解　説

ソフトウェア信頼性モデル入門†

大　場　　充††

1. はじめに

ソフトウェアが，社会のいろいろな分野に利用されるようになるにつれて，その信頼性に対する社会的な関心が増大している．特に，ソフトウェアに潜在するエラーによる障害の影響が広範囲にわたる，基本ソフトウェアや社会性の高いアプリケーション（鉄道，電話，航空など）の分野では，1970年代の初頭から，ソフトウェアの信頼性に関する研究が，米国を中心に行われてきている．

ソフトウェアのエラーを少なくして，その信頼性品質を向上させるためには，次の三つの技術が必要になる．

1) 信頼性実現技術，
2) 検証・分析技術，
3) 信頼性評価技術．

信頼性実現技術は，信頼性の高いソフトウェアを開発するためのプロセスや方法論などを言う．具体的には，クリーンルーム・プロセス，Nバージョン・プログラミング，リカバリ・ブロック，チェックポイント・リスタート，形式的方法などが含まれる．

検証・分析技術は，開発中のソフトウェアが信頼性の高いものであることを実証するためのプロセスや方法論などを言う．具体的には，設計や実現が正しく完全であることを証明する方法，FMEA (Failure Mode and Effects Analysis) を用いたインスペクションやレビューのやり方と方法，テストのプロセスやテストケース設計の方法などが含まれる．

信頼性評価技術は，開発中のソフトウェアや開発されたソフトウェアの信頼性がユーザの要求水準を満足しているかどうかを評価したり，異なるソフトウェアの信頼性を比較検討するためのプロセスや方法論を言う．具体的には，ソフトウェアに潜在するエラーの総数を予測する方法や，ソフトウェアの故障間隔を推定する方法などが含まれる．

本稿では，特にソフトウェア信頼性評価技術の重要な要素としてのソフトウェア信頼性モデルについて，できるだけ信頼性理論の概念や術語を使わずに，簡単なアナロジで解説する．

2. ソフトウェアの信頼性とは

一般に，**信頼性**の概念は，ある構造物がどの程度の確率で失敗なしにユーザの利用目的を達成できるか，どの程度の確率でユーザの利用したいときに利用できるか，失敗した場合どの程度の確率でユーザに致命的な打撃を与えるかなどについての特性を，総合的にとらえたものである．

これに対して，**信頼度**は，あるシステムが指定された時間内において，その意図されている機能の遂行に失敗することなく動作する確率，と定義できる．したがって，信頼度の概念は，信頼性の部分概念であり，機能遂行の成功（失敗）確率と時間との関係を説明する概念である．

よく知られているように，ハードウェアでは，物理的な経時劣化のため，システムはほぼ一定の確率で故障する．すなわち，同じ入力に対して，出力が生成されるのは，一定の確率分布に従い確実ではない．これを時間軸にそって観察していると，ほぼ一定な時間間隔分布にしたがって（ランダムに）故障が発生していることになる．

ソフトウェアは，物理的な構造物でないため，経時的な劣化が生じない．すなわち，同じ入力列に対して確実に同じ出力列を生成するため，現象は確率的ではない．ソフトウェアに故障が観測される場合は，与えられた入力列に対して，『期待されている機能』が実行されずに，『異なる機能』が実行されたときである．これを時間軸にそって観察していると，誤動作がある時間間隔で発生しているように見えることもある．

このような現象は，ソフトウェアに設計や実現上の

† Software Reliability Models by Mitsuru OHBA (IBM Japan, Tokyo Research Laboratory).

