Vol. 31 No. 4　　情　報　処　理　　Apr. 1990

解　説

ソフトウェアの信頼性†

当麻　喜弘††

1. はじめに

最近，ハードウェアのフォールトトレランス技術が向上するにともない，ソフトウェアの信頼性が問題になっている．

たとえば，文献1)によると，システムダウンの原因のうちソフトウェアの不備によるものが全体の約27%であった．また，ベル電話会社の電子交換システム（ESS）の経験によれば，システムダウンの約1/3はソフトウェアの問題に帰せられるという[2]．

そこで，ソフトウェアの信頼性を数量的に評価できるか，ソフトウェアに対するフォールトトレランス技術が存在するかという2点が問題になる．

2. ソフトウェアの信頼性評価

「ソフトウェアの信頼性」が何を意味するかという点についてはまだあまりはっきりしていないが，いろいろな立場から直接的，間接的に評価する試みが行われている．それらを整理すると次のようになる．

1．誤りの生起頻度の推定，すなわち，誤りが生じる時間間隔を推定する．

2．カバリッジ（coverage），すなわち，プログラム上のテストされたパスの相対的割合を評価する．

3．残存フォールト☆数の推定，すなわち，テスト後にプログラム中に残っているフォールトの数を推定する．

以下では，特に1，3について説明しよう．

2.1 誤りの生起頻度の推定

プログラムのフォールトによって動作中に誤りが生じる．この誤りの生起を確率過程とみなし，実際の観測結果から誤りが生じる時間間隔を推定する．対象とする動作が，テスト／デバッグ中であれば，時間間隔の推定の際にパラメータの一つとして定められるプログラム中の初期フォールト数（デバッグを始めるときにプログラム中に含まれているフォールトの総数をこう呼ぶことにする）が，むしろ，主な関心事になろう．また，動作がサービス中であれば，誤りの生起頻度の推定は，ハードウェアの場合の誤り率に直接対応するものとなる．

対象とする過程をどのような確率過程とみなすかということからいろいろなモデルが提案されている．ここでは，必ずしも良いモデルというわけではないが，代表的な二つのモデルを紹介しておく．

2.1.1 Jelinski-Moranda のモデル[3]

このモデルは，

- 誤りが検出されそのフォールトが除去されてから次の誤りが検出されるまでの時間間隔は指数分布に従う
- 誤りの生起頻度 λ_i は，（個々のフォールトには依存せずに）そのときにプログラム中に残存するフォールトの数に比例する
- 誤りの検出ごとに（一つずつ完全に）フォールトが除去される

と仮定する．すなわち，**図-1** に示すように，$i-1$ 番目の誤りが検出されてから i 番目の誤りが検出されるまでの時間間隔を表す確率変数を T_i とすると，$T_i \leq t_i$ となる確率 $\mathrm{Prob}(T_i \leq t_i)$ は

$$\mathrm{Prob}(T_i \leq t_i) = 1 - e^{-\lambda_i t_i} \equiv F(t_i) \qquad (1)$$

であるとする．さらに，λ_i は，初期フォールト数を m とすれば，

$$\lambda_i = k\{m-(i-1)\} \qquad (2)$$

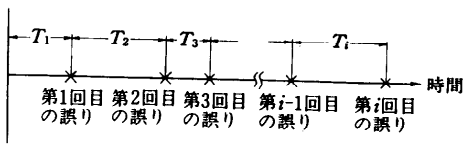

図-1 誤りの生起

† On Software Dependability by Yoshihiro TOHMA (Department of Computer Science, Tokyo Institute of Technology).

†† 東京工業大学工学部情報工学科

☆ ソフトウェアの場合，正しくない部分をバグ（bug）ということがあるが，ハードウェアの場合と統一してここではフォールトということにする．

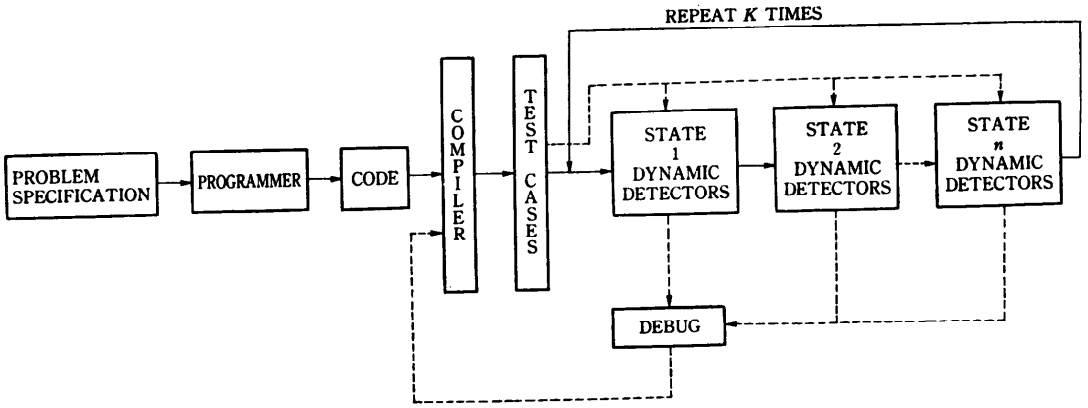

図-2 NASA の実験

したがって，T_i の確率密度関数 $f(t_i)$ は

$$f(t_i)=dF(t_i)/dt_i=\lambda_i e^{-\lambda_i t_i} \qquad (3)$$

さて，実際に，$1, 2, \cdots, n$ 回目の誤りを検出するまでの時間間隔が，それぞれ，$\Delta_1, \Delta_2, \cdots, \Delta_n$ と観測されたとする．複合確率密度関数

$$L=\prod_{i=1}^{n} f(\Delta_i) \qquad (4)$$

を尤度関数とし，これを最大にするようにパラメータ m, k を選ぶ．実際は

$$\frac{\partial(\ln L)}{\partial m}=0, \quad \frac{\partial(\ln L)}{\partial k}=0$$

