ソフトウェア工学 124－4
（1999．10．21）

残存エラー数の推定が可能なプログラムの試験法（5）
－信頼度成長曲線－

若杉忠男
静岡産業大学国際情報学部
連絡先：〒426-8668 静岡県藤枝市駿河台 4-1-1
電話 054-646-5406(ﾀﾞｲﾔﾙｲﾝ)
Eメール：wakasugi@fujieda-ssu.ac.jp

あらまし

筆者は，先にフローグラフのパスを試験項目でカバーすることにより，エラー数を推定する方法を提案し，パス分解法と名付けたが，本論文では，その理論をコルモゴロフの理論と結びつけ，テストカバレジを定義し，それに基づいて信頼度成長曲線を導き，シミュレーションでその使用例を示す．またその妥当性を裏付けるために，そのモデルから，よく知られている指数形やＳ字形の信頼度成長曲線モデルを導き，あわせて従来のモデルに新たな解釈を与える．

On a software test method capable of estimating the number of programming errors (5)
- Reliability growth curves of program testing -

Tadao WAKASUGI, Member
Sizuoka Sangyou University
4-11-1 Surugadai Fujieda-City, Sizuoka-Prefecture, 4268668 Japan
E-mail : wakasugi@fujieda-ssu.ac.jp

Abstract

The author proposed a technique of estimating the number of errors of program on the basis of path coverage of flowgraph and called it the path decomposition method. Here the method is combined with Kolmogorov's theory, and a new concept of test coverage is defined. By the coverage, a reliability grows model is introduced, and an example of simulation by the model is shown. To validate usefulness of the method, traditional exponential and S-shaped reliability growth models are deduced from the technique, and besides, a new explanation about traditional reliability grows models are given.

1 はじめに

筆者はプログラムのフローグラフをパスに分解し，残存エラー数を評価する方法を提案し，パス分解法と名付けた[1][2][3]．ループを含むパスのテストは、試験項目のループの回数をむやみに多くしてもエラーを見逃す確率はあまり変わらず、適当に打ち切ってもかまわないであろう．本論文では，この考えをコルモゴロフの理論に関連付け，無限ループを含むプログラムの試験項目のパスカバレジを定義し，そのカバレジと発見エラー数との関係から残存エラー数を推定する．またシミュレーションによりパス分解法の妥当性を示し，さらに従来から提案されている信頼度成長曲線とパス分解法との関係を示す。

2　従来の信頼度成長曲線の問題点

現在使われている信頼度成長曲線は主に帰納的方法によるものであり，実際に得られたバグ累積曲線に適当な関数を当てはめ，うまく当てはめられない場合には他の関数を選んで合わせるというもので，どのような関数が当てはまるかを事前に推定することは難しく，試験作業全工程の２／３程度が終了したころに収束値が推定できるという状況である．したがって試験の初期工程の進捗把握などにはあまり使えず，主に試験の終わりの段階で，残存エラー数の推定や完了時点の予測などに使われる．推定精度の悪い理由として，次のようなことが考えられる．

(1)現在よく使われている指数型，S字型，ゴンペルツ曲線など信頼度成長曲線のモデルには，プログラムの難易度と作業者の能力が明確な形では含まれていない．

(2)従来の信頼度成長曲線は，一般に横軸に試験作業経過時間やＣＰＵ時間を採用するが[4]，これらは作業上のいろいろな都合で変動するものである．ハードウエアの使用段階での信頼性は、初期不良や劣化などがあるので時間に依存すると言えるが、その考えをソフトウエアの開発段階に当てはめるのは無理がある。

試験項目数やテストインスタンスという作業単位を独立変数に採用した方法も提案されているが，まだ決定的なものとは言えない．

(3)モデルの理論的根拠としては，主に Non-homogeneous Poisson process を使い，「単位時間あたりに発見されるフォールトの数はその時点でソフトウエア内に残存するフォールト数に比例する」と仮定する．しかし，ソフトの一部を試験しているときのフォールトの発見率が，当面の試験の対象ではない他の部分に存在するフォールトも含めた全体のフォールトの数に比例するとは考えられない．

ここで紹介するパス分解法はコルモゴロフの理論に基づいた演繹的方法で，モデルにプログラムの複雑度と作業者のエラー発見力を含み，横軸は時間でなくパスの長さを使い，エラー発見数は試験項目が通過するパスにおけるフォールト数に依存するとする．

