l'analizzare il processo stocastico.

6.7 Il processo di Poisson non omogeneo

Ricordiamo brevemente che data una variabile aleatoria N distribuita secondo Poisson con parametro λ risulta:

$$\begin{aligned}
&\mathbb{P}[N=n] = \frac{\lambda^n}{n!}\, e^{-\lambda},\\
&\mathbb{E}[N] = e^{-n}\sum_{n=0}^{\infty} n\,\frac{\lambda^n}{n!} = \lambda\, e^{-\lambda}\, e^{\lambda} = \lambda,\\
&\mathbb{E}[N^2] = e^{-n}\sum_{n=0}^{\infty} n^2\,\frac{\lambda^n}{n!} = e^{-n}\left\{\sum_{n=0}^{\infty} n\,(n-1)\frac{\lambda^n}{n!} + \sum_{n=0}^{\infty} n\frac{\lambda^n}{n!}\right\} = \lambda^2+\lambda,\\
&\mathbb{V}[N] = \mathbb{E}[N^2] - (\mathbb{E}[N])^2 = \lambda^2+\lambda-\lambda^2 = \lambda.
\end{aligned}$$

Per un processo di Poisson $N(t)$ di intensità costante λ si ha:

$$\mathbb{P}[N(t)=n] = \frac{(\lambda\, t)^n}{n!}\, e^{-\lambda\, t}, \qquad \mathbb{E}[N(t)] = \lambda\, t, \qquad \mathbb{V}[N(t)] = \lambda\, t.$$

Osserviamo che il processo di Poisson può essere definito anche a partire dalle seguenti assunzioni:

- $\mathbb{P}[N(t+\Delta t) - N(t) = 1 | N(t) = n] = \lambda\,\Delta t + o(\Delta t)$,
- $\mathbb{P}[N(t+\Delta t) - N(t) = 0 | N(t) = n] = 1 - \lambda\,\Delta t + o(\Delta t)$,
- Eventi che si verificano in intervalli di tempo disgiunti sono indipendenti.

La notazione $o(\Delta t)$ rappresenta un infinitesimo di ordine superiore a Δt. In questo modo, denotando con

$$p_n(t) = \mathbb{P}[N(t) = n] \qquad n = 0, 1, \ldots,$$

si ha

$$p_n(t+\Delta t) = p_n(t)\,(1-\lambda\,\Delta t) + p_{n-1}(t)\,\lambda\,\Delta t + o(\Delta t)$$

da cui, dividendo per Δt e procedendo al limite per Δt che tende a zero, segue

$$\frac{d\,p_n(t)}{dt} = -\lambda\,p_n(t) + \lambda\,p_{n-1}(t). \tag{4}$$

La (4) deve essere risolta con la seguente condizione iniziale:

$$p_n(0) = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0. \end{cases} \tag{5}$$

La soluzione dell'equazione differenziale (4) con la condizione iniziale (5) è:

$$p_n(t) = \frac{(\lambda\,t)^n}{n!}\,e^{-\lambda\,t}. \tag{6}$$

Tale soluzione può essere ottenuta facendo uso della funzione generatrice delle probabilità.[3]

Se il parametro λ non dipende dal tempo il processo è temporalmente omogeneo. In questo caso, denotando con a, b due istanti di tempo con $a < b$, si ha:

$$\begin{aligned}\mathbb{P}[N(b) - N(a) = n | N(a) = 0] &= \frac{\left[\int_a^b \lambda\,dt\right]^n}{n!}\exp\left\{-\int_a^b \lambda\,dt\right\} = \frac{[\lambda\,(b-a)]^n}{n!}\exp\{-\lambda\,(b-a)\} \\ &\equiv \mathbb{P}[N(b-a) - N(0) = n | N(a) = 0].\end{aligned}$$

Quando si considera un processo di Poisson non omogeneo quest'ultima proprietà non vale più. Specificamente, sia $N(t)$ un processo di Poisson di intensità $\lambda(t) \equiv r(t)$. Nel nostro caso $N(t)$

