ソフトウェア工学 112－2
（1996．11．15）

残存エラー数の推定が可能なソフトウェア試験法について

若杉忠男
若杉情報技術コンサルタントオフィス　　電話0466(23)4832

プログラムの開発上のミスをフォールト，その結果の期待に反する現象をエラーと呼ぶ．フローグラフをリンクに分解し，リンクあたりの平均エラー発見率をｒとし，あるリンクに存在するフォールトはその下流で発見できるとすると，Ｌ個下流のリンクで発見できる確率は$(1-r)^{L-1}\times r$となり，フォールトの存在位置とエラーの発見位置との差からエラーの確率分布が定まる．パスの枝別れによって３つのモデルを考え，試験で発見したエラーの数とそのフォールトの位置から確率分布を求め，試験の効果，見逃しエラー数の推定を行う．これによって，パスと複雑度の関係，パスカバレジ試験の意味，枝別れとエラー数の関係などを説明する．

On a software test method capable of estimating numbers of program errors

Tadao Wakasugi
Wakasugi Information technology consultant office

In testing, we analysis a flow graph into paths of links, and assumes the average error detectability rate per a link is r. The distance between the location where a fault exists and location where the error detected shows the probability distribution of error. By using some fault models and from detected numbers of errors, the method for estimating the r and total number of faults are discussed. Complexity of systems, meanings of paths coverage, convergence conditions of numbers of errors and some associated problems are considered.

1　はじめに

筆者は，先に状態遷移図をパスに分解して試験するという方法を提案したが[1]，本論文ではそれを一般システムまたはプログラムに適用し，フローグラフと関係付けて解析し理論付けを試みた．なお資料[1]の５章ではフォールトとエラーの区別が不明確であったので，本論文で明確にする．関連の研究として，パスカバレジ[2]，パスによる複雑度[3]，その応用例[4]がある．

2　フォールトモデル

まずソフトウエア試験のフォールトモデルを考える．フォールトモデルとは，フォールトの状態と試験の実施手順をシミュレートするモデルである．ここではプログラムの間違った作り方をフォールトと呼び，その結果生じた予期しない現象をエラーと呼ぶ．フォールトは原因でエラーはその結果である．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病原菌がエラーである．

まず，前提条件として，プログラムをブロックまたはサブルーチンに分割する．各

ブロックはほぼ同じていどにフォールトを含むように分割してあるものとする．これを図で表現したものをプログラムのフローグラフと呼ぶが，図１に示すように，ブロックを矢印で表してリンクと呼び，リンクとリンクの句切りに○を描いてノードと呼ぶ．またリンクのシーケンスをパスと呼ぶ．１個のリンクを長さ１のパスと考え，Ｌ個並んだリンクは長さＬのパスという．ひとつのリンクを反復実行したり，いくつものリンクの間をループすることもあるので，パスの長さは無限となりうる．

ここで次のような仮定をする．

（１）リンクにフォールトがあった場合，その結果であるエラーはそのリンクを先頭にもったパスのどこかに現れる．

（２）どのパスからもエラーが発見されない場合にはフォールトはないものとする．

各リンクの終わりにチェックポイントをおき，同程度の丁寧さでチェックを行う．エラーを各リンクの終わりで発見する率をリンク当りエラー発見率ｒと呼ぶ．そこで見逃したエラーが次のリンク，すなわち長さ２のパスの終わりで発見される確率は$(1-r)\times r$であり，長さＬのパスの終わりで発見する確率は$(1-r)^{L-1}\times r$となる．したがってフォールトの存在するリンクとそのエラーが発見されたパスの長さがＬ個の場合，すなわちＬ回のチェックののちに発見された場合のエラーの個数をＥL個と表すとこれは幾何分布にしたがう．

ここでもしＥ1とＥ2，すなわち２個の連続するリンクの組合せまでをすべて試験してそのエラーの個数を求めると，ｒが定まりしたがって分布の形が定まり，総エラー数を推定することができる．

バグの数の集計と分類については多くの例があり収集のためのツールもあるが，存在位置と発見位置とで分類した例は知らない．

図１　フローグラフの例１

３　フォールトモデルとパスカバレジ

プログラムの試験法についてパスカバレジという考え方がある[2]．すべてのリンクを一通り試験をすることをここでは長さ１のパスカバレジと呼び，連続するＬ個のリンクの組合せすべてを試験することを長さＬのパスカバレジと呼ぶ．図１のような枝別れのないプログラムでは長さ１でも２でも試験項目の数は同じようなものであるが，図２のような枝別れのある例を考えると，長さ１では６ケース，長さ２は３×３＝９ケースの試験項目が必要になる．プログラムが複雑になりパスの長さが長くなるとパスの個数は急増する[3]．

