解　説

ソフトウェアの品質評価に関する考え方と動向

ソフトウェア信頼度成長モデルに基づく定量的品質評価法†

山　田　茂††

1. まえがき

現代のような情報ネットワーク社会あるいは高度情報化社会では，コンピュータシステムの故障や障害により長時間にわたるシステムダウンが続くと，社会生活に及ぼす影響は計り知れないものとなっている．その原因の多くは，システムの構成要素のうち大規模化・複雑化・多様化したコンピュータソフトウェアに関係し，その開発プロセスで潜入した人為的誤りや欠陥によるものが顕著になってきた．これは，ソフトウェア開発には多人数の要員が参画して，知的生産物である文書（ドキュメント）とソースプログラムを成果物とする人間集約的作業であるので，回避することのできない問題である．そこで，ソフトウェア開発における生産性とともに品質を高めるために，品質・コスト・納期を考慮したうえで開発作業をすすめ，開発プロセスとリリース後の運用・保守までのライフサイクル全体を管理していこうというソフトウェア工学の考え方が強調されている．特に，ソフトウェア生産技術の中でも品質管理技術として，ソフトウェアに関する障害が発生することなくコンピュータが正常に動作を続行するという品質特性である，ソフトウェア信頼性を高める研究が重視されている．

ソフトウェアの開発管理においては，そのライフサイクルに基づく品質・コスト・納期を計量化し，それぞれ品質管理・費用管理・工程管理という活動の中でいかに分析・評価して検討を加えるかということが重要である．本稿では，ソフトウェアの品質向上が生産性向上に寄与する最大の誘因であるとして，まずソフトウェア品質の計量化と管理法について述べる．次に，充足されていて当り前であり，不十分であると不満足感を与えるという，「当り前品質」としてのソフトウェア信頼性を最も重要な品質特性と考えて，その基本的概念を整理・検討する．同時に，その定量的評価技術であるソフトウェア信頼度成長モデルと，そのうち最近特に有望視されている NHPP モデルを考察する．さらに，NHPP モデルに基づくソフトウェア信頼性評価の現状と問題点を述べたうえで，実際のソフトウェア信頼性評価のツールと実施例も示す．

2. ソフトウェア品質の計量化と管理法

一般に生産管理とは，製品（プロダクト）を作り出すプロセスの管理であり，品質管理，工程管理，費用管理，および要員管理といった要素から構成されている[1]．ソフトウェア開発についても，ソフトウェア工学に基づく近代的な生産管理の考え方が必要となってきている．ソフトウェアの生産管理あるいは開発管理における重要な問題は，ソフトウェアのライフサイクルに基づく品質，納期，コストを計量化し，それぞれ品質管理，工程管理，費用管理という管理活動の中でいかに分析・評価して検討を加えるかということである．**図-1** は，ソフトウェアの開発プロセスにおける開発管理の考え方を示している．

これらの管理活動を効果的かつ効率的に推進していくためには，**管理のステップ**と**標準化**の考え方が重要である．前者は，**図-2** に示すような計画 (Plan)，実行 (Do)，確認 (Check)，改善 (Action) という，PDCA のサイクルを繰り返す管理の実践・維持の手順である．後者は，ソフトウェアの開発作業の均質化を計るものであり，個人の能力に頼る職人的な手工業生産からソフトウェア工学的な工業製品の生産へと脱皮するために不可欠である．ソフトウェアの開発プロセスにおいて，適切

† Principles and Current Views in Software Quality Assessment: Quantitative Software Quality Assessment Method Based on Software Reliability Growth Models by Shigeru YAMADA (Department of Industrial and Systems Engineering, Faculty of Engineering, Hiroshima University).

†† 広島大学工学部第二類（電気系）

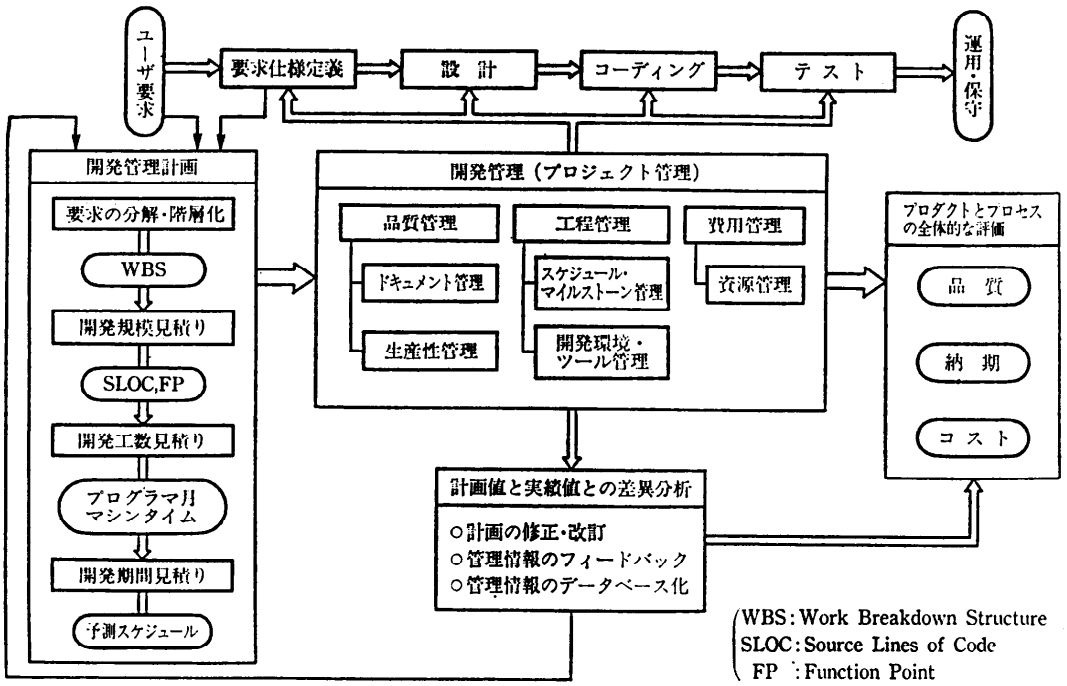


図-1　ソフトウェア開発管理の考え方

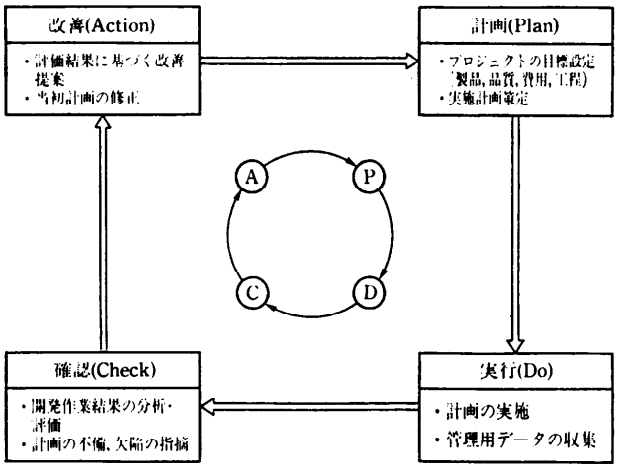


図-2　ソフトウェア開発管理に対する管理のサイクル

なソフトウェア生産技術とソフトウェアツールを適用したうえで，上述の管理活動の機能を管理のサイクルと標準化により支援することによって，ソフトウェア工学が目的とするソフトウェアの生産性と品質の向上が計れることになる[2]．ここで，生産性と品質の関係は，「品質は開発プロセスで作り込め」と言われることに象徴されるように，品質向上が生産性向上に寄与する最大の誘因であるというものである．