[3] Sia

$$G(z,t) = \mathbb{E}[z^{N(t)}] = \sum_{n=0}^{\infty} z^n p_n(t)$$

la funzione generatrice delle probabilità $p_n(t)$. Dalla (4), moltiplicando per z^n e sommando su $n = 0, 1, \ldots$ si ottiene

$$\frac{dG(z,t)}{dt} = -\lambda(1-z)G(z,t) \quad \Rightarrow \quad \frac{dG(z,t)}{G(z,t)} = -\lambda\,(1-z)\,dt$$

da cui segue che

$$\ln G(z,t) = -\lambda\,(1-z)\,t + \widetilde{c} \quad \Rightarrow \quad G(z,t) = c\,\exp\{-\lambda\,(1-z)\,t\}.$$

Ricordando la condizione iniziale si ha che $G(z,0) = z^0 = 1$ da cui segue che $c = 1$. Pertanto,

$$G(z,t) \equiv= \sum_{n=0}^{\infty} z^n p_n(t) = \exp\{-\lambda\,t\}\,\exp\{\lambda\,z\,t\} = e^{-\lambda\,t}\sum_{n=0}^{\infty}\frac{(\lambda\,z\,t)^n}{n!}$$

da cui, uguagliando i coefficienti di uguali potenze di z, si ottengono le probabilità (6).
Notiamo inoltre che la funzione generatrice delle probabilità può essere utilizzata anche per valutare i momenti. In particolare si ha:

$$\mathbb{E}[N(t)] = \frac{dG(z,t)}{dz}\Big|_{z=1} = \lambda\,t, \qquad \mathbb{E}[N^2(t)] = \left[\frac{d^2G(z,t)}{dz^2} + \frac{dG(z,t)}{dz}\right]_{z=1} = \lambda^2 t^2 + \lambda\,t,$$

da cui segue immediatamente che la varianza del processo di Poisson è

$$\mathbb{V}[N(t)] = \mathbb{E}[N^2(t)] - \{\mathbb{E}[N(t)]\}^2 = \lambda\,t.$$

descrive il numero di faults rilevati e corretti, o più in generale possiamo parlare di eventi registrati, nell'intervallo di tempo $(0, t)$. Si ha:

$$\mathbb{P}[N(t) = n | N(0) = 0] = \frac{[\Lambda(t)]^n}{n!} e^{-\Lambda(t)} \qquad \mathbb{E}[N(t)] = \Lambda(t), \qquad \mathbb{V}[N(t)] = \Lambda(t),$$

dove

$$\Lambda(t) = \int_0^t \lambda(\tau)\, d\tau.$$

La funzione $\Lambda(t) \equiv m(t)$ descrive il numero medio di eventi registrati nell'intervallo di tempo $(0, t]$. Nel contesto dell'affidabilità del software $\Lambda(t)$ rappresenta il numero medio di faults rilevati e corretti fino all'istante t, inoltre $\Lambda(t)$ è non decrescente e risulta $\Lambda(0) = 0$. L'intensità del processo $N(t)$ è

$$\lambda(t) = \frac{d\Lambda(t)}{dt}$$

ed è connessa alla probabilità che si verifichi un evento in $(t, t + \Delta t)$, infatti risulta

$$\mathbb{P}[N(t + \Delta t) - N(t) = 1 | N(t) = n] = \lambda(t)\, \Delta t + o(\Delta t).$$

Notiamo esplicitamente che i risultati relativi al processo di Poisson temporalmente omogeneo basati sull'indipendenza degli incrementi valgono anche per il processo non omogeneo sebbene le espressioni debbano essere riscritte. In particolare, consideriamo il numero di eventi che si verificano nell'intervallo di tempo $(a, b]$, risulta:

$$\begin{aligned}\mathbb{P}[N(b) - N(a) = n | N(a) = n_0] &\equiv \mathbb{P}[N(b) - N(a) = n] \\ &= \frac{\left[\int_a^b \lambda(t)\, dt\right]^n}{n!} \exp\left\{-\int_a^b \lambda(t)\, dt\right\} = \frac{[\Lambda(b) - \Lambda(a)]^n}{n!} \exp\{-[\Lambda(b) - \Lambda(a)]\}.\end{aligned}$$