3　パス分解法の紹介

ここでパス分解法を簡単に紹介するが，詳細は資料[1][2][3]などを参照されたい．

まず次のような定義を行う．

定義１　フローグラフ

プログラムのフローグラフはノードとリンクからなり，リンクはプログラムのステートメント，ノードはステートメントの区切りや分岐点を表す．

定義２　パスベクトル

一連のリンクをパスと呼ぶ．パスがＬ個のリンクからなるとき，長さＬのパスと定義し，フローグラフ全体に含まれる長さＬのパスの個数をP_L，そのうち試験項目でカバーした部分集合をp_Lなどと記述する．またそれらを長さ順に並べ，それぞれパスベクトル $\{P_L\}$ $\{p_L\}$ と記述する．

P_1はプログラムステートメントの個数を代表し，P_2は真ん中にノードを含むパスなのでブランチのあるパスの個数を代表する．同様にP_3はノードを二つ含むパスの個数であり，$\{P_L\}$ はプログラム全体の複雑さや規模，構造を代表する指標と考えられる．なお，$\{P_L\}$ は連結行列を使えば簡単に求められる[1][2][3]．

定義３　フォールトとエラー

プログラムの間違った作り方をフォールト，その結果生じた仕様書と一致しない現象をエラーと呼ぶ．

フォールトが原因で，エラーが結果である．エラーとは，フォールトのあるリンクと，その悪影響が発見されたリンクとの不整合と考え，実際に前後どちらのリンクを修正したかにかかわらず，前のリンクをフォールト，後のリンクをエラーとする．なお，資料[4]ではエラーではなく，フェイラーという言葉を使っているが，

ここでは資料[2]に合わせてエラーとする.

前提1　対象とするプログラムの条件は次のとうりである.

(1) プログラムはフローグラフで表わされる.

(2) プログラムは摩耗や劣化がなく，無限に繰り返すことができ，リセットすれば完全に初期状態に戻るものとする.

(3) フローグラフのすべてのパスは実行でき，またすべてのループは無限回実行できるものとする.

(4) 簡単化のために，リンク内のエラー発見率rはリンクによらず一定とする.

サンプルとして，資料[1]にも使用した図1のフローグラフに，abbbdというパスを通る試験項目を適用する例を使って説明する.

パス分解法の基本式は，次のようなものである．あるリンクaに含まれるフォールトの数をf_a個とすると，リンク内でのエラー発見期待値は$f_a \times r$となる．そのリンクで見逃したエラーが下流の次のリンクで発見される期待値は$f_a \times r \times (1-r)$となる．以下同様にして，リンクaに$f_a$個のエラーがあれば、リンクaを先頭にもつ長さLのパスのエラー発見期待値は，

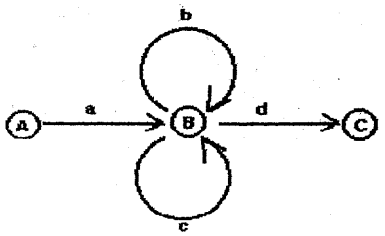
$$f_a \times \sum_{I=1}^{L} \{r \times (1-r)^{I-1}\} \qquad (1)$$

となる．フォールトはすべてのリンクにありうるから，試験項目が通過するリンクすべてについて(1) 式を求めて加えなければならない.

前提2 汚染モデル

パスに枝別れがある場合，上流にあるフォールトの影響は (1) 式に基づいて下流のリンクすべてに及ぶ．したがって，フローグラフに枝別れがあるとエラーの数は増加し，リンクaにあるフォールトの悪影響がbとcに伝わり，bで見つけて修正してもcへの悪影響は残ると考える．これを汚染モデルと呼ぶ[3].

ここで，パス分解の仕方と(1)式の計算例を示す．パスの中に含まれるすべてのリンクの連なりを考え，同じものが二つ以上ある場合には一つにする．図1の例で示すと，試験項目abbbdがカバーするパスについて，長さ1はa, b, b, b, dの5個，長さ2はab, bb, bb, bdの4個，長さ3はabb, bbb, bbdの3個，長さ4はabbb, bbbdの2個，長さ5はabbbdの1個であるが，このうち，bが3個，bbが2個あるのでそれぞれ1個とすると，長さ別に並べたパスは，{a, b, d, ab, bb, bd, abb, bbb, bbd, abbb, bbbd, abbbd} の12個となる．したがって，この試験項目のパスベクトルは{3, 3, 3, 2, 1}で，これが試験項目abbbdの効果を示す指標となる．これらすべてについて，(1)式を求め合計したものが，試験項目abbbdの発見エラー数の期待値となる.