†† 日本アイ・ビー・エム(株)東京基礎研究所

誤り（エラー）があり，与えられた入力の処理の過程で，その誤りのある部分（フォールト）が実行されたときに観測される．したがって，**ソフトウェア信頼度**は，その入力列の分布と，潜在するエラーの個数と分布の組合せで決まる．

以上のような考察から，**ソフトウェア信頼度の実現レベル**を評価するためには，次の二つの立場があることが理解できる．第1は，故障の現象としての発生間隔でソフトウェア信頼度の実現レベルを評価する立場である．第2は，故障の原因である潜在エラーの個数でソフトウェア信頼度の実現レベルを評価する立場である．

ソフトウェアの稼働中に故障が発見されると，その直接の原因であるエラーが解析され，その結果としてフォールト（バグ）は除去される．このエラー除去によって，ソフトウェアに潜在するエラーの個数が減少する．これが故障の発生確率を小さくするため，故障の発生間隔はエラーが取り除かれてゆくに従って，長くなってゆく．この稼働時間と故障の発生間隔（MTTF, Mean Time To Failure）の関係を**図-1**に示す．

同じ現象を，稼働時間と発見され除去されるエラーの総数（累積数）の関係に注目して観察すると，**図-2**に示されるような時間変化が得られる．このような信頼度の経時的な変化を，**信頼度成長**と呼ぶ．ソフトウェアの信頼性評価においては，この信頼度成長を分析して，その将来を予測することが必要になる．

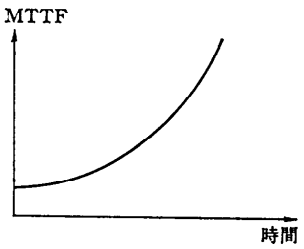

図-1　MTTF の時間変化

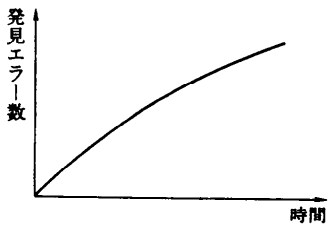

図-2　発見エラー総数の時間変化

3. ソフトウェア信頼性モデルの発展

ソフトウェアの信頼性に関する実務的な研究の歴史はかなり古く，1960年代の後半からさまざまな取組がなされてきていた．それらの多くは，ハードウェアの信頼性理論をソフトウェアに応用しようとしたものが多く，成功しなかった．

ソフトウェアの信頼性評価に関する最初の理論的考察は，Jelinski と Moranda および Shooman によって行われた．特に Jelinski と Moranda は，ソフトウェアの信頼度成長を，ソフトウェアに潜在するエラーの総数と稼働時間の関係から説明するモデルを，1972 年に発表した[1]．

Schick と Wolverton は，Jelinski と Moranda のモデルではエラーの発見率が時間に対して一定であることを仮定していることに注目し，時間とともにエラー発見率の増加するモデルを提案した．これは，テストなどでは，一般にあるエラーが発見されてから，時間が経てば経つほど，次の（新しい）エラーが発見されやすいことに着目して作られたモデルである．

ベル研究所の Musa は，Jelinski と Moranda のモデルを発展させ，テスト時におけるソフトウェア信頼度成長と，実稼働時のソフトウェア信頼度とを関連付けた実行時間モデルを提案した．これによって，Musa は信頼度成長を CPU 時間に基づいて分析すべきであると主張した[3]．

Goel と Okumoto は，Musa や Jelinski-Moranda と同じ仮定から，Musa の実行時間モデルを簡単化したモデルを導出し，それをハードウェアの信頼度成長予測でよく応用されている**非定常ポアソン過程**の平均値関数とすることにより，モデルのパラメータを簡単に推定する方法を示した[2]．

英国においては，Littlewood が**ベイズ統計学**を応用し，Jelinski-Moranda のモデルに，別の解釈を与えている．その後，Littlewood は Jelinski-Moranda のモデルの仮定を修正して，いくつかの新しいモデルを提案している．これらのモデルは，主にソフトウェア故障の発生間隔（MTTF）を推定するモデルである[4]．

スウェーデンでは，産業界を中心に，非定常ポアソン過程に基づくモデルの，実際のプロジェクトへの適用が研究されている．

フランスやイタリアでも，近年プロジェクトへの適用を念頭に置いたいくつかの試みが，非定常ポアソン

過程に基づくモデルや Littlewood のモデルを中心として行われている.