の二式を連立させて，m, k についてそれぞれ解く．この場合，解析的には解けないので，数値解法を用いる．

n 個の誤りを観測して m, k の推定値を決定すれば，$n+1$ 番目の誤りが検出されるまでの時間間隔の推定値 $\hat{T}_{n+1}$* は，

$$\hat{T}_{n+1}=\frac{1}{\lambda_{n+1}}=\frac{1}{\hat{k}(\hat{m}-n)} \qquad (5)$$

2.1.2 NASA の実験[4]

式(2)は，いずれのフォールトも同じような頻度で(互いに独立に) 誤りを起こす，ということを暗黙に仮定している．しかし，この仮定が妥当か否かを調べるために，NASA のラングリー研究センタでは図-2に示すような実験を行った．

仕様に従って作成されたプログラムは，実行段階のいくつかのテストケース（アクセプタンステスト）をパスするまでデバッグされた後，実験に供される．

実験は，プログラムへの入力をランダムに次々と生成して加える．一つの入力ケースごとに結果がチェックされ，正しければ次の入力が加えられる．最初の誤りが検出されると**，原因となったフォールトを見いだし取り除く．このときまでのプログラムの状態を状態1とし，最初のフォールトを除去するまでの段階を第1段階と呼ぶ．第1段階でのデバッグの状況，及び，誤りを検出するまでの経過時間を記録しておく．

第1段階でデバッグされたプログラムは先のアクセプタンステストについて再びテストされ，パスすると前と同様な実験が継続される．このときのプログラムの状態は状態2であり，次の新たな誤りが検出されるまでの段階が第2段階である．以後，第1段階と同様な経過を繰り返す．

あらかじめ定めた第 n 段階を終了すると，先の記録をもとにしてプログラムを第1段階の直前の状態に戻し，改めてランダムに生成した入力を用いて，実験を繰り返す．これによってそれぞれの各段階が繰り返されることになる．

レーダサイトで観測された結果（たとえば，高度，方向，速度など）から敵の飛行機であるか否かを判定し，敵機の場合，最適な基地から迎撃機を発進させる指令を出す，というプログラムに関して実験を行った．

第 j 段階の誤りの生起頻度を $\lambda_j{}^d$ で表す．実験開始時のプログラム中の初期フォールト数を m とすれば，第 j 段階のプログラム中のフォールト数は $m-(j-1)$ である．もし，式(2)に示されるように，誤りの生起頻度がプログラム中のフォールト数に比例するとすれば，上の $\lambda_j{}^d$ は j とともに直線的に変化する（減少す

* 一般に x の推定値を $\hat{x}$ で表す．

** 誤りの検出は十分に枯れた既成のプログラムの結果を正しいとし，それとの比較で判断した．

る）はずである．しかし，そのような結果は得られず，むしろ，$|\ln \lambda_j{}^d|$ がほぼ直線的に増大しているという．このことから，NASA の実験は，Jelinski-Moranda モデルに疑問を投げかけている．

さらに，各段階で検出されたフォールトを記録しているので，実験の繰り返しに際して同一フォールトに着目すれば，フォールトごとの誤り生起頻度 $\lambda_j{}'$ を測定することができる．実験結果によれば，$\lambda_j{}'$ は，j によって大きく異なっている（きわめて大きなオーダの差）．このことからも，それぞれのフォールトが同じ生起頻度で誤りを起こすとしている Jelinski-Moranda モデルには問題があるとしている．

2.1.3 Littlewood-Verrall のモデル[5]

フォールトごとの誤り生起頻度が一様でないのであれば，生起頻度自身も確率変数とみなそうというのが Littlewood-Verrall モデルの基本的考え方である．Jelinski-Moranda の場合と同様に，$i-1$ 番目の誤りが検出されてから i 番目の誤りが検出されるまでの時間間隔を表す確率変数 T_i の確率密度関数 $\mathrm{pdf}(t|\lambda_i)$ は

$$\mathrm{pdf}(t|\lambda_i)=\lambda_i e^{-\lambda_i t} \tag{6}$$

であるとし，さらに，λ_i も

$$\mathrm{pdf}(\lambda_i|b,\Phi(i))=\frac{\Phi(i)\{\Phi(i)\lambda_i\}^{b-1}}{\Gamma(b)}e^{-\Phi(i)\lambda_i} \tag{7}$$

なる確率密度関数をもつ確率変数であるとする．式(7)の確率密度関数をもつ確率分布を，ガンマ分布という．パラメータ b を変えることにより適当な形の分布が得られるので，ガンマ分布が選ばれたのである．また，後になるほど誤りは生じにくくなるという傾向を導入するために，パラメタ $\Phi(i)$ が用いられている．$\Phi(i)$ が大きくなると，λ_i は値の小さなほうに分布する．つまり，誤りにくくなるわけである．

