ソフトウェア工学　121－1
（1998. 11. 5）

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－

若杉忠男
若杉情報技術コンサルタントオフィス　湘南工科大，文教短大講師
連絡先：〒251-0033　神奈川県藤沢市片瀬山3-11-1
電話0466-23-4832
E-mail：FZC04503@nifty.ne.jp

あらまし　フローグラフを試験項目でカバーすることによりエラー総数を推定する方法について，この方法が成立するためのプログラムに必要な性質，対象とするエラーの種類，試験に課すべき条件を考察する．また無限の試験項目を有限で近似するアイデアと，エラー数の推定値の計算法を示し，その推定精度を“ある条件ｘを満たす試験環境のもとで，ある指標の値がｙであれば，エラー総数が推定値より多い危険率はｚ％以下である”という形で評価する．また試験しやすいプログラムの条件を考える．

キーワード　パス分解法，パスカバレジ，フローグラフ，プログラム試験，エラー数の推定

On a software test method capable of estimating the number of programming errors (4)
- The premises for error number estimation -

Tadao WAKASUGI, Member
WAKASUGI Information Technology Consultant Office
3-11-1 Kataseyama Fujisawa-city,251-0033 Japan
E-mail : FZC04503@nifty.ne.jp

Abstract

Concerning the method of estimating the number of errors of program on the basis of path coverage of flowgraph, necessary characters of program, kind of errors, and conditions of testing are discussed. The relation between coverage test and error numbers is presented. The error number is estimated in the form; " Under the condition x, if an index is y, then the level of significance of provability that the error number is greater than the estimated value is less than z ". And the conditions for an easy testable program are presented.

key words　path decomposition method, path coverage, flowgraph, program test, estimation of no. of errors

1　はじめに

筆者はフローグラフをパスの集合に分解し分析するパス分解法という手法を提案している．本論文はパス分解法により完全なカバレジ試験を近似的に実現するアイデアの紹介である．プログラムの試験については，“Non-exhaustive testing can be used to show the presence of bugs, but never to show their absence.”（完全な網羅試験でなければ，バグのあることは証明できてもバグのないことは証明できない）というダイクストラ氏の有名な言葉があるが[1]，本論文では，完全な網羅試験を有限個の試験項目で近似し，エラーの数を確率論的に推定することをこころみる．

試験対象には一般に無限回ループが存在する．たとえばコーヒーの自動販売機の試験を考えると，１杯目のコーヒーが無事に出たとしても，２杯続くと紙コップが接触してひっくり返るとか，５０杯も出せば原料が切れるとか考えられる．したがって事故がおきないことをテストで完全に確認するには，無限回のループの実行が必要である．

完全な網羅試験をするには二つの方法が考えられる．一つは完全な網羅試験ができるように，プログラム構造を単純化することである．たとえばプログラムにループをなくし，扱う数値の範囲を限定する．コーヒーの自動販売機の例でいうと，１杯づつしか出ず，１杯ごとに完全に初期状態にもどるように設計すれば完全な網羅試験ができる．ハードウエアの場合，少しづつ劣化し摩耗するから完全には初期状態に戻れないが，ソフトウェアでは戻れると考えられる．もう一つの方法は，有限の試験項目でも完全な網羅試験をしたと同じ効果があるということを示すことである．無限項の和，１＋１／２＋１／４＋・・が２になることが，はじめの数項の和から推定できるように，無限長で無限個の試験項目を有限長の有限個の試験項目から推定することである．

ここでは後者の考えに基づき先の論文[2][3]に続きフローグラフのパスカバレジ試験の理論を展開する．具体的には，“ある条件ｘを満たす試験環境のもとで，ある指標の値がｙであれば，エラー総数が推定値より多いという危険率はｚ％以下である”ということをいいたい．条件ｘを満たす試験環境をどう構築するかとか，指標ｙを測定するツールが作れるかということは今後の課題として，本論文では理論的考察を述べる．