我が国においては,1970年代の前半から**ロジスチック曲線**や**ゴンペルツ曲線**を応用した決定論的モデルのパラメータを最小2乗法で推定する方法が,品質管理の方法として研究された.これは,ソフトウェアに残存するエラーの個数を推定するためのモデルである.

その後,伊土らや大場は,**捕獲・再捕獲法**によるサンプリングの方法を改善し,残存エラー数を精度よく推定する方法を提案した.これらの方法は,現実のプロジェクトにも適用され,その効果が示された.捕獲・再捕獲法に基づく方法が実際のプロジェクトに適用された例は世界的にも少なく,我が国に特有であった[5].

1980年代の前半から,我が国においても,確率論的なモデルが研究されている.特に,Goel-Okumoto のモデルの仮定をより現実的なものに拡張した,非定常ポアソン過程のモデルについての研究が知られている.山田と尾崎は,Goel-Okumoto のモデルを拡張して,現実によく観測されるS字型の成長曲線をもつ,**遅れS字型ソフトウェア信頼度成長モデル**を提案した[6].

大場は,Goel-Okumoto のモデルのソフトウェア・エラーの独立性に関する仮定に着目し,ソフトウェア・エラーに部分的な順序関係を仮定した,**加速S字型ソフトウェア信頼度成長モデル**を提案し,ロジクチック曲線がその特殊型であることを示した[7].

大寺らは,Goel-Okumoto のモデルで仮定されている時間の定常性に着目し,テストのワークロード分布を考慮したモデルの拡張を行った.また,テスト時間とテスト・カバレージとの関係に着目した非定常ポアソン過程モデルが,従来の Goel-Okumoto のモデルや,遅れS字型ソフトウェア信頼度成長モデルと同型になることを示した[10].

当麻らは,テスト・カバレージとテスト・ラン,テスト・ランとテスト時間の複合的な関係を,超幾何分布を用いてモデル化した.この**超幾何分布モデル**は,テストにおけるエラー発見過程についての解釈を与えている点で興味深い[8].

大場と周は,ソフトウェア信頼度成長モデルの基本的な仮定である,エラー除去作業の完全性に注目し,**不完全デバグ仮定**に基づき,Jelinski-Moranda モデル,Schick-Wolverton モデル,Goel-Okumoto モデル,Littlewood のバグ計数モデルを再構築し,それらが完全デバグの仮定に基づく従来のモデルと同型になることを示した[9].

三道は,ハードウェアの受入れ検査でよく適用されているベイズ統計学の方法を応用し,**ソフトウェアの受入れ検査**を合理的に行う方法を提案した.

4. ソフトウェア信頼度モデルの基礎

4.1 地雷原のアナロジ

このアナロジは,**Jelinski-Moranda のモデル**を中心とした,ソフトウェア信頼度成長の基本的なモデルの多くが,この立場に立っているという意味で,最も重要な考え方である.パラメータの推定法として,それぞれ異なったモデル化が行われているが,基本的な立場は同じである.たとえば,**Musa モデル**,**Goel-Okumoto モデル**,初期の **Littlewood モデル**などが,このアナロジに基づいている.

今,地雷原を兵隊が渡ろうとしている.兵隊は,一人ずつ順番に地雷原に入り,ランダムに方向を変えながら,地雷原の中を敵陣に向かって進むものとする.この場合,最初の兵隊が地雷原に足を踏み入れる瞬間には,地雷原には数多くの地雷が埋められているため,その兵隊が地雷原の中を歩き続けられる確率は,時間とともに急激に小さくなる.

最初の兵隊が,地雷の一つに触れて,爆発が起こると,地雷原の状態が変化する.地雷は,一度爆発すると,二度と爆発しない.したがって,2番目の兵隊が,一定時間内に地雷に触れる確率は,1番目の兵隊よりも少しだけ小さくなる.