Sia T la variabile aleatoria "tempo di attesa per la prima occorrenza" di un evento di Poisson. Per un processo temporalmente omogeneo di intensità λ, T ha distribuzione esponenziale con media $\mathbb{E}(T) = 1/\lambda$ e varianza $\mathbb{V}(T) = 1/\lambda^2$; inoltre, l'affidabilità è data da

$$R(t) \equiv R(t|0) = \mathbb{P}(T > t) = e^{-\lambda t} \equiv R(t + s|s).$$

Per processi di Poisson temporalmente non omogenei è necessario considerare il vettore aleatorio $\mathbf{T} = (T_1, T_2, \ldots, T_n)$ che descrive gli istanti, ovviamente aleatori, di occorrenza dei primi n eventi di Poisson. Le variabili T_i non sono indipendenti ed identicamente distribuite.
In particolare, denotando con X_i le ampiezze degli intervalli di tempo che intercorrono tra un evento di Poisson e il successivo, risulta che le X_i sono indipendenti ed inoltre

$$T_i = X_1 + X_2 + \ldots + X_i = T_{i-1} + X_i.$$

Pertanto, relativamente al vettore $\mathbf{T}$, per $0 < t_1 < t_2 < \ldots t_n$ si ha:

$$\begin{aligned}F_{\mathbf{T}}(t_1, t_2, \ldots, t_n) &= \mathbb{P}[T_1 \le t_1, T_2 \le t_2, \ldots, T_n \le t_n] \\ &= \mathbb{P}[X_1 \le t_1, X_1 + X_2 \le t_2, \ldots, X_1 + X_2 + \ldots + X_n \le t_n] \\ f_{\mathbf{T}}(t_1, t_2, \ldots, t_n) &= \frac{\partial^n F_{\mathbf{T}}(t_1, t_2, \ldots, t_n)}{\partial t_1 \partial t_2 \ldots \partial t_n}.\end{aligned}$$

La variabile aleatoria "tempo di attesa residuo" denotata con T_R, conoscendo la "storia passata fino all'istante t", indicata con H_t, è ancora esponenziale e indipendente dalla storia passata, infatti risulta:

$$\begin{aligned}\mathbb{P}[T_R > x|H_t] &= \mathbb{P}[N(t+x) - N(t) = 0|H_t] = \mathbb{P}[N(t+x) - N(t) = 0] \\ &= \exp\left\{-\int_t^{t+x} \lambda(\tau)\, d\tau\right\} = \exp\left\{-[\Lambda(t+x) - \Lambda(t)]\right\}.\end{aligned}$$

La densità congiunta del vettore $\mathbf{T}$ è data da

$$f_{\mathbf{T}}(t_1, t_2, \ldots, t_n) = \frac{\partial^n F_{\mathbf{T}}(t_1, t_2, \ldots, t_n)}{\partial t_1 \partial t_2 \ldots \partial t_n} = \prod_{i=1}^{n} \lambda(t_i) \exp\left\{-\Lambda_{t_{i-1}}(t_i)\right\} \equiv \exp\left\{-\Lambda_0(t_n)\right\} \prod_{i=1}^{n} \lambda(t_i),$$

dove

$$\Lambda_y(t) = \int_y^t \lambda(\tau)\, d\tau, \qquad 0 < y < t.$$

Inoltre, la densità marginale della generica variabile T_i può essere calcolata come segue:

$$\begin{aligned}f_{T_i}(t_i) &= \int_{t_0}^{t_i} dt_{i-1} \int_{t_0}^{t_{i-1}} dt_{i-2} \ldots \int_{t_0}^{t_2} dt_1 f_{T_1, T_2, \ldots T_i}(t_1, t_2, \ldots t_i) \\ &= \frac{\exp\{-\Lambda_0(t_i)}{k!} \int_{t_0}^{t_i} dt_{i-1} \int_{t_0}^{t_{i-1}} dt_{i-2} \ldots \int_{t_0}^{t_{k+2}} dt_{k+1} \lambda(t_{k+1}) [\Lambda_0(t_{k+1})]^k \\ &= \ldots = \lambda(t_i) \frac{[\Lambda_0(t_i)]^{i-1}}{(i-1)!} \exp\left\{-\Lambda_0(t_i)\right\} \qquad i = 1, 2, 3, \ldots, n\end{aligned}$$

da cui segue che per $t > 0$ e per $i = 1, 2, 3, \ldots, n$ la funzione di distribuzione marginale è:

$$F_{T_i}(t_i) = \mathbb{P}[T_i \le t_i) = \int_0^{t_i} d\tau\, f_{T_i}(\tau) = \int_0^{t_i} d\tau\, \lambda(\tau) \frac{[\Lambda_0(\tau)]^{i-1}}{(i-1)!} \exp\left\{-\Lambda_0(\tau)\right\},$$

da cui effettuando il cambiamento di variabile di integrazione

$$x = \Lambda_0(\tau) \quad \Rightarrow \quad dx = \lambda(\tau)\, d\tau$$

segue che

$$F_{T_i}(t_i) = \int_0^{\Lambda_0(t)} \frac{x^{i-1}}{(i-1)!} e^{-x} dx = 1 - \sum_{k=0}^{i-1} \frac{\Lambda_0(t)}{k!} e^{-\Lambda_0(t)}.$$