2　モデル

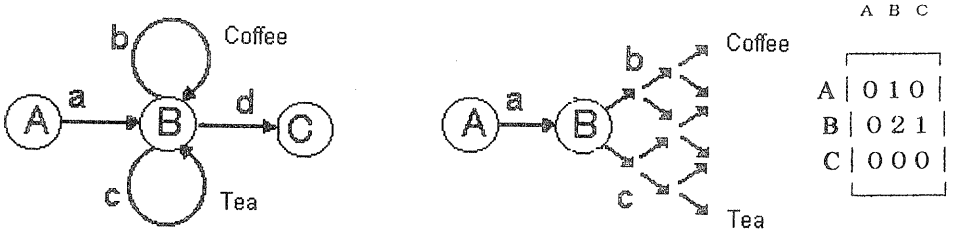
2．1　プログラムモデル

まず次の定義を行う．

定義１　フローグラフ

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適当な切れ目や分岐点を表す[4]．

図１の左にフローグラフの簡単な例を示す．ＡＢＣはノードを，ａｂｃｄはリンクを表す．リンクａは自動販売機の起動処理，ｂとｃはそれぞれコーヒーと紅茶の出力，ｄは自動販売機のリセット処理である．まん中の図はパ



フローグラフ　　パスツリー　　連結行列

図１　ループが二つあるフローグラフとそのパスツリー，連結行列

スツリーと呼び，フローグラフのパスを展開したもので，リンクａを経由したパスが上に行けばコーヒー，下に行けば紅茶を出すことを示す．この図では３杯までしか示していないが，パスの数はＬの増加と共に急速に増加する．また右図はこのフローグラフの連結行列を表す[4]．このマトリックスのｉ行ｊ列の要素がｋということは，ノードｉからノードｊにリンクがｋ本あることを示す．図１のマトリックスの２行２列の要素が２ということは，ノードＢにループが二つあることを示す．

フローグラフから次に説明するパスベクトルが導かれる．

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のパスと定義する．フローグラフのパスの長さＬのパスの個数をP_Lと記述し，またそれらを長さ順に並べたものをパスベクトルと呼び｛P_L｝や｛p_L｝などと記述する[2]．プログラムのフローグラフは｛P_L｝，それを試験項目でカバーした部分集合を｛p_L｝などとする．ただし同じパスがあるときは重複しては数えない．ここで同じパスとは，先頭のリンクも，経路も，長さも，終わりのリンクも同じパスのことである．

たとえば図１でコーヒーを３杯出す試験ａｂｂｂｄを考えると，長さ１のパスはａ，ｂ，ｂ，ｂ，ｄの５個であるが，ｂが重複するから省いて，ａ，ｂ，ｄの３個とする．また長さ２のパスはａｂ，ｂｂ，ｂｄの３個となり，以下同様にしてパスベクトルは｛３，３，３，２，１｝となる．一方，図１のフローグラフ全体では，長さ１のパスは４本（ａ，ｂ，ｃ，ｄ），長さ２のパスは９本（ａｂ，ａｃ，ａｄ，ｂｂ，ｂｄ，ｂｃ，ｃｂ，ｃｃ，ｃｄ），以下同様にしてパスベクトルは｛4, 9, 18, ・・, 2.25×2^L, ・・｝で，Ｌ＞１なるＬについて２．２５×2^L本となる．一般にＬが大になるとP_Lには簡単な規則性が現れる[2]．

パスベクトルの初めのいくつかのパスを試験項目でカバーするだけで，パス全体のエラーの推定をしたい．そのために，次のような前提条件を考える．

前提１　対象とするプログラムの条件

（１）プログラムはフローグラフで表わされる．

（２）プログラムに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，またループは無限回実行できるものとする．条件の組合せによっては通らないパスもありうるが，問題を単純化するために通れないパスはないとする．

２．２　エラーモデル

エラーについては次のように考える．

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．試験項目のパスを水道にたとえると，上流での異物の混入がフォールトで，下流で検出された汚染がエラーである．前のリンクと後のリンクに不整合が発見された場合には，実際にどちらのリンクを修正するかにかかわらず，前のリンクをフォールト，後のリンクをエラーとする．

前提２　フォールトとエラーについての条件

（１）試験項目をパスに適用した場合，あるリンクにフォールトがあれば，リンクの下流のパスのどこかに必ずエラーが生ずるとする．エラーを引き起こさないフォールトはないとみなす．

したがって，Ｘ＝０と記述すべきところを，Ｘ＝ＳＩＮ（π）と記述しても，結果的にエラーは生じないので，これはフォールトとしない．また定義した変数をどこにも使わないケースなどもフォールトではない