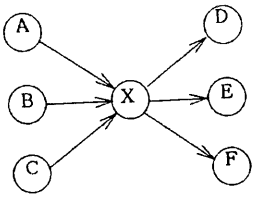


図２　フローグラフの例２

ここで枝別れのある場合の試験について，フォールトの影響と試験の効果の組合せによって，次の３つのタイプを考える．

（１）あるリンクに生じたフォールトは下流のパスの全てにエラーを生じ，それを見つけて修正するとそのフォールトによるエラーはすべて消える．

（２）リンクに生じたフォールトは下流のパスのどれか一本にだけエラーを生じ，それを見つけて修正すると，そのフォールトのエラーはすべてなくなる．

（３）リンクに生じたフォールトは下流のパスの全てにエラーを生じ，それを見つ

けて修正すると，そのエラー発見場所の下流については発見したフォールトによるエラーはすべて消える．しかしエラー発見場所の下流以外についてはフォールトを除去してもその影響は消えない．図３の例でＡのフォールトをＣで発見するとＣ以降のパスからはエラーは除去されるが，Ｅ以下にはエラーが残っているとする．フォールトを修正するとそのために別のフォールトを作ってしまうということがあるので，フォールトを除去しても他の支流についてその影響が除去されたかどうかは別途チェックしなければならないと考える．

水道にたとえて説明すると，（１）は上流で病原菌が混入したのを下流で検出し，混入場所を突き止めてふさぐことに当たる．（２）は上流で１個の異物が混入したのを下流で網をはって取り除いたという例であり，汚染が全体に広がらないとする．（３）は病原菌を検出したので検出場所で消毒剤を混ぜて下流への被害を防いだが他の支流は検査していないというような例である．

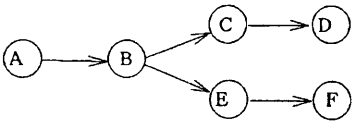
図３によって，長さ２までのパスをすべて試験した場合の長さ３のパスに与える影響，すなわち長さ３のパスへのエラーの見逃し率を考える．

タイプ１のエラーの見逃し率は，Ｂ，ＣおよびＥの３地点すべてで見逃す確率で，$(1-r)^3$となる．これは楽観的過ぎて現実と合わないと考えられる．

タイプ２の場合はエラーがＢ→Ｃまたは，Ｂ→Ｅで生ずる確率をそれぞれ1/2として，$(1-r)\times(1/2+1/2)\times(1-r)=(1-r)^2$となる．このタイプではフォールトの数とエラーの数は等しくなる．

タイプ３のエラーはＢで見逃しかつＣまたはＥでまた見逃す確率で$(1-r)\times\{(1-r)+(1-r)\}=2\times(1-r)^2$となる．このタイプでは，エラーの数は枝別れの数の増加に伴い増加する．

以降ではタイプ２と３について考える．



ﾊﾟｽ長さ	1	2
ﾊﾟｽが1本の場合のｴﾗｰ発見率	r	r^2

図３　試験の例

４　タイプ２のエラー見逃し率

タイプ２はフォールトとエラーは１対１なので単純で扱いやすい．ｒをリンク当り平均エラー発見率とし０＜ｒ＜１と考える．以降では単にｒと呼ぶ．この仮定のもとで次の定理が成り立つ．

定理１

タイプ２の条件のもとで，フローグラフに含まれているフォールトの総数をＦ個とし，フローグラフをパスの短い方から網羅するように試験すると，

（１）長さＬまでのパスを100%試験項目でカバーすると，パス当り平均エラー発見率ＹＬおよびその見逃し率ＺＬは

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

$$ZL=(1-r)^{L} \qquad (2)$$

$$ZL\text{-}1=ZL+YL \qquad (3)$$

となる．これらを以降では単にエラー発見率とか見逃し率という．

（２）したがって試験するパスの長さを増やすと，エラー数は０に近付く．

（３）長さＬのパスを100%カバーして発見したエラー数をＥＬとすると，次の式が成立する．

$$F \times YL = EL \qquad (4)$$

$$F \times (1 - ZL) = E1 + E2 + \cdots + EL \qquad (5)$$

証明省略

これから次が成り立つ.

定理2

rと全体のフォールト数Fは

$$0 < r = 1 - E2/E1 < 1 \qquad (6)$$

$$F = E1/r \qquad (7)$$

となる. また次の関係が成立する.

$$E1 > E2 > E3 > \cdots \rightarrow 0 \qquad (8)$$

証明

(5)式のLに1と2を代入して, 連立させてFを消去すると

$r = 1 - E2/E1 = (E1 - E2)/E1$

が得られ, (6)が得られる.

(5)式でL＝1として(7)式が得られる.