Da quest'ultima relazione si può dedurre che affinché l'evento i-esimo si verifichi prima dell'istante t in modo che $N(t) = n$ è necessario e basta che prima di t non si verifichino meno di n eventi, cioè:

$$\begin{aligned}\mathbb{P}[N(t) = n|N(0) = 0] &= \mathbb{P}[N(t) \ge n|N(0) = 0] - \mathbb{P}[N(t) \ge n+1|N(0) = 0] \\ &= \mathbb{P}[T_n \le t] - \mathbb{P}[T_{n+1} \le t] = F_{T_n}(t) - F_{T_{n+1}}(t) = \frac{[\Lambda_0(t)]^n}{n!} e^{-\Lambda_0(t)}.\end{aligned}$$

Possiamo analizzare anche gli intervalli di attesa tra un evento e l'altro. In particolare, per $n = 0, 1, 2, \ldots$, $X_n = T_n - T_{n-1}$ è la variabile aleatoria che descrive la durata dell'n-esimo intervallo di attesa. Risulta:

$$\begin{aligned}F_{X_{n+1}|T_n}(x|t_n) &= \mathbb{P}[X_{n+1} \le x|T_n = t_n] = 1 - \mathbb{P}[X_{n+1} > x|T_n = t_n] \\ &= 1 - \mathbb{P}[T_{n+1} > x + t_n|T_n = t_n] = 1 - \exp\left\{-\Lambda_{t_n}(t_n + x)\right\},\end{aligned}$$

dove

$$\Lambda_t(t+x) = \int_t^{t+x} \lambda(\tau)\, d\tau.$$

Pertanto, l'$(n+1)$-esimo evento condizionato dal fatto che l'ultimo evento si è verificato all'istante t_n, è distribuito in accordo alla distribuzione esponenziale di parametro

$$\lambda(t_n + x) = \frac{d\Lambda_{t_n}(t_n + x)}{dx}.$$

6.8 Modelli di crescita dell'affidabilità del software

A partire dal processo di Poisson non omogeneo si possono definire svariati modelli di crescita dell'affidabilità del software in base alle differenti funzioni che definiscono la media del processo. In particolare, se $m(t)$ rappresenta la media del processo di Poisson non omogeneo all'istante t, allora l'intensità del processo all'istante di testing t è:

$$r(t) = m'(t).$$

Ovviamente, se $m(t) = mt$, come accade per il processo di Poisson omogeneo, allora $r(t) = m$.

L'affidabilità del software, cioè la probabilità che non si verifichino fallimenti in $(s, s+t)$ dato che l'ultimo fallimento si è verificato all'istante s con $s \geq 0$ e $t \geq 0$ è

$$R(t|s) = \mathbb{P}(T > t+s|T > s) = \frac{\mathbb{P}(T > t+s, T > s)}{\mathbb{P}(T > s)} = \frac{\mathbb{P}(T > t+s)}{\mathbb{P}(T > s)} = \exp\left\{-\int_s^{s+t} r(\vartheta)\, d\vartheta.\right\}$$

Il tasso di rilevazione degli errori (fault detection rate) all'istante t è definito come

$$d(t) = \frac{r(t)}{a - m(t)}$$

e rappresenta "la rilevabilità"di un errore all'istante corrente t.

Analizziamo alcuni dei modelli di software reliability basati sul processo di Poisson non omogeneo.