このようにして,兵隊をどんどん地雷原に投入してゆくと,兵隊も失うが,地雷の数も減ってゆく.地雷の数がある程度少なくなると,今度は逆に,一定時間内に兵隊が地雷に触れる確率は,非常に小さくなる.しかし地雷は完全になくなっているわけではないので,時々運の悪い兵隊が地雷に触れる.

ここで,『地雷原』を『ソフトウェア』に,『地雷』を『潜在するエラー』に,『兵隊』を『テストケース』または『プログラムの実行』に読み替えれば,ソフトウェアの信頼度成長を説明していることになる.重要なことは,このアナロジに仮定されているいくつかの条件である.

地雷原のアナロジでは,つぎのことが仮定されている.

1) 兵隊が地雷に触れる確率は,地雷原に残っている地雷の数に比例する.

2) 兵隊はランダムに方向を変えながら地雷原の中を進む.(定数比例の仮定)

3) 地雷は,一度爆発すれば,二度と爆発しない.

第1の仮定は,地雷原に関して言えば,合理的な仮定である.第2の仮定は,現実的ではないが,解析なモデルを構築するためには,重要な仮定である.この仮定は,兵隊がある瞬間に地雷に接触しているかいないかが,一定の割合で決まることを意味している.第3の仮定も,地雷に関するかぎり,妥当な仮定である.

この地雷原のアナロジが,ソフトウェアに対して妥当なアナロジであるかどうかは,アナロジ自信の正当性とは独立の問題である.特に,ソフトウェアでは,第3の仮定は,妥当でない.これは,ソフトウェアの世界で解釈すると,デバグが完全であることを仮定していることと等価である.この仮定は,現実的でない.

第2の仮定も,ソフトウェアの世界で解釈すると,エラー相互間に関係がなく,独立であることを仮定していることになる.ソフトウェアに残存するエラーが少ない場合には,この仮定は一般に妥当である.しかしソフトウェアに残存するエラーが多い場合には,一部のエラーは,それらと同じ実行経路上でそれらの前に存在するエラーが取り除かれないかぎり,表面化することはない.すなわち,エラー相互間には順序関係がある場合がある.

このエラーに発見の順序関係がある状態を,地雷原のアナロジで言うと,地雷原に道があり,兵隊は道の上を進むことと同じになる.もちろん,地雷は道に沿って埋められる.兵隊にランダムな動きが許されている場合には,最初の兵隊でも,地雷に触れることなく敵陣にたどりつける確率は,非常に小さいが,ゼロではなかった.しかし,道の上を歩く場合にはもっと確定的で,最初の何人かの兵隊は,ある経路上に地雷がなくなるまで,確実に地雷に触れる.

4.2 クジラの固体数推定のアナロジ

ソフトウェアに多くのエラーが残っている場合には,地雷原のアナロジも有効かも知れない.しかし,残っている地雷の数が少なくなると,よほど運の悪い兵隊がいないかぎり,残りの地雷を全て爆発させてしまうことはできない.つまり,非常にエラーの少ないソフトウェアを開発しようとすると,地雷原のアナロジに基づくモデルでの評価では,不十分になる.

非常にエラーの少ないソフトウェアをテストして,実際にエラーがほとんどないことを実証しようとすることは,広い海で数の少ないクジラの数を,できるだけ正確に知ろうとすることに等しい.このような意図で提案されたのが,**エラー埋込み法**または**捕獲・再捕獲法**である.

今,ある広い海域に生存するクジラの固体数を,ある程度の誤差で推定しなければならないとする.対象となるクジラの固体数は,あまり多くない.このような場合に実際に行われる方法は,まず一定数のクジラをその海域で捕獲する.捕獲したクジラは,マークが付けられた後,再びその海域に放たれる.

その後,一定期間が経過してからもう一度同じようにして,その海域でクジラを捕獲する.今度は,マークの付いているクジラが一定数(マークを付けた数より十分少ない)に達するまで,捕獲を続ける.そして,捕獲したクジラをマークの付いているものと,付いていないものに分類し,その比からクジラの総数を推定する.