Path vector={4, 9, 18, 36, ・2.25× 2^L ・・}

	A	B	C
A	0	1	0
B	0	2	1
C	0	0	0

Connection matrix

図1 フローグラフの例

これを(2)式に示すように，試験項目$_i$の期待値関数E（試験項目$_i$）と記述する．

発見エラー総数=E（全試験項目）　　(2)

この計算は手計算では面倒であるが，プログラム化できる．

4　テストカバレジ

本理論ではプログラムを試験するということは，プログラムのパス全体｛P_L｝の中から，発見エラー数がなるべく多くなるように試験項目｛p_L｝を選択することと考える．P_1全体を試験項目でカバーすることはステートメントカバレジテストに相当し，P_1とP_2全体をカバーすることはブランチカバレジテストに相当する．同様に，P_3のカバレジテスト，P_4のカバレジテストと試験の質が高くなる．これらをC_1テスト，C_2テストなどと表現すれば，完全網羅試験はC_∞テストとなる．

エラー関数Eで各リンクにあるフォールトの数をすべて1としたものを，特にカバレジ関数Cと定義する．フローグラフのパス全体｛P_L｝すなわちリンクの組み合わせの数は一般に無限の長さをもち，また長さLの指数関数で増加するが，リンクの数は有限（図2の例では5個）で，したがって各リンクに$(1-r)^L$という重みを掛けることにより，6章定理2で述べる収束条件のもとで，C（パス全体）を有限の値に収束させることができる．

カバレジ関数CをC（パス全体）で割った関数を新たに$C^*=C/C$（パス全体）と定義すると，C^*には次のような性質がある．

(1) 任意の試験項目iについて

$$0 \leqq C^*(\text{試験項目 i}) \leqq 1 \qquad (3)$$

(2) C^*（フローグラフのパス全体）$=1$　　(4)

(3) 試験項目のパスが互いに排他的なときに，

$$C^*(\Sigma \text{試験項目}_i) = \Sigma C^*(\text{試験項目}_i) \qquad (5)$$

この3つの性質は，パスベクトルがコルモゴロフの公理[5]による確率空間をなすことを示す．ある試験項目のパスベクトル｛p_L｝が確率事象，そのパスベクトル｛p_L｝から求めたC^*（｛p_L｝）がフローグラフ｛P_L｝に対するカバレジである．

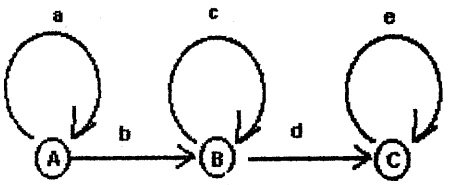
C^*の作り方から分るように，次の定理が成り立つ．

定理1

各リンクのフォールトの数が一定のとき，試験項目で発見できるエラーの数は，その試験項目のカバレジC^*に比例する．すなわち，

エラー総数＝試験項目iによる発見エラー数／試験項目iのカバレジC^*　　(6)

である．



Path vector= {5, 8, 12, 17, 23, ···L(L+3)/2+3··· •}

1:bd	7:bcde	13:abcccde
2:abd	8:abcde	14:abcdeee
3:bcd	9:aabcde	15:aabccde
4:bcde	10:abccde	16:aaabccde
5:abcd	11:abcdee	17:aabcccde
6:abde	12:aaabcde	18:abccdeee

Order of test cases

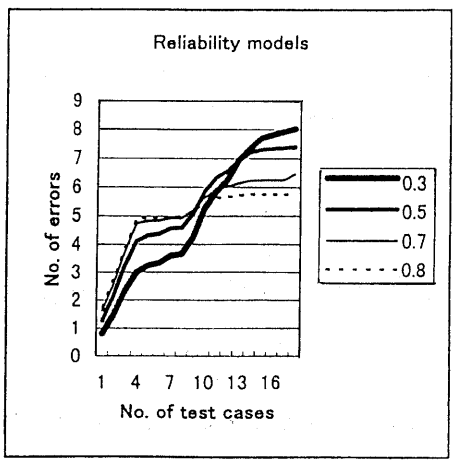


図2　パス分解法シミュレーション例

5　モデルのシミュレーション

前章のEを求めるには，rとf_a，f_b・・などが分らなければならない．実際にこの手法を使うときには，過去の経験から適当な値を仮定したり，試験で得られたフォールト数を式に入れたりすることになる．またすでに使用実績のあるモジュールをプログラムに組み込む場合には，そのリンクのフォールト数を0に近くすればよい．rについても，経験の浅い技術者がテストする部分にはrを小さくするなど工夫がいる．実際の使用法は今後の課題である．