Modello Goel-Okumoto Questo è uno dei modelli più popolari nel campo della software reliability, è anche detto modello esponenziale. In questo caso si ha

$$m(t) = a(1 - e^{-bt}), \qquad r(t) = abe^{-bt} \qquad (a, b > 0),$$

dove $a = m(\infty)$ e b è il tasso di rilevazione, infatti risulta

$$d(t) = \frac{r(t)}{a - m(t)} = b.$$

In generale la software reliability tende a migliorare e può essere trattata come un processo di crescita durante la fase di test.
In altri termini, la crescita della software reliability è dovuta ad errori fissati così che, sotto ampie condizioni, i modelli sviluppati per predire la crescita di popolazioni possono anche essere applicati come modelli di software reliability. Questi modelli implicano che il numero cumulativo di errori

rilevati segua un andamento noto. la curva di Gompertz e la curva logistica sono due esempi di tali curve.

Modello Gompertz Molti costruttori di computer e case software giapponesi hanno usato questo modello perché, tra i modelli caratterizzati da una funzione media $m(t)$ di forma sigmoidale, è uno dei modelli più semplici . In questo caso si ha

$$m(t) = ak^{b^t} \qquad a > 0,\ b \in (0,1),\ k \in (0,1).$$

Per interpretare il significato dei parametri osserviamo che $a = m(\infty)$, infatti, poiché $b \in (0,1)$ si ha che $b^t \to 0$ quando $t \to \infty$. Inoltre, risulta

$$\lim_{t\to 0} m(t) = ak \quad \Rightarrow \quad k = \frac{m(0)}{a}$$

e, ricordando che $\frac{dc^x}{dx} = c^x \ln a$, si ha $r(t) = \ln b\, \ln k\, b^t\, m(t)$ da cui

$$d(t) = \frac{\ln b\, \ln k\, b^t\, m(t)}{a - m(t)}.$$

Quindi, $a = m(\infty)$ è il numero medio totale di fallimenti, k è connesso a $m(0)$ e b al tasso di crescita. I parametri k e b possono essere stimati ad esempio usando l'analisi di regressione.
Notiamo inoltre che

$$\begin{aligned}\frac{d^2m(t)}{dt^2} &= \ln b\, \ln k[\ln b\, \ln k\, b^{2^t}\, m(t) + \ln b\, b^t\, m(t)] \\ &= (\ln b)^2\, \ln k\, m(t)\, b^t\, (\ln k\, b^t + 1) = 0 \quad \Rightarrow \quad b^t = [\ln(1/k)]^{-1} \quad \Rightarrow \quad t = \log_b[\ln(1/k)]^{-1}\end{aligned}$$

Modello logistico Assumiamo che

$$m(t) = \frac{a}{1 + k\, e^{-bt}} \qquad a > 0,\ b > 0,\ k > 0,$$

dove $a = m(\infty)$. Osserviamo che

$$m(0) = \frac{a}{1+k} \quad \Rightarrow \quad (1+k)\, m(0) = a \quad \Rightarrow \quad k = \frac{a - m(0)}{m(0)}.$$

Invece b è connesso al tasso di crescita. I parametri k e b possono essere stimati fittando i dati relativi ai fallimenti. Anche in questo caso $m(t)$ presenta un punto di flesso così che la sua forma è di tipo sigmoidale.
La funzione intensità del processo è

$$r(t) = \frac{dm(t)}{dt} = \frac{a\, k\, b\, e^{-bt}}{(1 + k\, e^{-bt})^2}$$

da cui

$$d(t) = \frac{r(t)}{a - m(t)} = \frac{b}{1 + k\, e^{-bt}}.$$

Inoltre si ha:

$$\frac{d^2m(t)}{dt^2} = a\,k\,b\,\frac{-b\,e^{-bt}(1+k\,e^{-bt})^2 + 2\,(1+k\,e^{-bt})\,b\,k\,e^{-bt}}{(1+k\,e^{-bt})^4}$$
$$= a\,k\,b^2 e^{-bt}\frac{2\,k\,e^{-bt} - (1+k\,e^{-bt})}{(1+k\,e^{-bt})^3} = 0 \quad \Rightarrow \quad 2\,k\,e^{-bt} - (1+k\,e^{-bt}) = 0$$
$$\Rightarrow \quad e^{-bt} = \frac{1}{k} \quad \Rightarrow \quad -t = -\frac{\ln k}{b}.$$

Quindi, nel punto di ascissa $t = \frac{\ln k}{b}$ la funzione media presenta un punto di flesso. Se $k > 1$ il punto di flesso risulta positivo, così che la curva logistica è di forma sigmoidale, se invece $k \in (0,1)$ allora il punto di flesso ha ascissa negativa.
Pertanto, la curva logistica è di forma sigmoidale per $k > 1$ e $b > 0$.