式(7)を式(6)に代入することによって，T_i の平均的な確率密度関数（確率密度関数の期待値）が得られる．すなわち，

$$\mathrm{pdf}(t|\lambda_i)=\int_0^\infty \mathrm{pdf}(t|\lambda_i)\,\mathrm{pdf}(\lambda_i|b,\Phi(i))d\lambda_i$$

$$=\frac{b\{\Phi(i)\}^b}{\{t+\Phi(i)\}^{b+1}} \tag{8}$$

なお，Littlewood と Verrall は，あまり明確な根拠はないが適当に，$\Phi(i)$ を

$$\Phi(i)=e^{(2.0+0.2i)} \tag{9}$$

と仮定している．

2.1.4 適合性のチェック

仮定した確率分布が実際の確率分布を正しく表しているか否かをチェックする方法はいろいろ知られているが，ここでは，分布関数を利用する方法を紹介しよう．確率変数 T_i をその分布関数に代入した $F(T_i)=Y_i$ は再び確率変数となるが，それは 0 と 1 の間で一様な確率密度関数 $g(y_i)=1$ をもつ，一様ランダムな確率変数となることが分かっている．したがって y_i の期待値 $\overline{y_i}^{\star}$ は

$$\overline{y_i}=\int_0^1 y_i g(y_i)dy_i=\frac{1}{2} \tag{10}$$

よって，y_i の観測値 $F(\Delta_i)$, $i=1,2,\cdots,n$ の正規化累積値

$$s_j=\frac{\sum_{i=1}^{j}F(\Delta_i)}{\sum_{i=1}^{n}F(\Delta_i)} \tag{11}$$

は，**図-3** の直線が示すように，j の増加とともに 45 度の傾斜で上昇していくはずである．

Jelinski-Moranda モデルと Littlewood-Verrall モデルについて調べてみると，主として，前者は 45 度の直線より上に観測結果が分布するのに対し，後者は下に分布している．このことは，前者では，i が小さな範囲では（すなわち，観測の初期のころは）分布関数の仮定が実際より大きすぎ，i が大きくなると（すなわち，観測の後半では）分布関数の仮定が小さすぎる（したがって，T_i の推定が大きすぎる）ことを，逆に後者では，初期に大きすぎ，後半で小さすぎることを意味する．

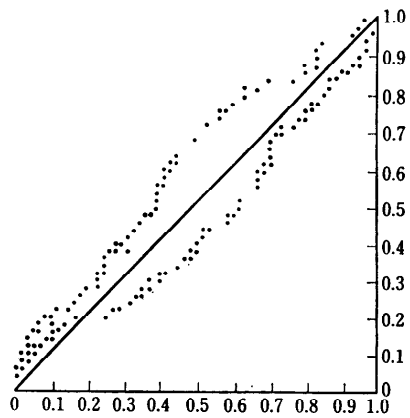

図-3 s_j の成長

☆ 一般に x の期待値を $\bar{x}$ で表す．

2.2 残存フォールト数の推定

プログラムの開発にはデバッグが重要であるが，初めにいくつ（初期）フォールトが含まれていたか分かっていないから，テストとデバッグを繰り返しても，いつの時点でフォールトが全部取り除かれたのか実は分からない．そこで，なんとかして，プログラム中の残存フォールトの数を知ることができないかということになる．決定的に知ることは不可能であるから，なんらかの方法で推定する．初期フォールトの数 m を推定できれば，デバッグの途中で除去したフォールトの数は分かるので，その時点の残存フォールト数を推定できる．結局，問題は m の推定に帰着する．

2.2.1 数式による近似

横軸にテストの経過，たとえば，テスト時間，縦軸に検出されたフォールトの累積数を取ると，図-4のような曲線（実線）が得られる．これを，次のような数式で近似する．

$$y=ka^{b^t},\quad a<1,\ b<1 \tag{12}$$

この式の曲線を Gompertz 曲線という．k が全フォールト数，すなわち，m を表している．したがって，k の推定値をもって m の推定値とする．パラメタの推定は適当な変数変換によって線形化し，最小自乗誤差法を用いる．