200頭のクジラを最初に捕獲しマークを付けたとする.今,マークの付いているクジラを5頭と,マークの付いていないクジラを95頭,2回目の調査で捕獲したとすれば,その海域に生息するクジラの固体数は,

$$\frac{\frac{1}{5}}{\frac{200}{}} \times (95+5) = 4000,$$

と推定される.

ここで,クジラを潜在エラーと読み替え,海域をソフトウェアと読み替えると,ソフトウェアに潜在するエラーの推定が可能になる.テストを念頭におくと,本番のテストを実施する前に,いくつかのエラーを発見しておき,それらを記録した後修正せずに本番のテストを実施する.

こうすると,本番のテストでは,記録にされている既知のエラー(マークの付いたクジラに対応する)と,未知のエラー(マークの付いていないクジラに対応する)が発見される.この2種類のエラーの数の比から,ソフトウェアに潜在するエラー数を推定できる.

この捕獲・再捕獲法は,埋め込まれた(マークされた)エラーが,真のエラーに(その発見の困難さの分布が)似ている場合,精度の高い推定値が得られる.これは,クジラのアナロジで言えば,同じ海域に2種類の異なるクジラが生息しており,マークを付けたときに捕獲したクジラはその一方だけで,2度目の調査

で捕獲したクジラはその両方を数えた場合に，正しい推定値が得られないことに等しい．つまり，エラーの分類が精密でないとき，精度は悪くなる．

4.3 人口予測のアナロジ

ソフトウェアが大規模で，テストの開始時点で数多くのエラーが残存している場合のエラー発見過程を近似的に記述するための考え方として，このアナロジは興味深い．一見するとソフトウェアとの直接的な関連が付きにくいが，現実にはよく適合する場合がある．

今，人間の住んでいなかった広大な島に，突然かなりの数の古代人が漂着したとする．その人数はたまたま種の保存に必要最小限であったとし，また島には食物の原料が豊富であったとする．この快適な島で彼らが生活を始めると，少しずつ人口は増え始め，何代か経過するとかなりの人口になる．

人口が一定数を超えると，人口は加速度的に増加し始めるが，今度は食物の供給がその人口増加に追いつかなくなり，食料不足のために人口増加率が徐々に低下してゆく．人口増加があまりに急速な場合には，飢きんが発生し，逆に人口が減ってしまう．適度な人口増加率が保たれている場合には，ゆっくりとその島に適正と思われる人口に向かって収束してゆく．

このような人口増加の過程をモデル化したものが，**ロジスチック曲線**である．ロジスチック曲線による人口増加の過程をグラフに示すと，**図-3** のようになる．この曲線の形状は，テスト中に発見されるエラー数の増加曲線（信頼度成長曲線）に似ている．その場合，ある時点での人口をテストのある時点までに発見されたエラーの総数，ある島に適した最大人口をテストで発見可能なエラーの総数と読み替える．

人口増加のアナロジは，ナンセンスのように見える．しかし以下のような解釈も可能である．最初に島に漂着した古代人の数をテスト開始時点ですでに発見可能なエラーの総数とする．彼らは子孫を産む．すなわち，最初のエラーが発見された後に発見可能になるエラーがあり，さらにそれらの第2代目のエラーが取り除かれた後に発見可能になるエラーがある．

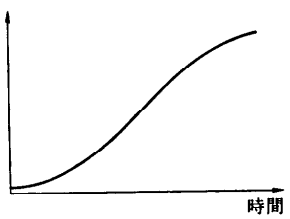

図-3 ロジスチック曲線

このようにエラーが階層的に順序化されていたとする．また，エラーの発見のしやすさ（人口増加）は，ソフトウェアに残存するエラーの総数（島で再生産される食料の量に見合った最大人口との差）に比例するとする．これは地雷原のモデルで，地雷が道に沿って埋められていることと等価である．ソフトウェアに残存するエラーが多く，それらの間に発見の順序関係がある場合には，このアナロジが適用可能である．