ソフトウェアの品質を向上させるためには，開発プロセスの全工程で人為的誤りや欠陥などのソフトウェアエラーが潜入しないように適切な生産技術を適用するとともに，潜入したソフトウェアエラーはできるだけ早期検出するように組織的に管理していく**総合的品質管理**（Total Quality Control，TQC と略される）の考え方が重要である．そこで，ソフトウェアの品質特性の定義および計量化が必要となる．ソフトウェアの品質管理のための品質特性についてはさまざまな議論がなされているが，適用性および実用性の観点から総合的に考えると，国際的な標準化の動きの中で ISO（International Standard Organization）でも検討されているように，**機能性**（functionality），**信頼性**（reliability），**使用性**（usability），**効率性**（efficiency），**保守性**（maintainability），**移植性**（portability）の 6 つが代表的である．これらの品質特性を以下の目的のためにできるだけ計量化し，開発プロセスを可視化するために，品質尺度とその計測基準を確立しなければならない[3]．

（1） ソフトウェア生産活動（開発規模，開発工数，開発期間，生産性，教育・訓練の必要性など）の把握

（2） ソフトウェアプロダクトの品質評価

（3） ソフトウェア生産技術とソフトウェアツールの有効性把握

このような背景から，**ソフトウェアメトリックス** (software metrics) が，ソフトウェアのさまざまな特性を判別する客観的な数学的尺度として知られており，ソフトウェアがシステムの動作環境において発揮するソフトウェア品質特性の定量的尺度を与えている（たとえば，Boehm et al.[4], Perlis et al.[5], Sherif et al.[6] 参照）．実用的なソフトウェアメトリックスの一例を**図-3**に示した（文献7), 8) 参照）．

ソフトウェアの品質管理では，実現品質（観測される品質）とユーザの要求品質とを比較しながら，その差を許容範囲内に収めるように開発プロセスの問題の発生源を修正し，開発結果である最終品質を保証することが目的となる．したがって，適切な品質管理の実施は，欠陥や誤りなどの発生源を管理状態におくことにより，後戻り作業に費やされる労力を最少化することができる．そのためには，ソフトウェアメトリックスを開発プロセスで観測された品質データに適用したソフトウェアの計測(measurement)が必要不可欠である．特にソフトウェアの品質特性のうち，充足されていて当り前であり，不十分であるとユーザに不満足感を与えるという，「**当り前品質**」としての信頼性が重要な特性要因である[2]．**ソフトウェア信頼性** (software reliability) は，

「所定の環境の下で意図する期間中に要求仕様どおりの機能を果たす品質特性」

である．しかし，開発プロセスの上流工程である設計やコーディング段階で，ソフトウェア品質を作り込むために TQC をいくら十分に実施したとしても，開発作業の結果として作り出されたソフトウェア内にさまざまな人為的誤りや欠陥などのソフトウェアエラーが潜入することは避けられない．したがって，これらのソフトウェアエラーを発見・修正・除去するテスト工程は，ソフトウェア信頼性を高めて最終的な品質/信頼性を計測・評価するために，開発プロセスにおける重要な最終工程となっている．

テスト工程におけるソフトウェア信頼性を科学的方法に基づいて計測・評価することにより，次のような開発管理上の問題について指針を与えることができる．

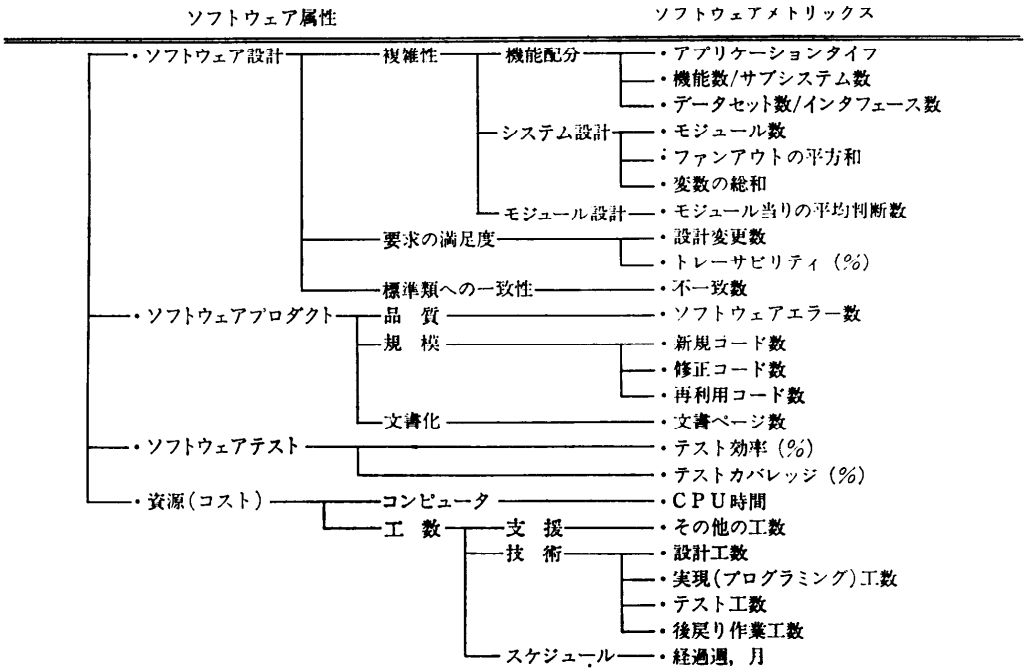


図-3 実用的なソフトウェアメトリックスの例 (Card and Glass[8] 参照)

（1） テストの進捗状況の把握
（2） ソフトウェアプロダクトの出荷品質の把握
（3） ユーザに対するリリース時期の見積り
（4） テスト工程で未発見となったソフトウェアエラーに対する運用段階における保守コストの見積り

もちろん，ソフトウェア信頼性を高めるには，使用時にソフトウェアエラーの潜入する可能性の少ない生産技術の開発も必要である．しかし，いかにテスト工程におけるソフトウェアエラーの発見（および修正・除去）過程を記述し，ソフトウェアの品質/信頼性をどのような尺度で計測・評価するかについては，さまざまな立場からの提案があり統一的な考え方はないのが現状である．

3.「当り前品質」としてのソフトウェア信頼性

以上のことから，当り前品質としてのソフトウェア信頼性をユーザに対して保証するのは重要であることが分かる．一般にソフトウェアは，ある条件の下で特定の入力データが入ってこないかぎりは，誤りや欠陥のあるプログラムパスの処理が実行されることはないのでソフトウェアによる障害を引き起こさない．したがって，コンピュータ上で実行可能なプログラムの信頼性は，入力データとプログラムの内部状態により決まる．

まず，テスト工程あるいは運用段階におけるソフトウェア信頼性を議論するために必要な，ソフトウェアの挙動を記述する用語と定義を以下にまとめる[2),9),10)]．

- **ソフトウェア故障** (software failure)
ソフトウェアが期待どおりに動作せず正しく機能しないこと．

- **ソフトウェアフォールト** (software fault)
ソフトウェアエラーが表面化して，ソフトウェア故障を引き起こすプログラム内の誤りや欠陥であり，一般にソフトウェアバグと呼ばれているもの．

- **ソフトウェアエラー** (software error)
ソフトウェアフォールトを作り込む原因となった知的活動（ユーザ要求の誤解釈や欠落，設計仕様における要求仕様の欠落や誤解釈など）を指す概念的なもの．

- **ソフトウェア信頼度** (software reliability)
ソフトウェアが所定の環境の下で意図する期間中にソフトウェア故障を引き起こすことなく期待どおりに動作する確率．

上記の用語の統一的な定義はないのが現状であるが，標準化される方向にはある．本稿では，一般的に受け入れられて使用されているように，ソフトウェア開発のテスト工程や運用段階で発見・修正される誤りや欠陥をソフトウェアエラーと呼ぶことにし，ソフトウェアフォールトと区別することなく用いる．ソフトウェアエラーは開発プロセスの各工程で作り込まれる人為的誤りであるので，要求仕様定義エラー，設計エラー，およびコーディングエラーの三つに分類することができる．