図-4 の点線Gが，この曲線を実際のデータに適用した状況を示す．

式(12)の代わりに

$$y=\frac{k}{1+be^{-at}}\quad 0<a,\ 0<b \tag{13}$$

を用いることもある．この式の曲線はロジスティック (logistic) 曲線と呼ばれる．Gompertz 曲線の場合と同様に，

$$\lim_{t\to\infty} y=k \tag{14}$$

この曲線の適合状況を図-4 の点線Lで示す．

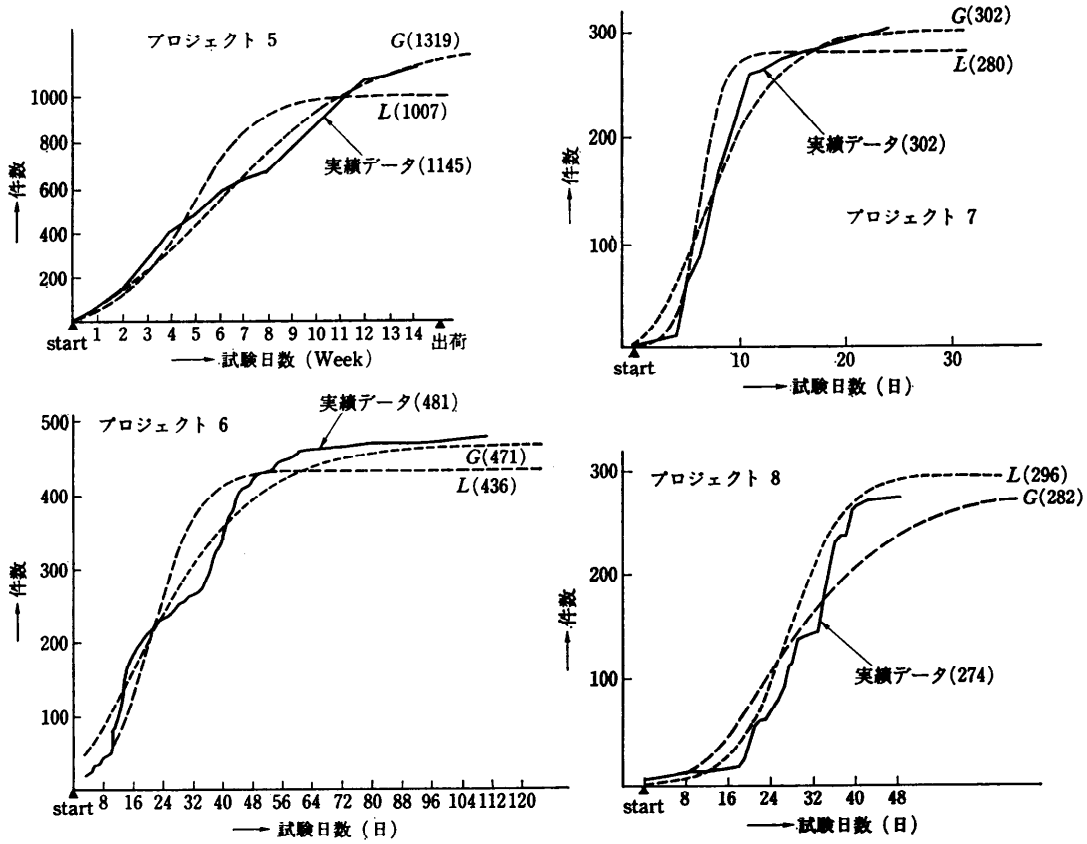

図-4 累積フォールト数成長曲線

2.2.2 非斉次ポアッソン過程 (NHPP) の適用

0から t までの間の誤りの生起回数を $C(t)$ とする．NHPP モデルでは，$C(t)=x$ となる確率 Prob$\{C(t)=x\}$ を

$$\text{Prob}\{C(t)=x\}=\frac{H(t)^x}{x!}e^{-H(t)} \tag{15}$$

と置く．すると

$$\overline{C(t)}=H(t) \tag{16}$$

このことから，$H(t)$ を平均値関数という．$\hat{m}$ は $H(\infty)$ で与えられる．

テストの終わりごろになると，大部分のフォールトが検出されるので，$H(t)$ の増加は鈍くなるであろう．Goel と Okumoto は[6]，$H(t)$ の増加率 $dH(t)/dt$ はその時点でプログラム中に残留するフォールトの数（の期待値）に比例すると考えた．すなわち，

$$\frac{dH(t)}{dt}=b\{H(\infty)-H(t)\}$$

$H(0)=0$, $H(\infty)=a$ と置いて上式を解くと，

$$H(t)=a(1-e^{-bt}) \tag{17}$$

この場合，

$$\hat{m}=\hat{a} \tag{18}$$

となる．

$C(t_i)$ が NHPP に従う場合，$C(t_i)-C(t_{i-1})$ も平均値関数 $H(t_i)-H(t_{i-1})$ の NHPP に従うことが証明されている．すなわち，

$$\text{Prob}\{C(t_i)-C(t_{i-1})=c(t_i)-c(t_{i-1})\}$$
$$=\frac{\{H(t_i)-H(t_{i-1})\}^{c(t_i)-c(t_{i-1})}}{\{c(t_i)-c(t_{i-1})\}!}e^{-\{H(t_i)-H(t_{i-1})\}} \tag{19}$$

尤度関数 L を

$$L=\prod_{i=1}^{n}\text{Prob}\{C(t_i)-C(t_{i-1})=c(t_i)-c(t_{i-1})\}$$
$$=e^{-H(t_n)}\prod_{i=1}^{n}\frac{\{H(t_i)-H(t_{i-1})\}^{c(t_i)-c(t_{i-1})}}{\{c(t_i)-c(t_{i-1})\}!} \tag{20}$$

と置く．次の二つの連立方程式を a, b についてといた解が $\hat{a}$, $\hat{b}$ を与える．

$$\frac{\partial(\ln L)}{\partial a}=0,\quad \frac{\partial(\ln L)}{\partial b}=0 \tag{21}$$

山田ら[7]は，平均的に，新たに検出される誤りの数は初期フォールトの数とその時点までに検出された誤りの累積数の差に比例するとし，さらに，新たに検出されるフォールトの数はその時点までに検出された誤りの累積数とフォールトの累積数の差に比例すると考えて，次のような $H(t)$ を誘導した．

$$H(t)=a\{1-(1+bt)e^{-bt}\},\ b>0 \tag{22}$$

フォールトの検出累積数の成長曲線がS字形となることが多いが，式(22)の $H(t)$ はこの傾向を表すことができる．

上に述べた二つの平均値関数を用いて実際のデータに NHPP モデルを適用した例を，図-5 に示す．図中の $\hat{m}(t)$ と書かれた曲線は式(17)の平均値関数を用いた場合の推定曲線であり，$\hat{M}(t)$ と書かれた曲線は式(22)を用いた場合を示す．

大場らは[8]，新しく検出されるフォールトの数は残存フォールト数のみならずこれまでに検出されたフォールトの数にも一部依存すると考えた．これより

$$H(t)=\frac{a(1-e^{-bt})}{1+\frac{1-c}{c}e^{-bt}} \tag{23}$$

を求めている．c がすでに検出されたフォールトへの相対的な依存度を表している．さらに，山田らは[9]，2種類の生起頻度の項をもつように指数関数を拡張した

$$\left.\begin{aligned}&H(t)=a\sum_{i=1}^{2}p_i\{1-e^{-b_it}\}\ 0<b_2<b_1\\&\sum_{i=1}^{2}p_i=1,\ 0<p_i<1\end{aligned}\right\} \tag{24}$$