4.4 テスト過程のモデル化

ソフトウェアの信頼度成長をモデル化する場合のアプローチとしては，エラーの発見過程をブラックボックスとして，その外部からの観測をモデル化するものと，エラー発見過程を直接モデル化するものがある．これまでに述べてきたものは，全て前者に属する．ここでは，後者のアプローチについて簡単に説明する．

このアプローチで基本になるのは，テストのカバレージとその時間的推移である．今，テストケースを実行すると，それはプログラムのある実行経路に沿って処理され，プログラムになんらかの状態変化を発生させる．この状態の変化が仕様によって記述されたものであるかどうかで，テストケースが成功か失敗かに分類される．

プログラムの部分にフォールトがあると，その部分を実行経路に含むか含まないかによって，テストケースが失敗するか成功するかが決まる．プログラムをある単位に分割し，テストケースの実行経路に含まれた分割単位の集合を，テストケースによって被覆された部分とよぶ．特に，プログラム全体に対する被覆された部分の割合をカバレージとよぶ．

今，ある方法によってプログラムを M 個の単位に分割したとする．また，一つのテストケースを実行して被覆できる分割単位の平均の個数を s 個，それまでに被覆されている分割単位の個数を m 個とする．このような条件で新しいテストケースを実行したとき，新たに k 個のそれまでのテストケースでは被覆されていない分割単位を被覆できる確率 $p(k|m)$ は，**超幾何分布**によって，以下のように与えられる．

$$p(k|m)=\frac{\binom{M-m}{k}\binom{m}{s-k}}{\binom{M}{s}}$$

すなわち，カバレージがゼロである最初のテストケースでは，s 個の分割単位が被覆される確率が最も大きい．次のテストケースでは，すでに s 個の分割単

位が被覆されてしまっているので，新しく被覆される分割単位の個数は，s よりも小さくなる確率が高い．このようにして，新しく被覆される分割単位の個数の期待値は，急激に小さくなってゆく．このアナロジに基づいているのが**超幾何分布モデル**である．

ここで，プログラムに潜在するエラーが分割単位にまたがって一様に分布していると仮定すれば，一定時間のテストで発見されるエラーの数は，テストの初期に多く，時間とともに減少する．これはまさに，地雷原のアナロジと等価である．

特に，一つのテストケースの実行にかかる時間がほぼ一定であることを仮定すれば，テストの時間経過と発見されるエラーの総数との関係は，図-2 に示されるような曲線になる．これは，Musa や Goel-Okumoto のモデルと同型である．

テストケースの実行に必要な時間の分布や，プログラムに潜在するエラーの分布を，より現実的なものに変えることによって，より現実的なモデルを構築できる．また，一度被覆されただけでは，エラーを捕捉できない場合があることを仮定すると，地雷原のアナロジに基づく Goel-Okumoto のモデルを改善した遅れS字型ソフトウェア信頼度成長モデルと同型のモデルが得られる．

4.5 ハードウェアのアナロジ

ソフトウェアのテスト過程をブラックボックスとみて，かつその現象だけに注目すると，ハードウェアの MTTF（平均故障間隔）を推定する方法が，そのまま応用できる．この立場に立つモデルでは，残存エラー数の推定はできない．そのようなモデルとして，Musa と Okumoto による**対数実行時間モデル** や **Littlewood のモデル**が知られている．

ここでは，エラーの現象として現れるソフトウェアの故障に注目し，その発生時間間隔を推定することが関心事である．エラーは，一度発見されると，除去されるので2度と発見されることはない．しかし，現象としての故障は，同じエラーを原因として，無限回観測される可能性がある．

したがって，ハードウェアのアナロジでは，エラーの個数と故障発見の頻度（または確率）との対応は考えない．このため，モデルのパラメータに対する物理的な意味付けは，ほとんどの場合できない．これが，この方法の限界であり，問題点である．