ここで，より具体的に，図2を使ってシミュレーションでEを求めてみる．

単純化するために，先にrがリンクによらずすべて一定と仮定したが，それに加えて，フォールトはすべてのリンクに一様に1個づつあるとする．すなわち，(1)式で$f_a = f_b = f_c = 1$とする．この条件ではCとEは同じである．図2のようにループが互いに共通のノードを持たないフローグラフについては，$0 < r < 1$なる任意の値についてEが収束することが保証されているので[2][3]，エラー発見率rを0.3，0.5，0.7，0.8と変えてEを求める．試験項目は短いものから使用するとして，図2の左に示した順序に適用する．その発見エラーの累積値は図2の右図のようになる．グラフの横軸は試験項目番号で，それぞれの試験項目のパスは図2左にアルファベットで記述した．

これによると，$r = 0.3$，では階段状のカーブになり，rが増加するにつれて指数関数に似てくる．階段状の形状は試験項目の実施順序を変えると変化する．

階段状になる理由は次のように考えられる．エラー発見数は試験の開始時に多く，だんだん少なくなるという指数曲線に近い形状になる．この例の場合，試験項目の4番目と9番目のところで試験項目の長さが変化しているので，ここで曲線の勾配が変化したのである．また，試験前半のパスの長さの短い単純なエラーを発見する指数曲線と，後半の複雑なエラーを発見するという指数曲線の二つがあって，9番目の試験項目のところでつながっているとも解釈できる．このような形状は，指数－S字形信頼度成長曲線として資料[6]などに紹介されており，そこでは，修正の容易なエラーと困難なエラーが混在している場合に見られると説明されているが，本手法の結論とほぼ一致している．

エラー総数は，曲線の飽和値で表されるが，図2から分かるように，エラー総数は，$r = 0.3$，0.5，0.7，0.8と増えるにつれて減少する．$L = 20$までエラー数を求めると，それぞれ17.2，9.0，6.5，5.8となる．パス分解法の汚染モデルでは，フローに枝別れがあるとエラーが増殖すると考えるが，エラー発見率が大きいと，エラーを上流で発見除去するために，エラーの増加が抑えられると解釈される．ただしこの例では，前提として各リンクに1つのフォールトがあるとしたので，エラー総数が5より少なくなることはない．

このシミュレーションは，エラーは下流に広がらないうちに除去することが望ましいことを示し，多くのモジュールをまとめてテストするビッグバンテストが効率的でないことが分る．

6　従来の信頼度モデルとの関係

信頼度成長曲線については，S字カーブなどのいくつかのタイプが研究者の間で興味をもたれているが[7][8]，本モデルのシミュレーションを見ると，試験項目の順序や，試験担当者の技量やデバックツールなどに依存していろいろな形状が現れると考えられる．

ここでは本手法の試験モデルから従来の信頼度成長曲線を導き，その関係を調べる．

まず試験項目を一斉に投入するとすれば，エラーはパスの長さの短いものから発見され，長いパスのエラーはより短いパスのエラーに妨害され，短いものがなくならないと長いものは発見できないと考えられる．したがって，エラーはそれが含まれるパスの長さLの順に発見される．すなわち，Lと作業時間tとは比例して増加するとする．こうしてパス分解法のモデルに作業時間を導入する．

前提3

試験作業を開始してからエラーが発見されるまでの時間tは，発見されるエラーのパスの長さLに等しいとして、$L = t$とする．ここではLは離散的な値でなく連続的な値とする．また各リンクに含まれるフォールトの数は一定とし、fと記述する。

さらに、パスベクトルの性質について次の定理2を使う．証明は資料[1][2][3]参照．

定理2

パスベクトルとエラー発見率について

(1) フローグラフが図2のように，ループを含むが互いに共通のノードをもたない場合には，長さLのパスの個数P_Lは，Lが大なるときにLの多項式で増加する．また (2) 式は$0<r<1$の任意のrで収束する．