を提案している．

2.2.3 超幾何分布の利用[10]

検出されたフォールトの累積数の成長を曲線で表す場合，横軸に時間を取ると，コンピュータのアイドル時間をも含めた単純な経過時間（カレンダタイムということがある）なのか，コンピュータが本当にテストしている時間の累積値なのか，はっきりしないことが

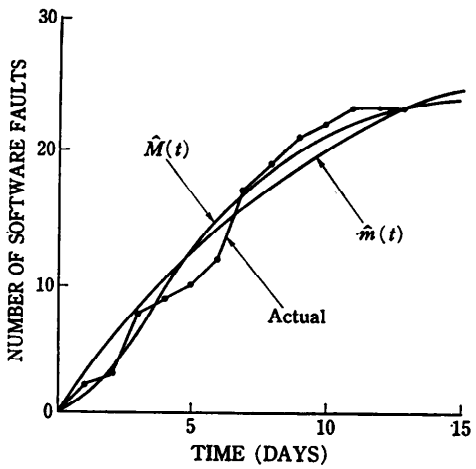

図-5 NHPP モデル

ある.

以下では，単にテストといった場合，テストケース (test instance) の集合を指し，横軸の $i=1,2,\cdots$ は第 i 回目のテストケースを示すものとする．こうすれば時間の曖昧さを避けることができる．第 i 回目のテストケースを t_i で表す.

フォールト f_i のテストケース t_i に対する反応関数 (response function) $r(i,j)$ を次のように定義する．t_i の印加によって f_j が誤りを生じるなら

$$r(i,j)=1$$

その他の場合を

$$r(i,j)=0$$

とする．$r(i,j)=1$ のとき，f_j は t_i に反応するという.

t_i に反応するフォールトの数を，t_i に関する反応係数 (sensitivity factor) $w(i)$ ということにする．すなわち,

$$w(i)=|\{f_j\in F|r(i,j)=1\}| \tag{25}$$

ここで F は初期フォールトの集合である.

ここで，以下のことを仮定しよう.

- t_i に反応するフォールトは，次のテストケース t_{i+1} が印加される前に完全に取り除かれる.
- デバッグの際に新たなフォールトが挿入されることはない.
- 本来，t_i に反応するフォールトは決定的に定まっているが，どのテストケースをいつの時点で加えるかは一定していず，むしろランダムに選ばれることが多いと思われる．そこで，t_i に反応するフォールトは m 個のフォールトからランダムに選ばれた $w(i)$ 個であるとする.

テストケース $t_1,t_2,\cdots t_{i-1}$ を加えた時点までに検出されたフォールトの総数を $C(i-1)$ で表そう．すると，t_i によって新たに検出されるフォールトの数 $N(i)$ が x となる確率 $\mathrm{Prob}\{x|m,C(i-1),w(i)\}$ は

$$\mathrm{Prob}\{x|m,C(i-1),w(i)\}=\frac{\binom{m-C(i-1)}{x}\binom{C(i-1)}{w(i)-x}}{\binom{m}{w(i)}} \tag{26}$$

x は明らかに

$$0\le x\le U_x,\ U_x=\min\{w(i),m-C(i-1)\} \tag{27}$$

の範囲にある．式(26)が表す確率分布を超幾何分布 (Hyper-Geometric Distribution) という．$\overline{N(i)}$ は

$$\overline{N(i)}=\{m-C(i-1)\}\frac{w(i)}{m} \tag{28}$$

となることが知られている[11].

もちろん，$C(i-1)=\sum_{k=1}^{i-1}N(k)$ であるが，

$$\hat{C}(i-1)=\sum_{k=1}^{i-1}\overline{N(k)} \tag{29}$$

をもって $C(i-1)$ の推定値とする．ただし，$\hat{C}(0)=0$ とする.

さて，式(28)の $C(i-1)$ の代わりに式(29)を用いれば，

$$\hat{C}(i)=\hat{C}(i-1)\left\{1-\frac{w(i)}{m}\right\}+w(i) \tag{30}$$

いま，m と $w(i)$ が分かっているものとすれば，上式を再帰的に適用し，$\hat{C}(i)$, $i=1,2,\cdots$ を次々と求めることができる．実際には m や $w(i)$ を定めるパラメータは分かっていないので，$\hat{C}(i)$ と観測結果の $C(i)$ との差が最小となるように m や $w(i)$ のパラメータの値を選び，それらを推定値とする．差の評価としては，

$$\left.\begin{aligned}\text{EF 1}&=\sum_{i=1}^{n}|C(i)-\hat{C}(i)|/n\\ \text{EF 4}&=\sum_{i=1}^{n}\{C(i)-\hat{C}(i)\}^2/n\end{aligned}\right\} \tag{31}$$

などが考えられる．これらの差の評価値を最小にする m や $w(i)$ のパラメタを解析的に定める方法はまだ得られていないので☆，これらを逐次的に変えて探す.

実際のテストデータとして $w(i)$ が陽に与えられることはまずない．1回のテストケースに反応するフォールトの数は，そのテストケースに従事したテストワーカの数，テスト項目数，テストプロセスの実際の処理時間，などによって変わるであろう．したがって，$w(i)$ はこれらの関数として取り扱う[12].