最近，ベル研究所で行われた方法では，テスト中エラーの修正を行わず，一定期間中に目標とする MTTF が達成されたかどうかを評価し，達成していればテストを終了する．達成されていない場合には，記録されているエラーを修正し，また同じテストを実施するという，まさにハードウェアと同じものが試行された．この場合，故障の発生確率分布は，ハードウェアと同じく，テスト期間中一定である．

ソフトウェア工学の立場から考えると，意味付けのできないパラメータの値を決めて，将来の MTTF を推定することは，単なる曲線の当てはめとしか思えない．しかし，ソフトウェアのユーザの立場に立つと，残存するエラーの数よりも，ソフトウェアが何時間稼働し続けるかのほうが重要である．そのような意味で，ソフトウェア故障現象のモデルは，必要である．

5. ソフトウェア信頼度モデルの応用

ここでは，ソフトウェア信頼性モデルが，ソフトウェアの開発現場でどのように利用できるかの例を示す．データからモデルのパラメータを推定する方法については，それぞれのモデルに関する論文を参照されたい．

5.1 残存エラー数の推定

不特定多数のインストレーションを仮定した，オペレーティング・システムなどの基本ソフトウェア開発では，その最終段階で開発されたソフトウェアの保守が，準備された資源で十分に可能かどうかを判定する必要がある．もし，保守の体制が不備であれば，ユーザの保守要求に答えられなくなるからである．

ユーザからの保守要求が，ソフトウェアに残存するエラーの数と，そのユーザ数に比例することは，経験的に知られている．また，そのことはソフトウェアの信頼度成長モデルの考え方にも合致している．そこで，問題となるのは，テスト期間中のエラー発見の履

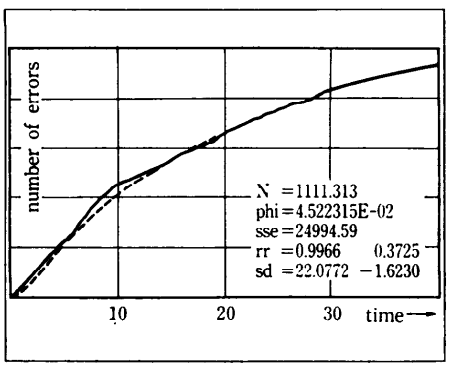

図-4 発見エラー数の推移

歴から，残存エラー数を予測することである．

ここでは，Goel-Okumoto のモデルを適用して，オペレーティング・システムの拡張部分に対するシステム・テストの最終段階での，残存するエラー数推定の例を示す．図-4 にテスト期間中に発見されたエラーの累積数と，モデルを用いて推定された信頼度成長曲線を示す．モデルによって推定された残存エラー数（不完全デバグによって保守工程で新たに混入するエラーを含む）は，約240個であった．

この残存エラー数をそのソフトウェアの大きさで割ると，1 KLOC 当たりのエラー数で約1個となり，満足できる品質であった．単純にそのコードのライフを20年と仮定し，単純平均をとれば，1年間に発見されるエラーは 1 KLOC 当たり約 0.05 個となる．これは基準を満足していた．

5.2 MTTF の推定

特定のインストレーションにおいて，ソフトウェアがどの程度の時間，故障なしに稼働し続けるかを知ることは，ソフトウェアのユーザにとって重要なことである．もし故障の頻度が高く，作業の途中で故障が起きてしまうと，ユーザの作業を中断させてしまう．

たとえば，銀行のオンライン業務中に故障が発生すると，銀行内のソフトウェア・ユーザだけでなく，銀行のお客にまでその影響が及ぶ．また，飛行機の制御をしているソフトウェアが飛行中に誤動作をすると，飛行機が墜落して，乗客に多大な被害を与えてしまう．

そのような場合，あらかじめソフトウェアがどの程度の MTTF をもっているかを知ることが重要になる．もし，非常に高い確信度で MTTF が2時間であると分かっていれば，そのようなソフトウェアによって制御される飛行機を，飛行時間の上限を1時間に設定して，運用することも可能になるからである．