(2)図1のようにノードを共有する二つ以上のループをもつ場合には，Lが大なるときにP_LはLの指数関数で増加する。その関数を、$a\times b^L$と表すと、(2)式の値がL＝∞まで加えて収束するためには，

$$b\times(1-r)<1 \qquad (7)$$

でなければならない．すなわちエラー発見率rが，

$$r>1-1/b \qquad (8)$$

でなければならない．

6．1 指数形信頼度成長曲線

図1の場合，定理2の(2)により長さLのパスの数は$a\times b^L$という形で近似できる．したがって，エラー発見数を長さの短い方から加えて行くと，長さ1からLまでのエラー数は，(1)式$\times a\times b^L$をL＝1からLまで加えて，

$$\fallingdotseq F\times a\times r/(1-r)\times\sum_{I=1}^{L}\{b^I\times(1-r)^I\} \qquad (9)$$

とする．

rが (7) 式を満すときに，

$$b\times(1-r)\}=e^{-\alpha}<1 \qquad (10)$$

となるαを選ぶ．すなわち，

$-\alpha=\ln\{b\times(1-r)\}$ とする．

ここでLを連続変数と考えて，Σを∫におきかえると，(9) 式は次のように近似される．

$$\fallingdotseq F\times a\times r/(1-r)\times\int_0^L b^t\times(1-r)^t dt$$

$$=F\times a\times r/(1-r)\times\int_0^L e^{-\alpha t}dt$$

$$=F\times a\times r/(1-r)\times(1-e^{-\alpha L})/\alpha \qquad (11)$$

(11)式はもっとも基本的な信頼度成長曲線である指数形信頼度成長曲線の形である[4]．したがって，フローグラフがノードを共有する二つ以上のループをもち，エラー発見率が (7) 式を満たす場合には，発見エラー数は指数形信頼度成長曲線で近似できると言える．

6．2 修正指数形信頼度成長曲線

(11)式でエラーにエラー発見率の違う2種類がある場合には，修正指数形信頼度成長曲線となる[4]．パス分解法の考えでは，rが大きいエラーとは定義してすぐ使う変数などでエラーの発見までのパスの長さが短く，rが小さいエラーとはプログラムの前の方で定義して後の方で使う変数などでエラー発見までのパスの長さが長いものと説明できる．図2の例がこれに近い．すなわち，試験項目の8番目と9番目のところで試験項目の長さが変わり，そこでエラー発見数の勾配が変わっている．

6．3 遅延S字形信頼度成長曲線

フローグラフが互いに共通のノードをもたないループを含む場合には，定理2の(1)によりP_LはLの多項式で近似できる。特に，P_LがLの一次式で近似できる場合について考察する．この場合，0次の項を無視すれば，$P_L\fallingdotseq a\times L$と近似して，(10)式の場合と同じように

$$(1-r)=e^{-\alpha}<1 \qquad (12)$$

となるαを選び，(1)式$\times a\times L$をL＝1からLまで加え，それを積分に置き換え部分積分を使えば，信頼度成長曲線は

$$=f\times r/(1-r)\times\int_0^L\{a\times t\times(1-r)^t\}dt$$

$$=f\times r/(1-r)\times a\times\int_0^L(t\times e^{-\alpha t})dt$$

$$=f\times r/(1-r)\times a\times$$

$$[-t\times e^{-\alpha t}/\alpha-e^{-\alpha t}/\alpha^2]_0^L$$

$$= f \times r / (1-r) \times a / \alpha^2 \times$$

$$\{1-(1+\alpha \times L) \times e^{-\alpha L}\} \qquad (13)$$

となる．(13)式は，遅延S字形信頼度成長曲線の形である[4]．

P_Lが一次式でない場合でも，定理2から，一般にフローグラフが共通のノードをもたないループをもつとき，(13)式はLの多項式×$e^{-\alpha L}$という形で近似される．この値はLが小さいときにはLの多項式により下に凸に増加し，Lが大きくなると$e^{-\alpha L}$が効いてきて増加が抑えられる．すなわちS字型の曲線になる．

また(12)式から分るようにαの値はrに依存し，エラー発見率rが1から0に近づくにつれ，αは0に近づく．したがって(13)式の収束が遅くなる．

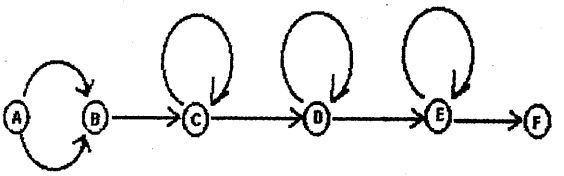
遅延S字型成長曲線については，エラーを発見してからそれを認知するまでに時間がかかるためにS字形が現れると説明されているが[7]，パス分解法の考えでは，プログラムのフローグラフがノードを共有するループを持たないような比較的単純な形で，エラー発見率rが小さいときに現れる現象であると解釈される．