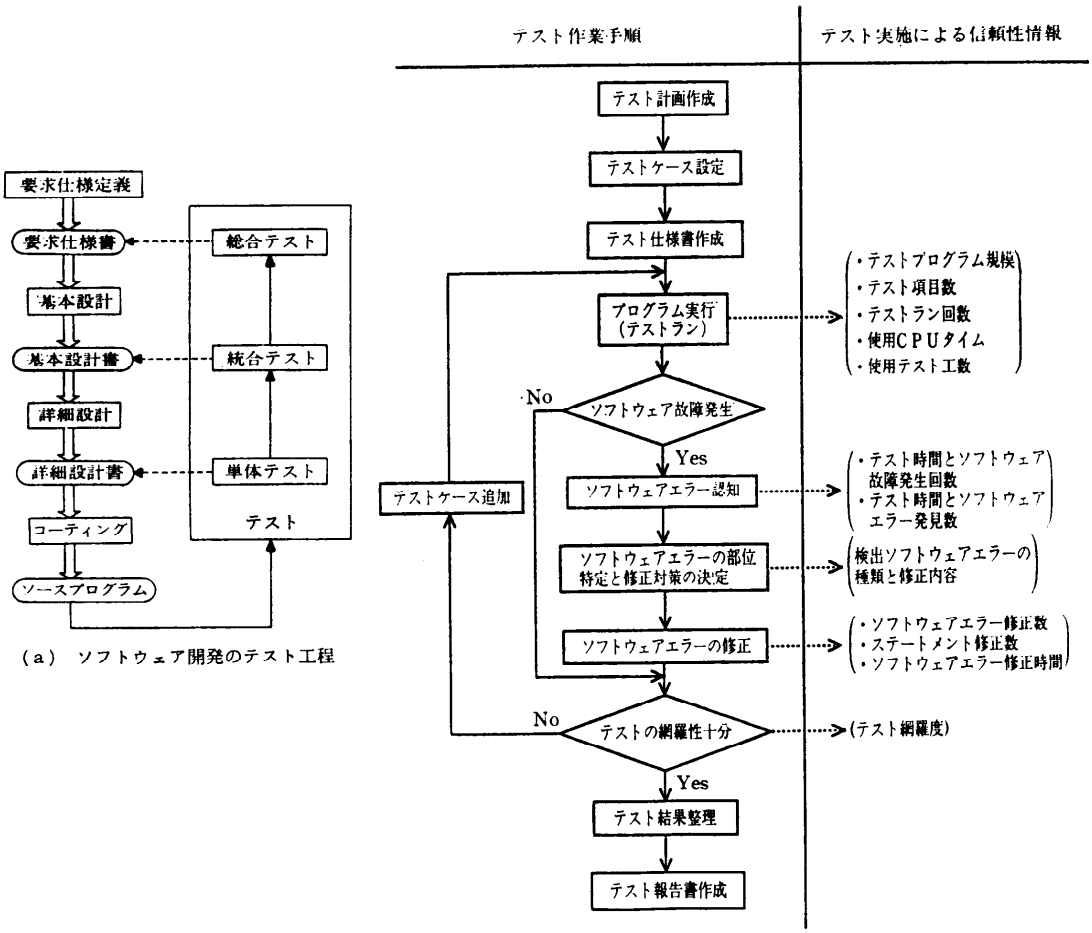
これらのソフトウェアエラーを発見・修正・除去するテスト工程は，プログラムを実行せずにソースコードを解析する**静的解析**と，テスト用データやプログラムの実行過程を追跡するテストツールを用いてその実行過程と実行結果からプログラムが正しく機能するかどうかを調べる**動的解析**という，二つの段階からなる．さらに動的解析は，次の三つの手順で進められる（**図-4** 参照）．

（1） **単体テスト**
各ソフトウェアモジュールに対して設計された機能および性能が満たされているかどうかを確認するもので，プログラム論理を中心にみるテストである．

（2） **統合テスト**
単体テストで所定の機能を有すると確認されたモジュールを結合して，サブシステムあるいはシステムとしてテストするもので，モジュール間のインタフェースや論理的つながりなどが確認される．

（3） **総合テスト**
ユーザの使用環境をシミュレートして，要求仕様に定義したものを満たしているかどうかを，機能面および性能面から全体的にテストを行う．

このようなテストを実施することにより，ソフトウェア信頼性に関する有益な情報が得られ，これをもとに品質/信頼性評価を行うことになる．特に，動的解析の中の総合テストにおけるソフト



(a) ソフトウェア開発のテスト工程

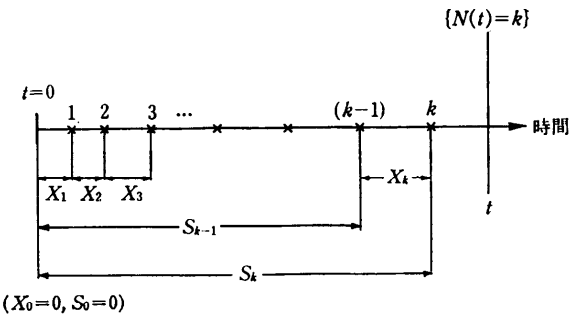
(b) テストの作業手順と信頼性情報

図-4 ソフトウェア開発におけるテスト工程とソフトウェア信頼性情報

ウェア信頼性の計測・評価は，**2.** で議論した開発管理上の問題(1)〜(4)のために重要である．図-4 には，テストの作業手順と，その結果得られるソフトウェア信頼性に関する情報を示した．

4. ソフトウェア信頼性モデル

一般に，多くの不確定要因をともなうテスト工程におけるソフトウェアエラー発見事象やソフトウェア故障発生現象を記述し，ソフトウェア信頼性を定量的に計測・評価するためには，確率・統計論に基づく数理モデル，いわゆる**ソフトウェア信頼性モデル** (software reliability model) が有用である．ソフトウェア信頼性モデルについて議論する前に，ソフトウェア信頼性の特性を表すための尺度について述べる．適当な仮定の下で[10)〜12)]，テスト工程において発見されるソフトウェアエラー数やソフトウェア故障の発生時間に関係する，次のような変数を導入できる(**図-5** 参照)．



(×：ソフトウェアエラー発見またはソフトウェア故障発生時点)

図-5 ソフトウェアエラー発見事象あるいはソフトウェア故障発生現象における確率量

$N(t)$=テスト時刻 t までに発見される総ソフトウェアエラー数（または発生した総ソフトウェア故障数），

S_k=k 番目のソフトウェア故障発生時刻 ($k=0, 1, 2, \cdots$; $S_0=0$),

X_k=$(k-1)$ 番目と k 番目の間のソフトウェア故障発生時間間隔 ($k=1, 2, \cdots$; $X_0=0$).