（２）エラーとは，対象とするフローグラフからの出力が期待するものと異なることであり，したがって次のようなものは含まない．

①フローグラフの形が変わってしまう場合．たとえばエラーの修正によりブランチの行き先の変更やリンクの挿入などでフローグラフの形が変われば，試験をやり直すとする．

②フローグラフに表されないもの，たとえば，計算結果の精度，プログラムの使いやすさ，応答時間，あるいはプログラム仕様書の誤り．

（3）二つのエラーがキャンセルしあって，パスの途中でエラーが消えるということはないとする．

（4）任意のリンクについて，試験してエラーが起きる場合には必ず起きるものとし，データの値によってエラーが生じたり生じなかったりということはないとする．そういう場合には，フローグラフが粗すぎたのである．たとえば，あるリンクの処理で，ある計算式が，パラメータ＜０と＝０と＞０では違うというような場合には，フローグラフのリンクは１本でなく，マイナスと０とプラスの３つの場合に分けて３本にしなければならない．

（5）エラーとフォールトの数え方については次のようにする．

①ひとつのリンク内に同じフォールトから発生したエラーがいくつか見つかった場合には，エラーはまとめて１個と数える．これによって一つのリンク内のエラーの発見数はフォールトの数より多くはならず，したがって０≦ｒ≦１となって，収束が保証される．

②ひとつのエラーの原因がｎ個のフォールトの影響である場合には，このエラーの数はｎ個のフォールトの合成として，１でなくｎと数える．

③エラーを発見してその原因であるフォールトがすでに修正したことのあるフォールトであればまた修正し，一つのフォールトからもう一つエラーが出たと数える．

②と③の条件によりフォールト数≦エラー数となる．

以上の条件から，プログラムを構成するすべてのパスについて１回の試験をしてエラーが見つからなければ，そのプログラムにはフォールトはなく，一つのパスについて２回以上試験する必要はない．

定義４　エラーの長さ

エラーがその原因であるフォールトと同じリンクで発見された場合は長さ１のエラー，下流のＬ個目のリンクで発見された場合は長さＬのエラーという．

定義５　リンク当りエラー発見率

あるフォールトから出たエラーがリンク内で発見される割合をリンク当りエラー発見率といいｒと記す．

２．３　試験モデル

前提３　試験のやり方

（1）一つの試験項目を走らせてエラーを発見した場合にそのエラーを除去し，再度同じ試験項目を走らせてエラーがなくなったことを確認してから次の試験項目を走らせる．

（2）試験項目はパスの短いものから網羅的に実施するようにこころがける．またプログラムの一部を試験しないというような偏りはないとする．

（3）各リンクにおけるエラー発見率は一定とする．

（4）エラーを発見して修正した場合に，ミスにより新たにフォールトを作り込むということはないとする．

（5）リンク内のフォールトを修正すると，そのエラーを発見したリンク以降のパスでは同じフォールトからのエラーはすべてなくなるが，フォールトが存在するリンクとエラーを発見したリンクの間から出たパスのエラーまではなくならないとする[3]．例えば図１のフローグラフで，リンクａにあるフォールトでリンクｂにエラーが生じた場合，それをなくすためにリンクａを修正するとリンクｂのエラーはなくなるが，リンクｃのエラーのおきる確率は変わらないとする．

３　パスカバレジとエラーの関係

３．１　パスが１本の場合

あるリンクに含まれるフォールトの数をｆ個，エラー発見率をｒとすると，リンク内でのエラー発見数はｆ×ｒとなる．そのリンクで見逃し

たエラーは下流の次のリンクに伝わり，そこでの発見数はｆ×ｒ×（１－ｒ）となる．以下同様にして，長さＬのエラー数＝ｆ×ｒ×（１－ｒ）$^{L-1}$となるから，

長さＬのパスのエラー数

$$= f \times r \times \sum_{i=1}^{L} (1-r)^{i-1}$$

$$= f \times (1-(1-r)^{L}) \qquad (1)$$

（１－ｒ）＜１であるから(1)式は収束し，試験をパスの長さＬで打ち切った場合，

$$\text{残存エラー数} = \sum_{i=L+1}^{\infty} e_i = f \times (1-r)^{L} \qquad (2)$$

となる．

これから次のことが言える．

（１）発見されるエラーの数は，最初のリンクにあるフォールトの数と，試験項目がカバーするパスベクトルと，エラー発見率ｒに依存し，その期待値は(1)式で決まる．

（２）フォールトの影響は下流に伝わり，遠ざかるにしたがってエラー発見期待値は小さくなる．したがってあるていどから先の項目は無視できる．

たとえば一つのループを３回繰り返す試験項目のパスベクトルを考える．長さ１のパスは３個であるが同じものは１回しか数えないので１とする．同様に長さ２は１個，長さ３も１であるから，パスベクトルは{１，１，１，０，０・・}と表される．それに重み（１－ｒ）^{L}＜１がかかるので，長くなるほど小さくなる．