また, (1)式(4)式から

$$F \times (1 - r)^{L-1} \times r = EL$$

$$F \times (1 - r)^{L} \times r = EL+1$$

これから, $0 < 1 - r = EL+1/EL < 1$ となり, (8)式が成立する.

証明終了.

図4に(1)式のパス当りエラー発見率YLを示す.

(5)式でL→∞とするとΣEL＝Fとなるが, これはフォールトの個数と発見したエラーの個数が等しいということで, このモデルの前提条件から当然である.

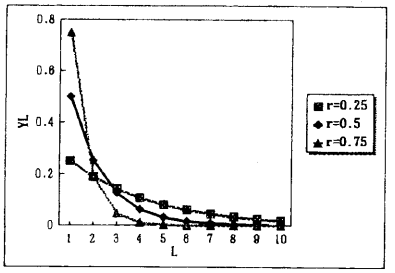


図4　パス当りエラー発見率YL(ﾀｲﾌﾟ2)

ここで幾何分布の平均値は1／rなので, rを(7)式から求める代わりに, 適当にランダムに試験項目を選んで実施し求めたE1, E2, ・・の平均値の逆数

$$r = \Sigma(EL) / \Sigma(L \times EL) \qquad (9)$$

からrを求めることもできる. この場合には幾何分布の分散は$(1 - r)/r^2$で表されるので, 発見エラー数の分散は

$$E1 \times (1 - r)/r^2 \qquad (10)$$

で表される. これはr＝0.5でE1×2となり, r＝0.2でE1×20となるから, この推定値のばらつきは相当大きい.

一通りの試験をしたあとで, 見つかったフォールトの合計に, 試験対象の複雑さや開発者のレベルなどを考えた倍率を掛けてバグ数を推定するという方法は, よく使われる経験則である. (7)式でrに0.5を代入するとF＝E1×2となり, 0.2を代入するとF＝E1×5となる. すなわち, rは担当者の実力を表す値で, (7)式はそれに基づく見積式と考えられる.

このフォールトタイプ2では試験をすればエラーは必ず減少する. また(1)式で表されるエラー発見率は長さLにのみ依存するので, プログラムのエラーはステップ数に

比例すると考えている．このタイプではプログラムの開発労力はステップ数に比例するということを仮定しており，これはあまり現実的でないと考えられる[5]．

5　タイプ３のエラー見逃し率の推定

タイプ３については一部をすでに発表したが[1]，ここではその一部を訂正してその後の考察結果を紹介する．

このタイプでは最初に存在したフォールト数をＦ個とすると，それから生ずるエラーの個数はパスの枝別れに伴って伝播して変化する．

（１）ＰＬを長さＬのパスの個数とし長さＬのパス当りのエラー発見率とエラー見逃し率をそれぞれＹＬ，ＺＬとすると

$$YL = PL \times (1-r)^{L-1} r / P1 \quad (11)$$

$$ZL = PL \times (1-r)^{L} / P1 \quad (12)$$

$$ZL\text{-}1 = ZL + YL \quad (13)$$

で表される．ＰＬはプログラムの複雑度を表す指標となる．(13)式は(3)式と等しい．

（２）このモデルではエラー数は一定の数ではなくパスの枝別れが増えるにつれて多くなり，逆にリンク当りエラー発見率ｒが大きいと抑制されて少なくなる．したがってパスが短いときに丁寧に試験をすべきで，パスが長くなってから試験すると苦労が多い．いわゆるビッグバンテストの効率が悪いことがこれで理由づけされる．リンクをサブルーチンと見るとその組合せの数をなるべく少なくする方がテスト効率の向上になる．

長さＬのパスの試験による発見エラー数ＥＬは，フォールト数をＦとすると，

$$EL = F \times YL$$

$$= F \times PL \times (1-r)^{L-1} \times r / P1 \quad (14)$$

となる．(14)式で特にＬ＝１，２とすると，

$$E1 = F \times r \quad (15)$$

$$E2 = F \times P2 \times (1-r) \times r / P1 \quad (16)$$

これから

$$r = 1 - E2 \times P1 / (P2 \times E1) \quad (17)$$

これはフォールトタイプ２の(6)式に対応する式で，Ｐ2，Ｐ1が既知で，かつ長さ１と２のパスを完全に試験して発見したエラーの個数Ｅ1とＥ2が求まればｒが求められる．(17)式で，もしＰ2／Ｐ1＜Ｅ2／Ｅ1，すなわちパスの数の増加以上に発見されるエラーの数が多い場合には，リンク当りフォールト発見率ｒはマイナスとなる．これは試験をするとかえってエラーが増えることを意味し，現実にはないはずである．