表-1 のテストデータではテストケースごとのテストワーカの数 $tester(i)$ が与えられている．そこで

$$w(i)=A\{tester(i)\}^p,\quad A=w_m/[\max\{tester(i)\}]^p \tag{32}$$

と置き，w_m, p を推定する.

図-6 に，上のようにして求めた $\hat{C}(i)$ と実際の $C(i)$ との比較を示す．$\hat{m}$, $\hat{p}$, $\hat{w}_m$ を**表-2** に示す．これらは，m に関しては 450～600 の範囲を刻み 3 で，p に関しては 1.0～3.0 の範囲を刻み 0.2 で，また w_m に関しては 20～45 の範囲を刻み 3 で逐次的に変えて，求めたものである.

全テスト期間を通じてテストの性格が変わるような場合は，上の超幾何分布モデルを全テストケースに対して，一括して，適用するのは適当ではない．テスト

☆ 最近，われわれのところで若干の試みが行われている.

表-1 テストデータ

Date	$N(i)$	$C(i)$	No. of Workers	Date	$N(i)$	$C(i)$	No. of Workers	Date	$N(i)$	$C(i)$	No. of Workers
1	5*	5*	4*	51	2	433	4	101	0	477	1*
2	5*	10*	4*	52	2	435	4	102	0	477	1*
3	5*	15*	4*	53	2	437	4	103	1	478	1*
4	5*	20*	4*	54	7	444	4	104	0	478	1*
5	6*	26	4*	55	2	446	4	105	0	478	1*
6	8	34	5	56	0	446	4*	106	1	479	1*
7	2	36	5	57	2	448	4*	107	0	479	1*
8	7	43	5	58	3	451	4	108	0	479	1*
9	4	47	5	59	2	453	4	109	1	480	1*
10	2	49	5	60	7	460	4	110	0	480	1*
11	31	80	5	61	3	463	4	111	1	481	1*
12	4	84	5	62	0	463	4*				
13	24	108	5	63	1	464	4*				
14	49	157	5	64	0	464	4*				
15	14	171	5	65	1	465	4*				
16	12	183	5	66	0	465	3*				
17	8	191	5	67	0	465	3*				
18	9	200	5	68	1	466	3*				
19	4	204	5	69	1	467	3				
20	7	211	5	70	0	467	3*				
21	6	217	5	71	0	467	3*				
22	6	217	5	72	1	468	3*				
23	4	230	5	73	1	469	4				
24	4	234	5	74	0	469	4*				
25	2	236	5	75	0	469	4*				
26	4	240	5	76	0	469	4*				
27	3	243	5	77	1	470	4*				
28	9	252	6	78	2	472	2				
29	2	254	6	79	0	472	2*				
30	5	259	6	80	1	473	2*				
31	4	263	6	81	0	473	2*				
32	1	264	6	82	0	473	2*				
33	4	268	6	83	0	473	2*				
34	3	271	6	84	0	473	2*				
35	6	277	6	85	0	473	2*				
36	13	290	6	86	0	473	2*				
37	19	309	8	87	2	475	2*				
38	15	324	8	88	0	475	2*				
39	7	331	8	89	0	475	2*				
40	15	346	8	90	0	475	2*				
41	21	367	8	91	0	475	2*				
42	8	375	8	92	0	475	2*				
43	6	381	8	93	0	475	2*				
44	20	401	8	94	0	475	2*				
45	10	411	8	95	0	475	2*				
46	3	414	8	96	1	476	2*				
47	3	417	8	97	0	476	2*				
48	8	425	4	98	0	476	2*				
49	5	430	4	99	0	476	2*				
50	1	431	4	100	1	477	2*				

* Interpolated.

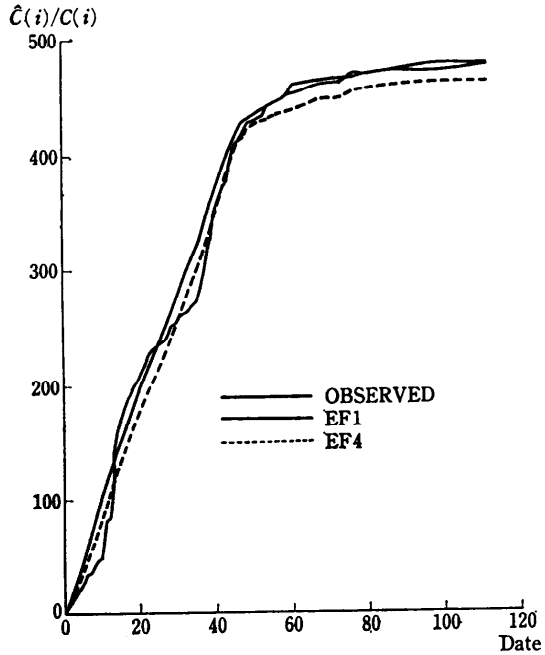

図-6 超幾何分布モデル

表-2 パラメータの推定

Parameters	Range	Increment	EF 1	EF 4
$\hat{m}$	450–600	3	597	600
$\hat{p}$	1.0–3.0	0.2	1.6	1.8
$\hat{u}_m$	20–45	3	29	29

のそれぞれの期間で超幾何分布を用いるのはよいとしても，それぞれの期間で個々にパラメータの値を推定したほうがよい．

また，対象とするプログラムがいくつかのモジュールからなり，それぞれのテストケースが一部のモジュールのみをテストするというような場合は，テストケースが及ぶプログラム上の領域をいくつか定め，それぞれの領域ごとに超幾何分布を適用したほうがよい．以上の拡張については文献10)を参照されたい．