３．２　一般の試験項目

実際の試験では，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．図１の例で説明すると，コーヒーを３杯出す試験項目のパスはａｂｂｂｄである．エラーはリンクａ，ｂ，ｄにあるｆ$_a$個，ｆ$_b$個，ｆ$_d$個のフォールトから発生すると考え，試験項目の通るパスを列挙すると，ａを先頭とするパスはａｂｂｂｄで長さ５，ｂを先頭とするパスはｂｂｂｄで長さ４，ｄを先頭とするパスは長さ１である．したがってこの試験項目に番号を１とつけると，発見されるエラー数の期待値ｅ$_1$は，(1)式から次のようになる．

$$e_1 = f_a \times (1-(1-r)^{5})$$

$$+ f_b \times (1-(1-r)^{4}) + f_d \times r \qquad (3)$$

この(3)式の右辺を，試験項目のカバレジ関数Ｃ（試験項目$_1$）と記述する．試験項目がいくつもある場合はそれらを合計して，

$$\Sigma e_i = C(\Sigma \text{試験項目}_i) \qquad (4)$$

とする．ただし試験項目に同じパスがある場合には，重複分は削除する．

４　エラー数の推定と推定誤差

４．１　エラー総数の期待値

ここで(4)式が収束するかどうかを検討する．たとえば図１のノードＢから出るパスを考えると，長さＬのパスの本数はコーヒーと紅茶の２種の中からＬ個のものを選ぶ順列の数となり，２^{L}本である．したがって１本のパスの残存エラー数が１／２^{L}でも，フローグラフ全体では長さＬのエラー数の期待値は２^{L}／２^{L}＝１になり，いくら試験をしても残存エラー数は０にならない．

エラー総数の期待値をｅ*とすると，(4)式の記法で

$$e^{*} = C(\text{完全網羅試験項目}) \qquad (5)$$

となる．(5)式をパスの長さ順に並べると，

$$e^{*} = C(\sum_{L=1}^{\infty} \text{長さＬのパス}) \qquad (6)$$

となる．(5)と(6)は計算順序が違うだけで同じ内容である．(6)式を具体的に記述すると，ｍをリンクの数，ｊをリンクの番号，ｉをパスの長さ，ｐ$_{ji}$をリンクｊを先頭とする長さｉのパス，ｆ$_j$をリンクｊにあるフォールトの数として，次の

式で表される．

$$e^{*} = \sum_{i=1}^{\infty}\sum_{j=1}^{m}\{f_j \times p_{ji} \times r \times (1-r)^{i-1}\} \quad (7)$$

図１について(7)式をマトリックスで記述すると，(8)式あるいは図２のようになる．ここに左辺のマトリックスは各リンクからの発生エラーを表すマトリックスでＥと記述する．右辺のマトリックスは図１にも示したがフローグラフを表しＭと記述する．右辺の左端は各リンクのフォールトの数ｆをＭに代入したものでＦとし，またＭ^{0}が単位マトリックスを表す．

$$E = F \times r \times \sum_{L=1}^{\infty}\{(1-r) \times M\}^{L-1} \quad (8)$$

縦軸に累積エラー数，横軸に試験したパスの長さを目盛ったグラフを描くと，(7)式は指数関数の和であり，これが収束すれば，Ｙ＝総エラー数が漸近線となる．実際の試験では長さ３のパスの試験をすれば，必然的に長さ１，２のパスの試験もすることになるから，長さ１だけの試験とか長さ２だけの試験などはできない．したがって横軸を長さにするグラフは描けないが，しかし前提３で述べたように，パスの短い方から試験すれば，e^{*}に飽和するグラフが描ける．したがってエラー総数の推定値が読み取れる．