図-5 では，テスト時刻 t までに k 個のソフトウェアエラーが発見されているので $\{N(t)=k\}$ という事象が起こったことになる．これらの変数 $N(t)$, S_k, および X_k は，確定的な値をとることはないので確率変数 (random variable) として取り扱う必要がある．変数 S_k と X_k の間には

$$S_k=\sum_{i=1}^{k} X_i, \quad X_k=S_k-S_{k-1} \qquad (1)$$

の関係が成立する．このとき，ソフトウェアの信頼性特性を表す尺度として，$N(t)$, S_k, あるいは X_k の確率分布を考えることにより，種々の信頼性尺度を測定することができる．さらに，一定時刻 t_k までに発見された総エラー数 $N(t_k)$ の実測値 $y_k(k=1, 2, \cdots, n)$ に関する観測データをソフトウェアエラー発見数データと呼び，各ソフトウェア故障発生時刻 S_k の実測値 $s_k(k=1, 2, \cdots, n)$ に関する観測データをソフトウェア故障発生時間データと呼ぶ．

なお，テスト時間の計測単位としては

（1） カレンダー時間 (calendar time)

（2） 実行時間(execution time)またはCPU間時

（3） 工数などのテスト労力 (test effort)

（4） テストラン試行回数またはテストケース実行数

などが考えられる．精密な信頼性評価を実施するには不偏性の高い実行時間や CPU 時間を用いることが勧められるが，現実にはそのテスト環境のデータ収集上の制約を考慮に入れた計測単位を採用する．

ソフトウェア生産技術の進展にともなって，開発管理技術面から品質向上のために，定量的な信頼性評価法とその数理モデルに関する研究が精力的に行われている[2),10)~14)]．この研究は大別すると[15)]，ソフトウェア開発の設計やコーディングなどの上流工程におけるソフトウェア固有の複雑さやその実現プロセスの複雑さをモデル化して信頼性を支配する要因を分析する研究と，ソフトウェア開発の下流工程にあるテスト工程におけるソフトウェアエラー検出過程をモデル化して信頼性の達成度合や推移状況を把握する研究とがある（高橋[16)]は，前者を静的信頼性モデル，後者を動的信頼性モデルと定義して，前者について詳細に議論している）．

前者の研究は，ソフトウェア構造の特性を評価して，複雑度や静的特性としてのソフトウェア信頼性に関係づけるもので，**ソフトウェア複雑性モデル** (software complexity model) がよく知られている．このモデルは，さらにプログラム複雑性モデルとプロセス複雑性モデルに大別される．具体的には，サイクロマティック (cyclomatic) 数によりプログラムの制御フローグラフの複雑度を評価するグラフ理論的複雑性モデル (McCabe[17)])，特定モジュールの呼出し数による複雑性評価モデル (Myers[18)])，プログラム内のオペレータ (operator) とオペランド (operand) の種類と出現回数により複雑性を評価するソフトウェアサイエンス理論 (Halstead[19)]) などがよく知られている．

後者の研究では，ソフトウェアエラー発見事象やソフトウェア故障発生現象をソフトウェアの信頼度成長過程とみなす**ソフトウェア信頼度成長モデル** (software reliability growth model)[2),10),12),20)] が特に有用である．このソフトウェア信頼性モデルは，テスト時間の経過とともにソフトウェア内に潜在するエラー数は発見されて減少していくので，ソフトウェア故障の発生する確率が減少してソフトウェア信頼度が増加したり，ソフトウェア故障発生時間間隔が長くなったりするテスト過程を記述するものである．これまでに研究されてきたソフトウェア信頼度成長モデルは，次のように分類することができる[2),10)]．

（1） **時間計測モデル**

ソフトウェア故障発生時間あるいはソフトウェアエラー発見時間に基づく確率・統計モデル（確率変数 X_k や S_k などの時間変数を基本にモデルを展開する）．

（2） **個数計測モデル**

発生したソフトウェア故障数あるいは発見されたソフトウェアエラー数に基づく確率・統計モデル（確率変数 $N(t)$ などを基本にモデルを展開する）．

表-1 ソフトウェア信頼性モデルの概要

	ソフトウェア信頼度成長モデル	ソフトウェア複雑性モデル
目　的	テストの履歴から現時点でのソフトウェア故障率や潜在エラー数を推定し，信頼性の達成状況やリリース後の運用段階における信頼性を予測する．(動的信頼性の推定・予測)	ソフトウェア自体の構造的特性から複雑性を評価して信頼性を計測する．(静的信頼性の推定・予測)
主な利点	テストの進捗状況やソフトウェア開発の結果を定量的尺度により把握できる．	ソフトウェア開発の初期の段階で構造的尺度によりソフトウェア特性が評価できる．
問題点	◎モデルが多岐にわたり，モデルユーザがそれらの選択を誤ると評価結果に信頼がおけない． ◎モデルを適用するにあたり，その仮定がソフトウェア開発の実状に即していない場合がある．	◎モデルによる結果と信頼性データとの客観的な関連がみられない． ◎複雑性と検出されたソフトウェアエラーとの観測された相関が当該プロジェクトあるいはアプリケーションに特定的である．
直接的に関連する固有技術	テスト技法	構造化プログラミング，構造化技法
分　類	(1) 時間計測モデル (2) 個数計測モデル (3) アベイラビリティモデル	(1) プログラム複雑性モデル (2) プロセス複雑性モデル

(3) アベイラビリティモデル

ソフトウェアの時間的挙動を，ソフトウェア故障の発生していない動作状態とソフトウェア故障の発生した不動作状態（ソフトウェアエラーの修正・除去状態）により記述する確率モデル（確率変数 X_k と，これに対応するソフトウェアエラーの修正・除去時間を表す確率変数 Y_k を考慮して，アベイラビリティ (availability, 稼働率) を算出する）

上記のソフトウェア複雑性モデルとソフトウェア信頼度成長モデルの特徴を，比較してまとめたのが**表-1** である．

5. NHPP モデル

上述のように，ソフトウェア信頼性を数理モデルにより定量的に推定・予測する研究が盛んに行われ，数多くのソフトウェア信頼性モデルが提案されてきた．ソフトウェアの品質管理あるいは信頼性管理の立場からみれば，ソフトウェアをコンピュータ上で実行してその挙動を調べることにより確認されるという**動的信頼性**の推定・予測を行う，ソフトウェア信頼度成長モデルが有力視されている．

日本では従来より，テスト工程における品質評価モデルとして，テストにより発見された総ソフトウェアエラー数がS字形成長曲線を示すことから，人口や需要の予測などに用いられているロジスティック (logistic) 曲線やゴンペルツ (Gompertz) 曲線をソフトウェアエラーデータに直接あてはめる決定論的方法がとられてきた[1),21)]．この方法は，各成長曲線の収束値 k をソフトウェア内に潜在する総ソフトウェアエラー数として回帰分析により推定するものであり，それぞれテスト時刻 t までに発見される総エラー数は

$$L(t)=\frac{k}{1+m\cdot e^{-\alpha t}} \quad (m>0, \alpha>0, k>0) \tag{2}$$

$$G(t)=k\cdot a^{(b^t)} \quad (0<a<1, 0<b<1, k>0) \tag{3}$$

により与えられる．ここで，m, α, a, b は関数形を決める定数パラメータである．これらのモデルは，観測データに対する傾向曲線のあてはめを目的としているが，個数計測モデルに属するソフトウェア信頼度成長モデルと考えることができる．

これに対して，テストの実行過程を考慮して構築され，近年個数計測モデルに属するソフトウェア信頼度成長モデルとして注目されて実際のソフトウェアプロジェクトへの適用例も多くみられるのが，**非同次ポアソン過程** (nonhomogeneous Poisson process, 以下 **NHPP** と略す) モデルである[10),12),22)]．NHPP モデルは，テスト時刻 t までに発見される総ソフトウェアエラー数を表す確率変数 $N(t)$ に NHPP を仮定するものであり，$N(t)$ の確率分布は次式により与えられる．

$$\left.\begin{aligned} Pr\{N(t)=n\}&=\frac{\{H(t)\}^n}{n!}\exp[-H(t)] \\ &\quad (n=0, 1, 2, \cdots) \\ H(t)&=\int_0^t h(x)dx \end{aligned}\right\} \tag{4}$$

ここで，$Pr\{A\}$ は事象 A の生起する確率を表す．式(4)の $H(t)$ は，$N(t)$ の平均値を表す**平均値関数** (mean value function) であり，時間区間 $(0, t]$ において発見される総期待ソフトウェアエラー数（または発生する総期待ソフトウェア故障数）を表す．また，式(4)の $h(t)$ は，時刻 t におけるソフトウェアエラー発見率（またはソフトウェア故障率）を表し，**強度関数** (intensity function) と呼ばれる．