Modello di Goel generalizzato Per descrivere una situazione in cui l'intensità di fallimento inizialmente cresce poco e poi ricomincia a decrescere, Goel propose una semplice generalizzazione del modello di Goel-Okumoto caratterizzata da un ulteriore parametro c. In questo caso si assume

$$m(t) = a(1 - e^{-bt^c}) \qquad a > 0,\, b > 0,\, c > 0,$$

dove $a = m(\infty)$ rappresenta il numero medio di errori che eventualmente saranno rilevati, b e c sono parametri che riflettono la qualità del test.
Se $c = 1$ il modello si riduce al modello di Goel-Okumoto in cui b rappresenta il tasso di rilevazione. Notiamo che per $c > 1$ la funzione media presenta un punto di flesso, mentre per $c \in (0,1)$ la funzione è monotona crescente.
In questo caso la funzione intensità del processo è:

$$r(t) = a\,b\,c\,e^{-bt^c}\,t^{c-1},$$

e il tasso di rilevazione degli errori è

$$d(t) = \frac{r(t)}{a - m(t)} = \frac{a\,b\,c\,e^{-bt^c}\,t^{c-1}}{a - a(1-e^{-bt^c})} = \frac{b\,c\,e^{-bt^c}\,t^{c-1}}{e^{-bt^c}} = b\,c\,e^{-bt^c}.$$

Inoltre,

$$\frac{d^2m(t)}{dt^2} = \frac{dr(t)}{dt} = a\,b\,(c-1)c\,e^{-bt^c}\,t^{c-2} - a\,b^2c^2\,e^{-bt^c}\,t^{2(c-1)}$$
$$= a\,b\,c\,e^{-bt^c}\left[(c-1)t^{c-2} - b\,c\,t^{2(c-1)}\right],$$

quindi $r(t)$ non è strettamente monotona in quanto la sua derivata si annulla in $t = (c-1)/bc$.

Modello di Yamada delayed S-shaped In questo caso $m(t)$ è modificato per ottenere una curva sigmoidale tale che il tasso di fallimento inizialmente cresce e poi decresce in modo esponenziale. Il processo di rilevazione degli errori descritto da tale curva si può interpretare come un *processo di apprendimento* in quanto le capacità del tester migliorano col passare del tempo. Per questo modello si ha:

$$m(t) = a[1 - (1 + b\,t)\,e^{-bt}] \qquad a > 0,\, b > 0,$$

dove $a = m(\infty)$. Inoltre,

$$r(t) = a\,b\,e^{-bt}[1 - (1 + b\,t)\,e^{-bt}] - a\,b\,e^{-bt} = a\,b^2\,t\,e^{-bt},$$

ed inoltre, il tasso di rilevazione degli errori è dato da

$$d(t) = \frac{r(t)}{a - m(t)} = \frac{a\,b^2\,t\,e^{-bt}}{a - a[1 - (1 + b\,t)\,e^{-bt}]} = \frac{b^2 t}{1 + b\,t}$$

e tende a b per $t \to \infty$.

Modello Inflected S-shaped Questo modello fu proposto da Ohba e si basa sull'idea che la curva di crescita della software reliability diventa sigmoidale se gli errori del programma sono dipendenti intendendo con ciò che alcuni errori non sono rilevabili prima che altri siano rimossi. Per questo modello risulta che

$$m(t) = a\,\frac{1 - e^{-bt}}{1 + \psi(\vartheta)\,e^{-bt}} \quad \text{con} \quad \psi(\vartheta) = \frac{1 - \vartheta}{\vartheta}, \qquad a > 0,\, b > 0,\, \vartheta > 0.$$

Il significato dei parametri è il seguente: $a = m(\infty)$, ϑ rappresenta il tasso di inflessione e indica il rapporto tra il numero di errori rilevabili e il numero totale di errori nel software, b invece è il tasso di individuazione degli errori e ψ è il fattore di inflessione. In questo caso risulta che

$$r(t) = \frac{a\,b\,c\,e^{-bt}(1 - e^{-bt})}{(1 + c\,e^{-bt})^2} + \frac{a\,b\;e^{-bt}}{1 + c\,e^{-bt}} = \frac{a\,b\,(1 + c)\,e^{-bt}}{c + e^{-bt}},$$

da cui si può facilmente ricavare il tasso di rilevazione degli errori $d(t) = \dfrac{r(t)}{a - m(t)}$.