４．２　収束条件

ここで発見エラー数の期待値が収束する条件を示す[3]．

定理１

$\{P_L\}$ をフローグラフのパスベクトルとすると，Ｌが大なる場合，

$$P_L \leqq a \times b^{L} \quad (9)$$

となるａとｂが存在する．このｂとエラー発見率ｒとの間に次の(10)式の関係があると，パスカバレジ試験でエラーを減らすことができる．(10)式の関係が成立しない場合は，試験をしてもエラーがなくなるとは保証されない．

$$b \times (1-r) < 1 \quad (10)$$

証明省略．

(9)(10)式の証明は資料[2][3]を参照のこと．たとえばａ＝ノード数，ｂ＝一つのノードから出るリンクの数の最大本数，とすると(9)式は成立する．

定理１のａ，ｂを図１のフローグラフについて考えると，ノード数＝３，一つのノードから出るリンクの数の最大本数＝３であるが，２章で示したようにＬ＞１では$P_L = 2.25 \times 2^{L}$となるから，ａ＝２．２５，ｂ＝２とすることができる．したがって(10)式から０．５＜ｒであれば試験を続けることにより残存エラー数は０に近付く．

一つのリンクｊから出るパスベクトルを$\{p_{jL}\}$とし，長さの短いパスから試験してゆくと，発見エラー数は(1)式に(9)式の条件を代入して，

$$\text{リンクｊから出る長さＬのエラー数} = f_j \times r \times p_{jL} \times (1-r)^{L-1} \leqq f_j \times r \times a \times b \times \{b \times (1-r)\}^{L-1} \quad (11)$$

となる．(10)式の条件が満たされる場合は，ｂ×（１－ｒ）＜１であるから，(11)式右辺をＬ＝１

$$\begin{bmatrix} 0 & e_a & 0 \\ 0 & e_b+e_c & e_d \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & f_a & 0 \\ 0 & f_b+f_c & f_d \\ 0 & 0 & 0 \end{bmatrix} \times r \times \sum_{L=1}^{\infty}\left\{(1-r)\times \begin{bmatrix} 0 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 0 \end{bmatrix}\right\}^{L-1}$$

図２　(8)式のマトリックス表現

から∞まで加えた結果は収束し，したがって左辺も収束し次の定理が成り立つ．

定理２

条件(10)式が成立する場合は(7)式は収束してｅ*が得られる．それを長さＬまでで打ち切った残存エラー数の期待値は(12)式で表される．ただしｍはリンクの数である．

残存エラー数

$$=\sum_{j=1}^{m}\{f_j\times r\times\sum_{i=L+1}^{\infty}\{p_{ji}\times(1-r)^{i-1}\}\} \qquad (12)$$

証明省略

実際のｒの求め方は試験を実施して得たエラーの数を(4)式左辺に代入する．'ｒは０＜ｒ＜１なので(4)式右辺のｒに適当な値を代入して試行錯誤的に求められる．このｒを(7)式に代入して適当に打ち切ればｅ*が求められ，その誤差の期待値が(12)である．

４．３　エラー総数の推定精度

定理２の(12)式の元である(4)式の誤差を考える．

（１）(4)式が実際に成り立つかどうかについては，成立するという前提で理論を構築したので，成立するような環境を整えることとし，その誤差の分析は今後の課題とする．

（２）(4)式を適当に打ち切ることにより生じる誤差については，(12)式右辺が１より小さくなるＬまで試験すればよい．(13)式である．

(12)式右辺

$$\leqq\sum_{j=1}^{m}\{f_j\times r\times a\times b\times\{b\times(1-r)\}^{L-1}\}<1 \qquad (13)$$

（３）(4)式のｆ$_j$の誤差について考察する．あるリンクを試験するのを忘れたとすると，当然そのリンクにあるフォールトから出るエラーが発見できない．したがって，試験はそのような偏りはなく実施するものとする．リンクｊにフォールトが１個あって長さＬまで試験したにもかかわらずそれから生じるエラーが発見できないとすると，その確率は（１－ｒ）^{L}である．パス数はＰ$_L$本で，それを危険率ｚで見逃すとすれば次式が得られる．

フォールトを見逃す期待値

$$=P_L\times(1-r)^L\leqq z \qquad (14)$$

(14)の左辺はＬを増やせば０に近付くので，これがｚ以下となるＬまで試験すればよい．このこれから次の定理が成立する．

定理３

(4)式が成立するとき，エラー総数の推定値ｅ*は(7)式で表され，(13)式と(14)式を満たすＬまでパスカバレジ試験をすれば，真のエラー数がｅ*＋１以上である危険率ｚ％である．