式(4)で定式化される NHPP モデルに基づいて，種々の信頼性評価尺度を導出できる[2),10),12)]．たとえば，テスト時刻 t におけるソフトウェア内の残存エラー数の平均値 $n(t)$ とその分散（平均値からの散布度を表す）$v(t)$ は一致し，

$$n(t)=v(t)=a-H(t) \qquad (5)$$

となる．ここで，$a=H(\infty)$ とし，パラメータ a はテストにより最終的に発見される総期待ソフトウェアエラー数，すなわちテスト開始前にソフトウェア内に潜在する総期待エラー数を表す．また，テストが時刻 t まで進行しているときに，時間区間 $(t, t+x]$ $(x\geq 0)$ においてソフトウェア故障の発生しない確率は，

$$R(x|t)=\exp[-\{H(t+x)-H(t)\}] \qquad (6)$$

となる．通常，式(6)は**ソフトウェア信頼度** (software reliability) と呼ばれている．式(6)を用いて，ソフトウェアの運用段階における信頼度を評価するときには，t を総テスト時間，x を所定の運用時間と考えればよい．さらに，式(4)の強度関数 $h(t)$ の逆数である

$$\mathrm{MTBF}(t)=\frac{1}{h(t)} \qquad (7)$$

あるいはテスト時間を式(4)の平均値関数 $H(t)$ で割った

$$\mathrm{MTBF}(t)=\frac{t}{H(t)} \qquad (8)$$

は，平均ソフトウェア故障発生時間間隔（または

表-2　代表的な NHPP モデル

NHPP モデル	平均値関数 $H(t)$	強度関数 $h(t)$	環　境
指数形ソフトウェア信頼度成長モデル (exponential SRGM)	$m(t)=a(1-e^{-bt})$ $(a>0, b>0)$	$h_m(t)=abe^{-bt}$	テスト期間を通じてエラー発見率が一定のソフトウェア故障発生現象を記述する[23),24)]．
修正指数形ソフトウェア信頼度成長モデル (modified exponential SRGM)	$m_p(t)=a\sum_{i=1}^{2}p_i(1-e^{-b_i t})$ $\left(a>0, 0<b_2<b_1<1, \sum_{i=1}^{2}p_i=1, 0<p_i<1\right)$	$h_p(t)=a\sum_{i=1}^{2}p_i b_i e^{-b_i t}$	テストの進行にともなうエラーの発見難易性を記述する（発見の容易なエラーの発見率 b_1，発見の困難なエラーの発見率 b_2）[25),26)]．
遅延S字形ソフトウェア信頼度成長モデル (delayed S-shaped SRGM)	$M(t)=a[1-(1+bt)e^{-bt}]$ $(a>0, b>0)$	$h_M(t)=ab^2te^{-bt}$	エラーの発見に至る過程を，ソフトウェア故障発見過程とエラー認知過程という，引き続く二つの事象により記述する[27),28)]．
習熟S字形ソフトウェア信頼度成長モデル (inflection S-shaped SRGM)	$I(t)=\frac{a(1-e^{-bt})}{(1+ce^{-bt})}$ $(a>0, b>0, c>0)$	$h_I(t)=\frac{ab(1+c)e^{-bt}}{(1+ce^{-bt})^2}$	テスト労力の変動やテストチームの習熟度を考慮して，ソフトウェア故障発生現象を記述する[29),30)]．
テスト労力依存型ソフトウェア信頼度成長モデル (test-effort dependent SRGM)	$T(t)=a[1-e^{-rW(t)}]$ $W(t)=\alpha[1-e^{-\beta t^m}]$ $(a>0, 0<r<1, \alpha>0, \beta>0, m>0)$	$h_T(t)$ $=ar\alpha\beta mt^{m-1}e^{-\beta t^m}e^{-rW(t)}$	テスト工程における工数などの投入労力量と発見される総エラー数を関係づけ，その時間的変化を記述する[31),32)]．
対数型ポアソン実行時間モデル (logarithmic Poisson execution time model)	$\mu(t)=\frac{1}{\theta}\ln(\lambda_0\theta t+1)$ $(\lambda_0>0, \theta>0)$	$\lambda(t)=\frac{\lambda_0}{(\lambda_0\theta t+1)}$	テスト時間を CPU 時間で計測する時に，ソフトウェア故障率が発生するソフトウェア故障数に関して指数関数的に減少する[33),34)]．

(SRGM: Software Reliability Growth Model)

a＝テスト開始前にソフトウェア内に潜在する総期待エラー数
b, b_i, r＝エラー発見率を表すパラメータ
c＝エラー発見の習熟度を表すパラメータ
α, β, m＝テスト労力関数 $W(t)$ を決めるパラメータ
λ_0＝初期ソフトウェア故障率
θ＝ソフトウェア故障率の減少率

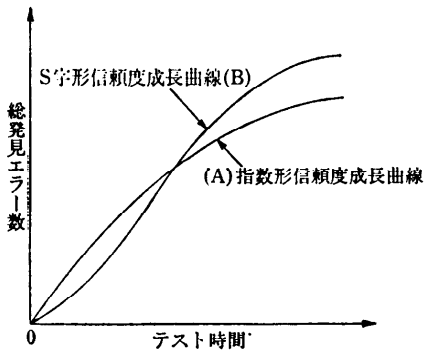


図-6 ソフトウェアの信頼度成長曲線

平均ソフトウェアエラー発見時間間隔）いわゆるMTBF (mean time between failures) の尺度として考えることができる．式(7)および式(8)は，それぞれテスト時刻 t における瞬間 MTBF (instantaneous MTBF) および累積 MTBF (cumulative MTBF) と呼ばれる[10]．

NHPP モデルでは，二つの代表的な信頼性評価基準，すなわちソフトウェア内に潜在する総エラー数とソフトウェア故障発生時間の両者を取り扱うことができる．そこで，具体的にソフトウェアの信頼性評価を行うには，式(4)の平均値関数 $H(t)$ を特定化しなければならない．従来より，種々のテスト環境におけるソフトウェアエラー発見事象あるいはソフトウェア故障発生現象を記述する NHPP モデルが，現実的な仮定[11]の下で数多く提案されている．代表的な NHPP モデルを**表-2**にまとめた．一般に，ソフトウェアの信頼度成長は，費やされたテスト時間と発見された総エラー数との関係によりとらえられ，その関係は数学的に**信頼度成長曲線** (reliability growth curve) により表すことができる．表-2に示した NHPP モデルのうち，指数形および修正指数形ソフトウェア信頼度成長モデルは，実際の信頼度成長曲線が指数（関数）形傾向を示す場合に適切であり，また遅延S字形および習熟S字形ソフトウェア信頼度成長モデルは，それがS字形傾向を示す場合に適切である（**図-6** 参照）．

このように，当該テスト環境に適切なNHPPモデルの平均値関数を特定化すると，モデルパラメータおよび信頼性評価尺度を，**最尤法** (method of maximum-likelihood)[2),10),12)] により推定できる．

6. ソフトウェア信頼性モデルの現状と問題点

Musa et al.[12),34),35)] は，実際の適用にあたり，ソフトウェア信頼性モデルの選択基準および比較基準を次のように与えている．

（1） **予測妥当性** (predictive validity)：将来のソフトウェアエラーやソフトウェア故障の挙動を十分に予測できるか．

（2） **有効性** (usefulness)：管理上有益な信頼性尺度を計算できるか．

（3） **仮定の質** (quality of assumptions)：妥当な仮定に基づいているか．

（4） **適用性** (applicability)：広範囲に適用可能か．

（5） **簡潔性** (simplicity)：単純かつ明解であるか．

これらの基準により Musa らは，既存のモデルを実際データに適用した総合的評価の結果，表-2のうち指数形ソフトウェア信頼度成長モデルおよび対数型ポアソン実行時間モデルを推奨している．さらに欧米では，テスト時間の計測単位として実行時間や CPU 時間を用いて，ソフトウェア故障発生時間を信頼性評価基準とするソフトウェ