超幾何分布モデルと他のいくつかの NHPP モデルとの相互関係が文献12)で論じられている．

その他のモデルについては文献13)が詳しい．

3. ソフトウェアのフォールトトレランス

ソフトウェアのフォールトを完全に取り除くことが困難であるなら，ソフトウェアをフォールトトレラントにすることが考えられる．これまでに，リカバリブロック法 (recovery block scheme) と N バージョンプログラミング (N-version Programming) が提案されているので紹介しよう．

3.1 リカバリブロック法[14)]

同一仕様に対する異なったバージョンを N 個用意し，それらをアクセプタンス（受容）テストのモジュールを介して図-7 のように接続する．これらのモジュールはできるだけ独立に作成する．

動作は，まず，第1のモジュールで処理を行う．結果をアクセプタンステストでチェックし，パスすれば次の動作に移る．パスしないときは，第2のバージョンが第1のモジュールの場合と同じ入力で処理を行う．この結果を再びアクセプタンステストでチェックする．パスすれば次の動作に，パスしなければ第3のモジュールに移る．以下アクセプタンステストをパスするまで同様のことを繰り返す．最後のバージョンでもアクセプタンステストをパスしなかったときに，初めて誤りの指示を行う．第1のバージョンで誤りが生じなかった場合には，その実行だけで済むのでオーバヘッドが少ない．

しかし，アクセプタンステストを作成する具体的な方法が示されていないので，現在までのところ，この実用化は進んでいない．

全体のソフトウェアシステムの中にどの程度の頻度でアクセプタンステストを挿入すればよいかという点も，実は，問題である．アクセプタンステストの挿入間隔が長すぎると誤る確率が高くなるし，逆に短すぎるとバージョンに対する制約が厳しくなり，バージョン間の独立性が阻害される．

3.2 N バージョンプログラミング[15)]

この方法でも，同一仕様に対して異なる N 個の

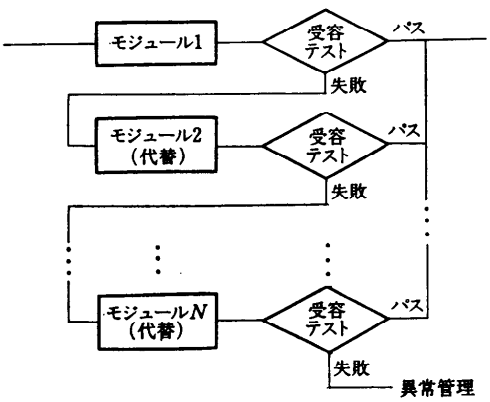

図-7 Recovery Block 法

バージョンを用意する．しかし，図-8 に示すように，ここでは同時にすべてのバージョンに処理を行わせ，それぞれの結果から最終的な出力を決定する．各バージョンの起動，バージョン間の同期などはドライバと称するシステムプログラムが制御する．ドライバは最終的な出力を決定する役割りも果たす．

最終的出力を決定する方法はいろいろありうるが，普通は多数決を用いる．したがっていくつかのバージョンが誤っても，多数が正しい出力を出せば，それらの誤りはマスクされる．

ドライバは単一系であるから，ドライバが誤るとそれを正すことはできない．しかし，ドライバの機能は一定でそう複雑ではないので，それを十分にデバッグしておくことはそうむずかしくはない．

表-3 に N バージョンプログラミングの実験結果の例を示す[16]．各バージョンは PL/1 の 600 行（ステートメント）以上からなるプログラムである．7 バージョンを用意し，それらから 12 個の三つ組（3 バージョンプログラミング）を作り，それぞれに 32 のテストケースを加えた．全体のテストケースは，したがって，12×32=384 個となる．

これらのうち，71 個のテストケースで，三つのうちの一つのバージョンに誤りが生じたが，59 ケースは正しい最終結果を与え，12 ケースは正しい最終結果を与えることができなかったという．これは，あるバージョンの誤りがシステム全体のアボートをひき起こしたからである．これは，3 バージョンプログラミングでも，単一のバージョンの誤りをいつもマスクするとは限らないということであり，注意を要する．

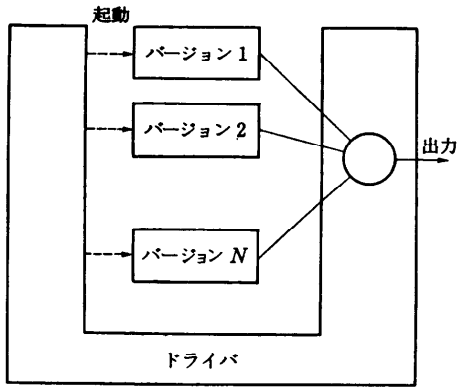

図-8 N-Version プログラミング

表-3 3-Version の実験結果

3 バージョン中で誤ったバージョンの数	テストケースの数	誤りマスク	
		成 功	失 敗
0	290	290	0
1	71	59	12
2	18	0	18
3	5	0	5

N バージョンプログラミングの実用化については，鉄道，原子炉制御，航空機操縦制御などの分野でいろいろ試みられている[17]．

3.3 相関する誤り

以上では，各バージョンは独立に誤ると仮定している．ところが，この仮定が期待したほど妥当ではないのではないか，という疑問が Knight ら[18]によって投げかけられた．