証明省略

例を図１によって説明する．リンク数ｍが４で，エラーを１個見逃す危険率を１／２０以下としてｚ＝０．０５，Ｐ$_L$＝２．２５×２^{L}である．また各リンクに１個づつフォールトがあるとし，ｒの値は＝３／４とする．条件(13)式からＬ≧５，また条件(14)式からＬ≧６となり，Ｌ＝６まで計算すればよいことが分かる．これはコーヒー，紅茶の組合せを６杯まで試験するということであり，試験項目は２^6＝６４ケースほど必要になる．図１の見かけが単純なのにもかかわらず必要な試験項目の数が多いが，このフローグラフは二つのループがノードBで分岐しているために２^{L}個の試験項目が必要となる．もしループｃをｂと切り離してノードCにもっていくことができれば，試験項目の数は$_2H_L$個＝（Ｌ＋１）×Ｌ／２＝２１個ていどでよい [2]．またｒが１に近付けばＬは減る．逆にｒが０．５に近付けばＬは増加し，場合によっては無限大となり試験は不可能になる．

５　試験しやすいプログラム

本論文は(4)式が成立するという前提で議論をすすめているが，(4)式がどの程度の精度で成立するかは確率的な問題である．本手法を適用し

てエラー数をうまく推定できたからといってすべての例に適用できるといは言えないし，またうまく推定できなくても，本手法が使いものにならないとも言えない．

ここでは(4)式が成立するようなプログラムの条件を考える．本手法が適合するかどうかではなく，手法に適合するようにプログラムを作ろうという考えである．それは①パスの本数が少なく，②パスの長さが短く，また③ｒが１に近いものである．これは次のように作られたものである．

（1）ループやブランチを少なくする

ループが多いと必要な試験項目の数と長さが増える．特にいくつかのループが共通のノードを持つと，パスの数はパスの長さの指数で増加するので[2]，ループの中にループがあるとか，ブランチとループを組み合わせるとかいうことはなるべく避ける．ＧＯＴＯ文の多用とか，ループへ飛び込んで飛び出すのもよくない．ＧＯＴＯレスプログラム[5]は望ましい．

（2）並列化やモジュール化する

フローは並列にできるならば並列にしてスタートからエンドまでのパスを短くする．またモジュール化して一つのプログラムを小さくするのが望ましい．互いに独立なモジュールに分割して大きなプログラムを小さくし，試験はモジュール単位に実施するとよい．

（3）隣接したリンクとの間の関係を強くする

あるリンクの実行結果は，すぐ下流のリンクで使用するように作るとｒが１に近くなり，発見エラー数の収束が速くなる．プログラムの最初の方で定義した変数を遠く離れた後ろの部分で使用するというようなプログラムは望ましくない．複合化設計でいうモジュール強度（Strength）の強い，すなわち内部の関連性の強いモジュール[6]や，オブジェクト指向設計でいうカプセル化されたモジュールが望ましい．

6　まとめ

パス分解法に基づくエラー数の推定のための理論を述べた．すなわち，①本手法の成立する前提条件を明確にし，②無限長のパスを有限長で近似する方法を述べ，③エラー数の推定と誤差評価を示し，④本手法の適用しやすいプログラムの条件を示した．

１章で述べた，“ある条件ｘを満たす試験環境の中で，指標値がｙであれば，エラー総数が推定値より多い危険率はｚ％以下である”という命題において，ｘは前提１～３と５章であり，指標ｙは定理３である．

本手法に都合のよいプログラムの条件を５章に述べたが，この条件は，従来から言われてきたＧＯＴＯレスプログラミングや複合化設計などと矛盾しない．これはパス分解法が，単なるエラー数推定のための一手法ではなく，プログラム設計論やソフトウェアメトリックや試験手法などソフトウェア工学の諸問題を広範囲にカバーする理論体系であることを示す．

ここで基本とした(4)式についての誤差解析と本手法の実用性の評価は，今後の課題とする．

参考文献

[1] Dijkstra, E.W.:On a Political Pamphlet from the Middle Ages, ACM SIGSOFT Software Engineering Notes 3-2,14-18,1978.

[2]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，情報処理学会マルチメディアと分散処理研究会報告，98-DPS-86,pp61-66(1998-1-29)

[3]若杉忠男：残存フォールト数の推定が可能なソフトウェア試験法について(3)，情報処理学会ソフトウェア工学　研究会報告,98-SE-118,PP127-134(1998-3).

[4] Beizer,B.: Software Testing Techniques, P.442, 小野間,山浦訳,日経ＢＰ出版センター（1994）.

[5] Dijkstra:Goto Statement Considered Harmful, CACM 11-3, 147-148,1968.

[6] Myers, G. J.: Reliable Software through Composite Design, Marson/ Charter Publishers, 1975.