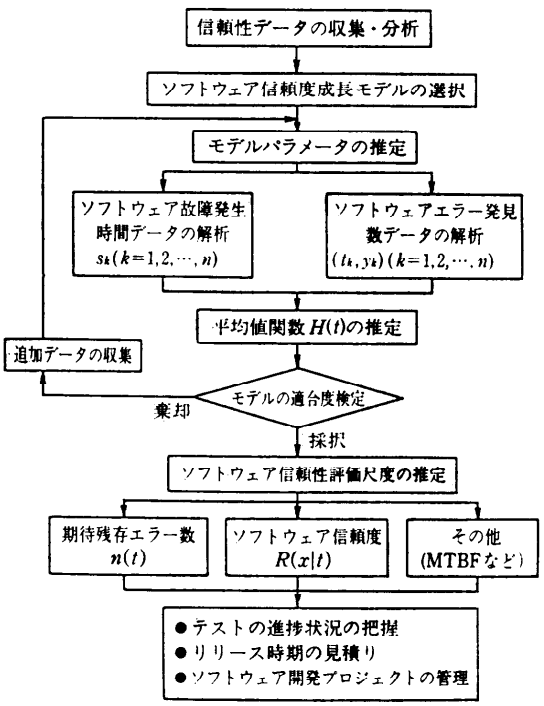


図-7 NHPP モデルによる信頼性データ解析手順

ア信頼度成長モデルがよく研究されている．これに対して日本では，テスト時間の計測単位としてカレンダー時間やテスト労力を採用した品質/信頼性評価例が多い[36]．さらに実際的な観点から，従来より適用されてきた前述のロジスティック曲線モデルおよびゴンペルツ曲線モデルと，NHPPモデルを組み込んだソフトウェア品質（信頼性）評価ツールも開発されており実用に供されている[10),36)]．これらのツールでは，主にS字形ソフトウェア信頼度成長モデルの適用に重点をおいている．

テスト時間の計測単位として実行時間やCPU時間を用いたソフトウェア故障発生時間に基づくNHPPモデルでは，精密な信頼性評価を実行できる利点をもつが，テスト時間の計測上の困難性をともなうとともに，ソフトウェア故障発生時間のデータ記録用ツールも必要である．これに対して，カレンダー時間やテスト労力を用いたソフトウェアエラー発見数に基づくNHPPモデルでは，データ収集と記録も容易であり，観測されたデータ数が比較的大きいならば妥当な結果も得られ，その信頼性評価結果は開発管理へ反映しやすい．これらの特徴は，NHPPモデル以外のソフトウェア信頼度成長モデル[2),10),21),37)]についてもあてはまる．

7. ソフトウェア信頼性評価のツールと実例

平均値関数 $H(t)$ をもつNHPPモデルをテスト工程で観測された測定データに適用するときの，一連の信頼性データ解析手順をまとめたのが**図-7**である．山田[10),38)]は，この信頼性データ解析手順をプログラムパッケージ化し，ソフトウェア開発のテスト工程において定量的に信頼性評価を行うための支援ツール**SRET** (software reliability evaluation tool) を作成している．SRETでは，三つのNHPPモデルと二つの回帰モデルが組み込まれ，モデルパラメータの推定法として最尤法[2),10),12)]を，また測定データに対するモデルの適合度検定法としてコルモゴロフ・スミルノフ (Kolmogorov-Smirnov) 検定 (K-S検定と略す)[2),10),11)]を採用している．SRETの構造を**図-8**に示す（このSRETと同様な考え方の下で実際的に構築された信頼性評価システムにSOREM[39)]がある）．

この SRET を用いて実施したソフトウェア信頼性評価例を示す．本例の評価対象となったソフトウェアは，大手銀行の外国証券システムであり，オンラインプログラム数とバッチプログラム数が，それぞれ 50 と 150 という勘定系システムである[36),40)]．このシステムの総合テストには 31 人のテスト要員が参画し，27 組のソフトウェアエラー発見数データが観測された．SRET による信頼性データ解析結果とテストの実行プロセスを考慮した総合的評価から，表-2 に示した遅延

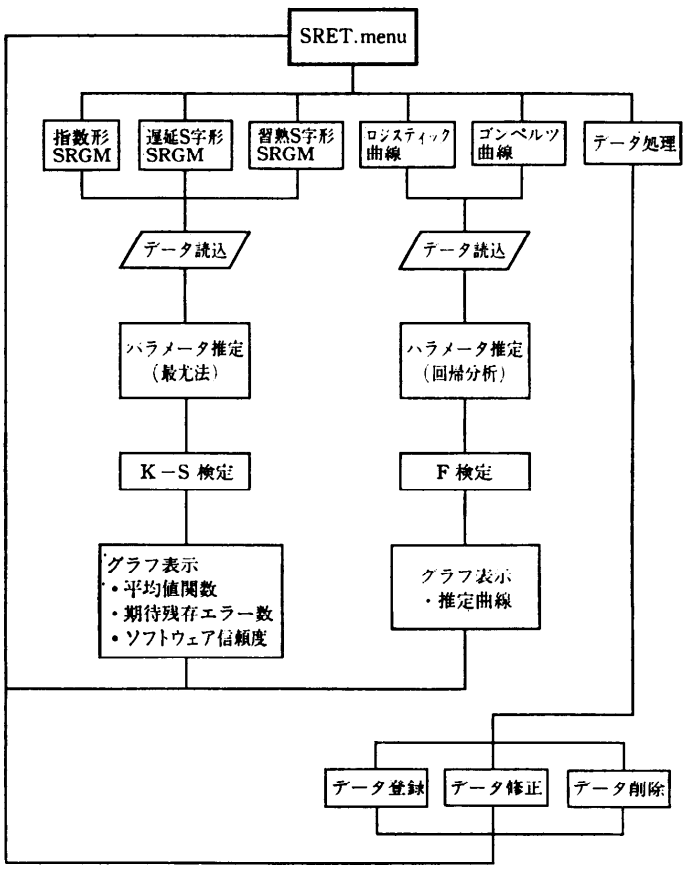


図-8　ソフトウェア信頼性評価ツール SRET の構造（三つの NHPP モデルとして指数形，遅延S字形，習熟S字形ソフトウェア信頼度成長モデルと，ロジスティック曲線モデル，ゴンペルツ曲線モデルを組み込む）

S字形ソフトウェア信頼度成長モデルの適合性の良さが確認された．**図-9** に，その平均値関数 $M(t)=a[1-(1+bt)e^{-bt}]$ の推定値 $\hat{M}(t)$ とその 90% 信頼限界[10),36)]（90% の信頼度で平均値関数 $M(t)$ が含まれている範囲）を，実測データとともに示した．ここで，図-9 からモデルパラメータの最尤推定値は $\hat{a}=171.1$ と $\hat{b}=0.1188$ であることが分かり，平均値関数 $M(t)$ の最尤推定値は

$$\hat{M}(t)=171.1\cdot[1-(1+0.1188t)e^{-0.1188t}] \quad (9)$$

により与えられる．このとき，K-S 検定により，式(9)の平均値関数 $\hat{M}(t)$ をもつ NHPP モデルの実測データに対する適合性も確認される．したがって，テスト終了時点 $t=27$(日) までに発見された総ソフトウェアエラー数は 142 個であるので，テストにより未発見となったソフトウェア内の潜在エラー数は $\hat{a}-142=29.1$ より約 30 個と推定される．$H(t)\equiv M(t)$ として，テスト時間の経過にともな