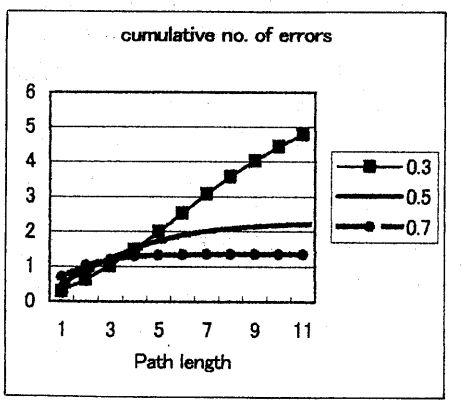
S字曲線のシミュレーション例として，図3のフローグラフについて，発見エラー数の変化を示す．計算式は，(1)式のfはすべて等しいとし，P_Lは連結行列から求めたものを使い，Lの小さい方からエラーが発見されるとして次式を使う．

長さLまでのエラーの発見数

$$= f \times \sum_{I=1}^{L} P_I \times \{r \times (1-r)^{I-1}\} \qquad (14)$$

rは0.3，0.5，0.7と変化させて，L＝11までの発見エラー数を示したのが図3である．これから，rが小さくなると，収束が遅くなり，あまり明確ではないが，r＝0.3のときにS字カーブをしていることが分る．





Path vector＝{9, 14, 24, 40, 64, 96, 136, 184, 240, 304, 376 ・・}

図3 パス分解法による遅延S字型信頼度成長曲線のシミュレーション例

7　まとめと今後の課題

本論文では，パスベクトルを使ってプログラムの複雑性を考慮した信頼度成長曲線を求め，シミュレーションによってグラフを描いた．

ただし，前提として，エラー発見率ｒが一定で，フォールト数がリンクによらず同じとしたが，このような仮定が現実的なものかどうかの確認などは，今後の課題である．そのために，データの収集、ツールの開発などが必要である．

ツールについては，パソコンで小規模なものを作成したので，今後紹介をする予定であるが，パスの数がＬの指数関数で急激に増加するので，その記録をどうやって保持するかなどが問題である．

パス分解法，すなわちプログラムのフローグラフを長さ別のパスに分解することは，たとえていえば，波形を時系列としてではなく振動数に分解することに似ている．

これによって次のような成果を得た．すなわち，

(1)プログラムの複雑度を，フローグラフのパスベクトルで表した．

(2)ループの形とパスベクトルの関係からプログラム構造とループの影響を示し，goto レスプログラミングの理論を拡張した．

(3)プログラム複雑度とエラー発見率を考慮した信頼度成長曲線のモデルを提案した．

(4)パスベクトルによるカバレジを定義し，テストカバレジと信頼性成長曲線を関連付けた．

(5)残存エラー数をコルモゴロフの理論と結びつけて，確率的に推定した．

このうち(1)と(2)については，論文[1]で述べ，(3)(4)(5)については本論文と論文[2]で述べた．これによって、ソフトウエア工学全般にわたって，体系的な理論を構築することができたと考えている．

参考文献

[1]若杉忠男：フローグラフや状態遷移図の特性のパス数による分析，情報処理学会論文誌，第40巻第２号 pp. 742-749(1999-2)

[2]若杉忠男：残存エラー数の推定が可能なプログラムの試験法 (4) －エラー数推定のための前提条件－，情報処理学会ソフトウエア工学研究会報告，98-SE-121, pp. 1-8(1998-11)

[3] 若杉忠男：残存エラー数の推定が可能なプログラムの試験法 (3) －パスカバレジに基づいたソフトウエア信頼性モデル－，情報処理学会ソフトウエア工学研究会報告，98-SE-118, pp. 17-134(1998-3)

[4]山田茂：ソフトウェア信頼性モデル－基礎と応用，P202，日科技連 (1994)．

[5]国沢清典：近代確率論，岩波全書，(1958)．

[6]木村，山田：２クラスの発見難易度をもつソフトウエアエラーを考慮した指数－Ｓ字形信頼度成長モデルの考察，「情報処理学会論文誌」Vol. 33, No. 11, pp. 1446-1452(1991-1)．

[7] 山田茂，大場充：エラー発見率に基づくＳ字型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, pp821-828, 1986-8.

[8] Yamada, S., Ohba, M., and Osaki, S.: S-shaped reliability growth modeling for software error detection. IEEE Trans. Reliability, Vol. R-33, No. 4, pp. 289-292, October 1984.