彼らは，27 個のバージョンを用意し，1,000,000 個のテストケースについて誤りの解析を行った．各バージョンが互いに独立に誤るとの前提の下で（バージョン V_i が 1 回のテストケースで誤る確率を p_i とする），n 個のテストケースのうち 2 個もしくはそれ以上のバージョンが同時に誤るテストケースの個数を X としたとき，

$$Z=\frac{X-nP_{more}}{\{nP_{more}(1-P_{more})\}^{1/2}} \tag{33}$$

ただし

$$P_{more}=1-P_0-P_1$$

$$P_0=(1-p_1)(1-p_2)\cdots(1-p_N)$$

$$P_1=P_0\left\{\frac{p_1}{1-p_1}+\frac{p_2}{1-p_2}+\cdots\frac{p_N}{1-p_N}\right\}$$

と変数変換すると，Z の確率密度関数は平均値 0，分散 1 の標準正規分布で近似される．

実験結果から，Z の観測値として

$$z=100.5$$

が得られた．ところで，標準正規分布に従う Z が z_0 もしくはそれ以下である確率を $\mathrm{Prob}(Z\leq z_0)$ で表すとすれば，

$$\mathrm{Prob}(Z\leq z_0)=0.99$$

となる z_0 は 2.33 である．これに比べると上の観測値 $z=100.5$ は異常に大きい．これは各バージョンが独立に誤るという仮定が適当でないことを推察させる．

バージョンの独立性といっても，その意味するところは実は曖昧である．Littlewood らは[19]，N バージョンの作成過程をモデル化し，二つのバージョンが同時に誤る確率がそれぞれ独立に誤ると仮定して得られ

る値より大きくなることがあることを理論的に示している．詳細は文献19)を参照されたい．

4. むすび

ソフトウェアの信頼性評価に関するいろいろなモデル，ソフトウェアのフォールトトレランス，誤りの相関性についての批判などについての最近の技術的動向を紹介した．

参考文献

1) 昭和57年度システムの高信頼性技術に関する調査研究（電子応用システム）成果報告書，pp. 20，日本電子部品信頼性センター（1983. 3）．
2) Clement, G. F. and Giloth, G. K.: Evolution of Fault-Tolerant Switching Systems in AT & T, Evolution of Fault-Tolerant Computing, ed. by Avizienis, A. et al., pp. 37-54, Springer-Verlag (1987).
3) Jelinski, Z. and Moranda, P. B.: Software Reliability Research, Statistical Computer Performance Evaluation, ed. by Freiberger, W., pp. 465-484, Academic Press (1972).
4) Dunham, J. R.: Software Errors in Experimental Systems having Ultra-Reliable Requirements, Digest of Papers, FTCS-16, pp. 158-163 (June 1986).
5) Littlewood, B. and Verrall, J. L.: A Bayesian Reliability Growth Model for Computer Software, J. Royal Statistics Society, C (Applied Statistics), 22, pp. 332-346, 1973 または A Bayesian Reliability Model with a Stochastically Monotone Failure Rate, IEEE Trans. Reliability, Vol. R-23, No. 2, pp. 108-114 (Feb. 1974).
6) Goel, A. L. and Okumoto, K.: Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures, IEEE Trans. Reliability, Vol. R-28, No. 3, pp. 206-211 (Aug. 1979).
7) Yamada, S., Ohba, M. and Osaki, S.: S-Shaped Software Reliability Growth Models and their Applications, IEEE Trans. Reliability, Vol. R-33, No. 4, pp. 289-292 (Oct. 1984).
8) Ohba, M. and Kajiyama, M.: Inflection S-Shaped Software Reliability Growth Model, Proc. WGSE Meeting, Vol. 28, IPS Japan (1983).
9) Yamada, S. and Osaki, S.: Nonhomogeneous Error Detection Rate Models for Software Reliability Growth, Stochastic Models in Reliability Theory, ed. by Osaki, S. and Hatoyama, Y., pp. 120-143, Springer-Verlag, Berlin (1984).
10) Tohma, Y., Tokunaga, K., Nagase, S. and Murata, Y.: Structural Approach to the Estimation of the Number of Residual Software Faults based on the Hyper-Geometric Distribution, IEEE Trans. Software Engg., Vol. 15, No. 3, pp. 345-355 (Mar. 1989).
11) Frey, T. C.: Probability and its Engineering Uses, 2nd Ed., New York, Van Nostrand, pp. 205 (1965).
12) Tohma, Y., Jacoby, R., Murata, Y. and Yamamoto, M.: Hyper-Geometric Distribution Model to Estimate the Number of Residual Software Faults, Proc. COMPSAC-89, Orlando (Sep. 1989).
13) 宮本 勲：ソフトウエアエンジニャリング：現状と展望，TBS 出版会（1982）．
14) Randell, B.: System Structure for Software Fault Tolerance, IEEE Trans. Software Engineering, Vol. SE-1, No. 1, pp. 220-232 (June 1975).
15) Avizienis, A. and Chen, L.: On the Implementation of N-Version Programming for Software Fault Tolerance during Execution, Proc. COMPSAC, pp. 149-155 (Nov. 1977).
16) Chen, L. and Avizienis, A.: N-Version Programming: A Fault Tolerance Approach to Reliability of Software Operation, Digest of Papers, FTCS-8, pp. 3-9 (June 1978).
17) Voges, U. ed.: Software Diversity in Computerized Control Systems, Springer-Verlag/Wien (1988).
18) Knight, J. C., Leveson, N. G. and St. Jean, L. D.: A Large Scale Experiment in N-Version Programming, Digest of Papers, FTCS-15, pp. 135-139 (June 1985).
19) Littlewood, B. and Miller, D. R.: A Conceptual Model of Multi-Version Software, Digest of Papers, FTCS-17, pp. 150-155 (June 1987).

（平成元年10月12日受付）