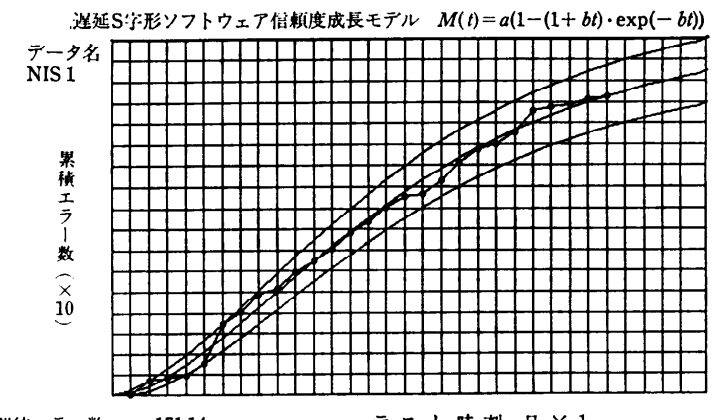


図-9 SRET による遅延S字形ソフトウェア信頼度成長モデルに基づく信頼性評価例

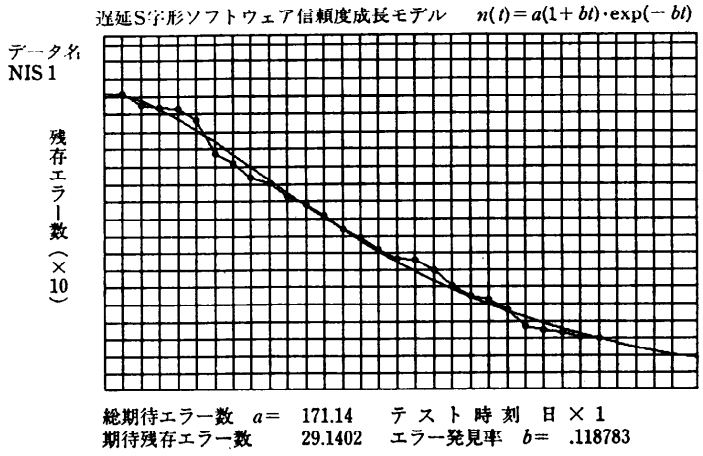


図-10 SRET による期待残存ソフトウェアエラー数 $n(t)$ の推定結果

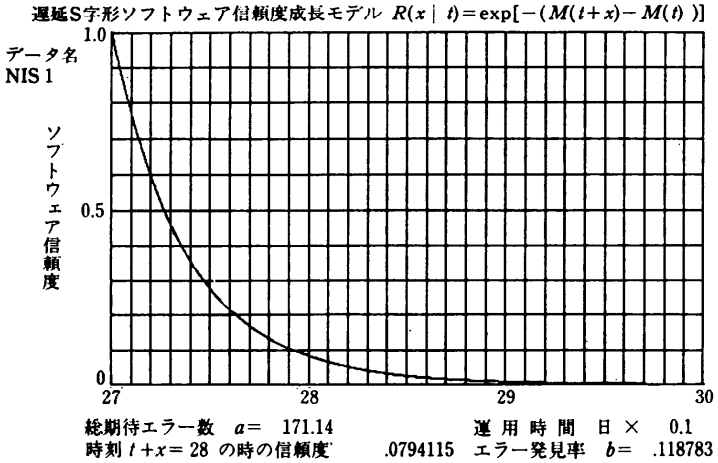


図-11 SRET によるソフトウェア信頼度 $R(x|t)$ の推定結果

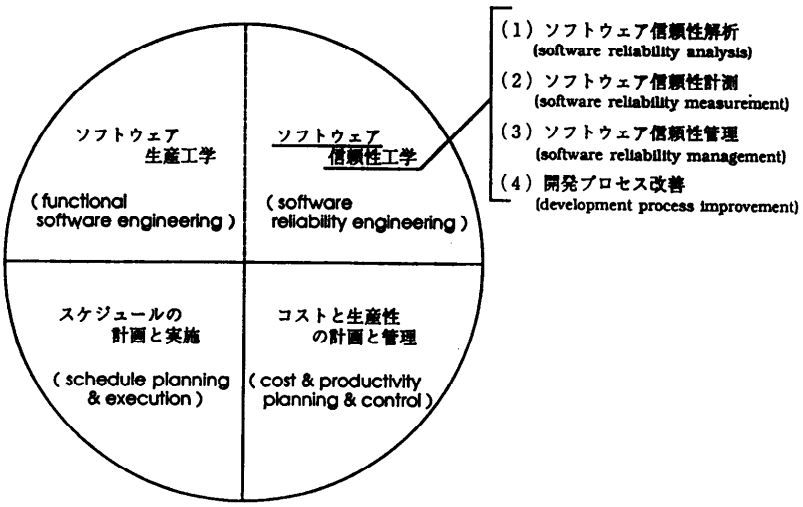


図-12 J. D. Musa らによるソフトウェア工学の分野とソフトウェア信頼性工学 (Redwine[41] 参照)

う式(5)の期待残存エラー数 $\hat{n}(t)=\hat{a}-\hat{M}(t)$ の推移を**図-10** に示した．さらに同様にして，**図-11** には式(6)のソフトウェア信頼度の最尤推定値 $\hat{R}(x|t)$ を示した．これは，テスト終了時点 $t=27$ (日)以降のソフトウェア信頼度の挙動

$$\hat{R}(x|t=27)=\exp[-171.1 \cdot\{(1+0.1188(27.0)) \cdot e^{-0.1188(27.0)}-(1+0.1188 \cdot(x+27.0))e^{-0.1188(x+27.0)}\}] \quad (10)$$

を示しており，この時点でソフトウェアをリリースしてテスト環境と同じ厳しさで運用するならば，運用を開始して1日後 ($x=1.0$) の信頼度は約 0.08 であることを示している．

8. むすび

本稿では，ソフトウェア品質の基本的な考え方を述べたうえで，その計量化と管理法について議論し，ソフトウェアの品質評価技術を考察した．特に，「当り前品質」としての品質特性であるソフトウェア信頼性を取りあげ，定量的な信頼性評価法とその問題点や動向について議論した．以上の考え方は，ユーザが望まないソフトウェアの運用特性，すなわちソフトウェア故障を回避することの重要性を強調することにより，ユーザの要求品質の充足性を高めることを研究目的とする**ソフトウェア信頼性工学** (software reliability engineering) に直結するものである．これは，AT&T 社ベル研究所の J. D. Musa らの研究グループが提唱するものであり（文献 41) 参照)，ソフトウェア工学における主要な構成要素として位置づけている (**図-12** 参照)．

システムの高信頼化は，ハードウェアについてはかなりの実績をあげてきたが，ソフトウェアについては現在でも発展途上段階にあると言わざるをえない．ソフトウェアの品質/信頼性に関する専門用語や実現技術の標準化をも含めて，さらに今後の研究成果に期待するところが大きい．なお，ソフトウェア品質評価の実際的な事例については文献 42) が参考になる．