Modello di Duane modificato In un report del 1962, Duane presentò i dati di fallimento di alcuni sistemi durante la fase di sviluppo. In quell'occasione fu notato che, diversamente dal tempo operazionale cumulativo, il mean time between failure cumulativo, se plottato su scala log log, si approssima ad una retta. Sulla base di questa osservazione fu proposto il modello di Duane modificato ipotizzando per il numero medio di fault all'istante t si presenta nella seguente forma funzionale:

$$m(t) = a\left[1 - \left(\frac{b}{b + t}\right)^c\right], \qquad a > 0,\, b > 0,\, c > 0,$$

con $a = m(\infty)$. In questo caso il failure rate e il tasso di rilevazione degli errori sono rispettivamente

$$r(t) = a\,c\left(\frac{b}{b + t}\right)^{c-1}\frac{b}{(b + t)^2} = \frac{a\,b^c\,c}{(b + t)^{c+1}}, \qquad d(t) = \frac{r(t)}{a - m(t)} = \frac{c}{b + t}.$$

Modello con due tipi di errori (Yamada e Osaki) Gli autori proposero un modello di software reliability immaginando l'esistenza di due tipi di errori:

- *Errori di tipo 1* che sono facili da rilevare,
- *Errori di tipo 2* più complessi da rilevare.

Denotando con p_i la percentuale di errori di tipo i con $i = 1, 2$ risulta che p_1, $p_2 \in (0, 1)$ ed inoltre $p_1 + p_2 = 1$. In questo caso il numero medio di faults rilevati al tempo t soddisfa la relazione:

$$m(t) = a \sum_{i=1}^{2} p_i (1 - e^{-b_i t}) \equiv a \left[1 - \sum_{i=1}^{2} p_i e^{-b_i t} \right], \qquad a > 0,\ 0 < b_2 < b_1 < 1,$$

dove b_i rappresenta il tasso di rilevazione per errori di tipo i per $i = 1, 2$. È evidente la familiarità tra questo modello e il modello di Goel-Okumoto, visto precedentemente, a cui ci si può ricondurre facilmente supponendo che ci sia un solo tipo di errori.
L'intensità del processo di Poisson sottostante e la rilevabilità sono dati rispettivamente da

$$r(t) = a \sum_{i=1}^{2} p_i\, b_i\, e^{-b_i t}, \qquad d(t) = \frac{r(t)}{a - m(t)} = \frac{\sum_{i=1}^{2} p_i\, b_i\, e^{-b_i t}}{\sum_{i=1}^{2} p_i\, e^{-b_i t}}.$$

Modello con funzione testing-effort tipo Weibull Yamada *et al.* proposero un modello di crescita della software reliability incorporando la quantità di tentativi (interpretata come *sforzo*) effettuati durante la fase di testing del software. Per questo modello il numero medio di faults rilevati è

$$m(t) = a \left[1 - \exp\left\{ -b\, \alpha \left(1 - \exp\{-\beta\, t^{\gamma}\} \right) \right\} \right], \qquad a, b, \alpha, \beta, \gamma > 0,$$

dove

- α rappresenta la quantità totale di testing-effort richiesta dal software testing,

- β è un parametro di scala,

- γ è un parametro di forma,

- b è connesso al tasso di rilevabilità degli errori.

Per questo modello l'intensità del processo di Poisson sottostante e la rilevabilità sono dati rispettivamente da

$$\begin{aligned} r(t) &= a\, b\, \exp\left\{ -b\, \alpha \left(1 - \exp\{-\beta\, t^{\gamma}\} \right) \right\} \exp\{-\beta\, t^{\gamma}\}\, \alpha\, \beta\, \gamma\, t^{\alpha - 1} \\ &= a\, b\, \alpha\, \beta\, \gamma\, t^{\alpha - 1} \exp\left\{ -b\, \alpha \left(1 - \exp\{-\beta\, t^{\gamma}\} \right) - \beta\, t^{\alpha} \right\}, \\ d(t) &= \frac{r(t)}{a - m(t)} = b\, \alpha\, \beta\, \gamma\, t^{\alpha - 1}\, \exp\{-\beta\, t^{\gamma}\}. \end{aligned}$$