ここでは，軍用ソフトウェアの MTTF を推定した例を示す．表-1 にテストで観測された MTTF データの一部を示す．Jelinski-Moranda モデルを用いてこのデータを解析した結果，テスト開始時にソフトウェアに残存していたエラーの推定値として 31.2 個，エラー発見率として 0.0069 を得た．

したがって，Jelinski-Moranda のモデルに基づけば，$i-1$ 個目のエラーが発見されてから i 個目のエラーが発見されるまでの故障率は，

$$h(i)=0.0069\ (31.2-i+1),$$

で与えられる．この逆数が MTTF の期待値になるので，27.9 が得られる．この値が，信頼性の要求水準を上回っていれば，テストを終了することができる．

表-1 ソフトウェア故障間隔データ

番 号	故障間隔
1	9
2	12
3	11
4	4
5	7
6	2
7	5
8	8
9	5
10	7
⋮	⋮
24	91
25	2
26	1

この例の場合 27 番目の故障が観測されたのは，87 単位時間後であった．同様にして，28 番目の故障までの期待値は 34.5 であるのに対して，実測値は 47 単位時間であった．

6. ま と め

ソフトウェア信頼性を評価するためのモデルの考え方と，簡単な応用例について説明した．モデル化については，その考え方が一つで，固定したものではなく，いくつかの立場があることを示した．この立場は，パラメータの推定のためにどのような統計的方法を適用するかとも強い関係がある．

最も基本的なモデル化を考えると，どのようなアプローチをとっても，ほぼ似たような結果が得られる．その意味で，Jelinski-Moranda や Goel-Okumoto のモデルは，現場での応用を考えると重要である．

モデル化や，モデルの選択で重要なことは，モデルの合理的な解釈がソフトウェアの世界で可能かどうかである．ソフトウェア信頼度の推定は，単なる曲線の当てはめであってはならない．その意味では今後，解析的なモデルだけでなく，コンピュータ・シミュレーションに基づいた，より現実的なモデルの研究が必要である．

参 考 文 献

1) Jelinski, Z. and Moranda, P.: Software Reliability Research (Statistical Computer Performance Evaluation) Freiberger, W. Ed., Academic Press, Inc., New York (1972).

2) Goel, A. L. and Okumoto, K.: Time-Dependent Error Detection Rate Model for Software Reliability and Other Performance Measures, IEEE Trans. Reliability R-28, pp. 206-211 (1979).
3) Musa, J. D.: The Measurement and Management of Software Reliability, Proc. IEEE, Vol. 68, No. 9, pp. 1131-1143 (1980).
4) Littlewood, B.: Theories of Software Reliability: How Good Are They and How Can They Be Improved?, IEEE Trans. Software Eng. SE-6, No. 9, pp. 489-500 (1980).
5) Ohba, M.: Software Quality=Test Accuracy x Test Coverage, Proc. 6th ICSE, Tokyo, pp. 287-293 (1982).
6) Yamada, S., Ohba, M. and Osaki, S.: S-Shaped Reliability Growth Modeling for Software Error Detection, IEEE Trans. Reliability R-32, pp. 475-478 (1983).
7) Ohba, M.: Software Reliability Analysis Models, IBM J. Res. & Dev., Vol. 28, pp. 428-443 (1984).
8) Tohma, Y. et al.: Structural Approach to the Estimation of the Number of Residual Software Faults Based on the Hyper-Geometric Distribution, IEEE Software Eng. SE-15, No. 3, pp. 345-355 (1989).
9) Ohba, M. and Chou, X-M.: Does Imperfect Debugging Affect Software Reliability Growth?, Proc. 11 th ICSE, Pittsburgh, pp. 237-244 (1989).
10) Ohtera, H., Yamada, S. and Ohba, M.: Software Reliability Growth Model with Testing-Domain and Comparisons of Goodness-of-Fit, Proc. International Symp. on Reliability and Maintainability, Tokyo, pp. 289-294 (1990).

(平成2年9月13日受付)