謝辞 本稿の執筆をお勧めいただきました(株)東芝システム・ソフトウェア研究所の大筆豊部長と，静岡大学工学部情報知識工学科の落水浩一郎教授に謝意を表します．

参考文献

1) 菅野文友編著：ソフトウェアの品質管理，日科技連出版社 (1986).
2) 山田 茂，大寺浩志：ソフトウェアの信頼性〜理論と実践的応用〜，ソフト・リサーチ・センター (1990).
3) Pressman, R. S.: *Software Engineering: A Practitioner's Approach (2nd ed.)*, McGraw-Hill, New York (1987).
4) Boehm, B. W., Brown, J. R. and Lipow, M. Quantitative Evaluation of Software Quality, *Proc. 2nd Int. Conf. Software Engineering*, pp. 592-605 (1976).
5) Perlis, A., Sayward, F. and Shaw, M. (eds.): *Software Metrics: An Analysis and Evalua-*

tion, MIT Press, Cambridge (1981).
6) Sherif, Y. S., Ng, E. and Steinbacher, J.: Computer Software Development: Quality Attributes, Measurements, and Metrics, *Naval Research Logistics*, Vol. 35, No. 3, pp. 425-436 (1988).
7) 山田 茂：ソフトウェアマネジメントモデル入門(2), **bit**, Vol. 23, No. 1, pp. 67-75 (1991).
8) Card, D. N. and Glass, R. L.: *Measuring Software Design Quality*, Prentice-Hall, Englewood Cliffs (1990).
9) IEEE (ed.): *IEEE Standard Glossary of Software Engineering Terminology*, ANSI/IEEE Std 729-1983, New York (1983).
10) 山田 茂：ソフトウェア信頼性評価技術, HBJ 出版局 (1989).
11) Goel, A. L.: Software Reliability Models: Assumptions, Limitations, and Applicability, *IEEE Trans. Softw. Eng.*, Vol. SE-11, No. 12, pp. 1411-1423 (1985).
12) Musa, J. D., Iannino, A. and Okumoto, K.: *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, New York (1987).
13) Bittanti, S. (ed.): *Software Reliability Modeling and Identification*, Springer-Verlag (Lecture Notes in Computer Science No. 341), Berlin (1988).
14) Malaiya, Y. K. and Srimani, P. K. (eds.): *Software Reliability Models: Theoretical Developments, Evaluation and Applications*, IEEE Computer Society Press, Los Alamitos (1990).
15) Bishop, P. G. (ed.): *Dependability of Critical Computer Systems 3: Techniques Directory*, Elsevier Applied Science, London (1990).
16) 高橋宗雄：ソフトウェアの信頼性評価法に関する研究, 九州大学学位論文 (1989).
17) McCabe, T. J.: A Complexity Measure, *IEEE Trans. Softw. Eng.*, Vol. SE-2, No. 4, pp. 308-320 (1976).
18) Myers, G. J.: *Reliable Software through Composite Design*, Petrocelli/Charter, New York (1975).
19) Halstead, M. H.: *Elements of Software Science*, Elsevier North-Holland, New York (1977).
20) Ramamoorthy, C. V. and Bastani, F. B.: Software Reliability-Status and Perspectives, *IEEE Trans. Softw. Eng.*, Vol. SE-8, No. 4, pp. 354-371 (1982).
21) 三浦 武：ソフトウェアの品質評価法, 日科技連出版社 (1981).
22) Yamda, S. and Osaki, S.: Software Reliability Growth Modeling: Models and Applications, *IEEE Trans. Softw. Eng.*, Vol. SE-11, No. 12, pp. 1431-1437 (1985).
23) Goel, A. L. and Okumoto, K.: Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures, *IEEE Trans. Reliab.*, Vol. R-28, No. 3, pp. 206-211 (1979).
24) Goel, A. L.: Software Error Detection Model with Applications, *J. Syst. Softw.*, Vol. 1, pp. 243-249 (1980).
25) Yamada, S. and Osaki, S.: Nonhomogeneous Error Detection Rate Model for Software Reliability Growth, in *Stochastic Models in Reliability Theory*, Osaki, S. and Hatoyama, Y. (eds.), pp. 120-143, Springer-Verlag, Berlin (1984).
26) Yamada, S., Osaki, S. and Narihisa, H.: A Software Reliability Growth Model with Two Types of Errors, *RAIRO-Operations Research*, Vol. 19, No. 1, pp. 87-104 (1985).
27) 山田 茂, 尾崎俊治：ソフトウェアの信頼度成長モデルとその比較, 信学論(D), Vol. J 65-D, No. 7, pp. 906-912 (1982).
28) Yamada, S., Ohba, M. and Osaki, S.: S-shaped Reliability Growth Modeling for Software Error Detection, *IEEE Trans. Reliab.*, Vol. R-32, No. 5, pp. 475-478, 484 (1983).
29) Ohba, M.: Inflection S-shaped Software Reliability Growth Model, in *Stochastic Models in Reliability Theory*, Osaki, S. and Hatoyama, Y. (eds.), pp. 144-162, Springer-Verlag, Berlin (1984).
30) 山田 茂, 大場 充：エラー発見率に基づくS字形ソフトウェア信頼度成長モデルの考察, 情報処理学会論文誌, Vol. 27, No. 8, pp. 821-828 (1986).
31) 山田 茂：テスト労力の投入量を考慮したソフトウェア信頼度成長モデルと適合性比較, 信学論(D), Vol. J 70-D, No. 12, pp. 2438-2445 (1986).
32) Yamada, S., Ohtera, H. and Narihisa, H.: A Testing-Effort Dependent Reliability Model for Computer Programs, *Trans. IECE Japan*, Vol. E 69, No. 11, pp. 1217-1224 (1986).
33) Musa, J. D. and Okumoto, K.: A Logarithmic Poisson Execution Time Model for Software Reliability Measurement, *Proc. 7th Int. Conf. Software Engineering*, pp. 230-238 (1984).
34) Musa, J. D. and Okumoto, K.: Application of Basic and Logarithmic Poisson Execution Time Models in Software Reliability Measurement, in *Software System Design Method*, Skwirzynski (ed.), pp. 275-298, Springer-Verlag, Berlin (1986).
35) Iannino, A. and Musa, J. D.: Software Reliability, in *Advances in Computers* (*Vol. 30*), Yovits M. C. (ed.), pp. 85-170, Academic Press, New York (1990).
36) Yamada, S.: Software Quality/Reliability Measurement and Assessment: Software Reliability Growth Models and Data Analysis, *J. Inf. Process.*, Vol. 14, No. 3 (1991).
37) 当麻喜弘：ソフトウェア信頼性モデルと評価, 信学誌, Vol. 73, No. 5, pp. 461-466 (1990).
38) 山田 茂, 磯崎龍史, 尾崎俊治：ソフトウェア信頼性評価ツール (SRET) の作成, 信学論 (D-I), Vol. J 72-D-I, No. 1, pp. 24-32 (1989).
39) Uemura, S., Yamada, S. and Fujino, K.: Software Reliability Evaluation Method: Application of Delayed S-shaped NHPP Model and Other Related Models, *Proc. Int. Symp. Reliability and Maintainability*, pp. 467-472 (1990).
40) 木村茂和：ホスト適用業務プログラムにおけるソフトウェア信頼性評価, ソフトウェア・シンポジウム '91 論文集, pp. B-26-B-34 (1991).
41) Redwine, S. T., Jr. (ed.): Software Engineering Technical Committee NEWSLETTER, Supplement to the January 1991 issue of *IEEE Software*.
42) 菅野文友監修：ソフトウェア品質管理事例集, 日科技連出版社 (1990).

(平成3年5月31日受付)

山田 茂（正会員）

昭和27年生．昭和50年広島大学工学部経営工学科卒業．昭和52年同大学院修士課程修了．昭和52～55年日本電装(株)勤務．昭和58年広島大学大学院博士課程修了．昭和58～63年岡山理科大学勤務．昭和63年広島大学工学部第二類（電気系）助教授，現在に至る．工学博士．ソフトウェア信頼性モデル，ソフトウェアマネジメントモデル，信頼性工学，品質管理などの研究に従事．著書「ソフトウェア信頼性評価技術」，「ソフトウェアの信頼性～理論と実践的応用」など．電子情報通信学会，日本 OR 学会，日本経営工学会，IEEE 各会員．