(6)(7)式と(17)(15)式が互いに対応し，プログラムが複雑な場合にはＰ2＞Ｐ1であるから，Ｅ1，Ｅ2が同じならば，

タイプ３のｒ＞タイプ２のｒ

タイプ３のＦ＜タイプ２のＦ

となる．またＥ1が一定でＥ2が小さければｒは増加し，これはエラーの減りが大きいとエラー発見率が大きいとを示す．

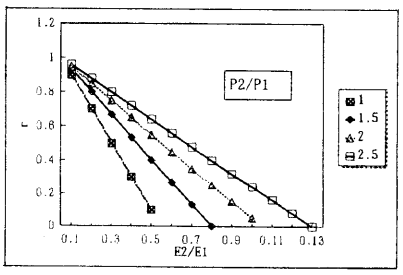


図５　ｒとＥ2／Ｅ1の関係

(17)式で，P2／P1＝1.0～2.5に対してE2／E1をいろいろ変えてｒを求め図５に示す．E2／E1が一定のときは，P2／P1が大きいほど，すなわちプログラムが複雑なほどｒが大きくなり，P2／P1＝１でフォールトタイプ２のｒと同じになる．

6　エラー総数

ｒが分かればフォールト総数Ｆは(15)式から容易に求められるが，エラー総数Ｇは(18)式でPLとｒに依存しあまり簡単ではない．PLは連結行列をＬ－１乗した行列の要素の値を合計すれば求められる．ここでフローグラフのうち簡単な条件のものについてその和を求めてみる．

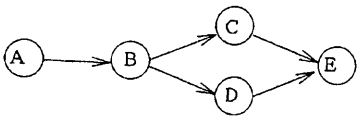
$$G = \Sigma EL = \Sigma F \times YL$$
$$= F \times r \times \Sigma PL \times (1-r)^{L-1} / P1 \quad (18)$$

6．1　ループを含まない場合

フローグラフにループがない場合には次のようなことがいえる．

・連結行列は，要素を適当に並べかえると下三角要素と対角要素は０とすることができる．もしどう並べ変えても行列の対角要素の下が０にならないとすると，戻るパスがあることになりループが生じ矛盾する．図６の例では，連結行列をべき乗すると３回目に０になる．これは長さ４以上のパスはないことを示す．｛　｝内は長さ順のパスの個数の列（パスベクトル）を表す．

・Ｌ＞４でPL＝０となるので容易にＧを求めることができる．ループがあっても有限回で打ち切ることができる場合には計算できる．



{PL} ＝ {5, 4, 2, 0}

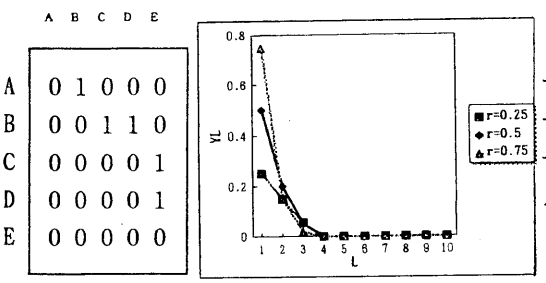


図６　ループのないフローグラフ

6．2　単純なループを直列に含む場合

・ループがＫ個直列に並ぶ場合には，長さＬのパスの個数はＫ個のループの中から順序を考慮せず重複を許しＬ個を選択する組合せの数となり，PLはＬのK-1次式になる[3]．図７の場合にはＬの１次式である．

・連結行列は，１が対角要素にループの数だけあり，下三角要素は０となる．

・PLはＬのK-1次式であるから，ＧはＣを適当な定数として

$$G = \sum_{L} C \times L^{K-1} \times (1-r)^{L-1}$$

という項目の和になるが，これは任意のＫについて収束する．証明は次のとおり．

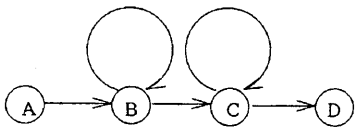
０＜ｒ＜１だから（１－ｒ）＝s^2とおけば，０＜ｓ＜１となる．したがって

$$G = \sum_{L} C \times L^{K-1} \times s^{2(L-1)}$$
$$= \sum_{L} (C \times L^{K-1} \times s^{L-1}) \times s^{L-1}$$

｜ｓ｜＜１ならば任意の正整数Ｋについて$L \to \infty$のときに，$L^K \times s^L \to 0$となるので[1]，上式の括弧内→０となる．したがってある定数Ｄによって括弧内の値＜Ｄとすることができ，よって上式は

$$G < \sum_{L} C \times D \times s^{L-1} = C \times D/(1-s)$$

となり収束が証明される．



{PL} ={5, 8, 12, 16, 20, 24, 28, 32・・}
PLの近似式＝4×L

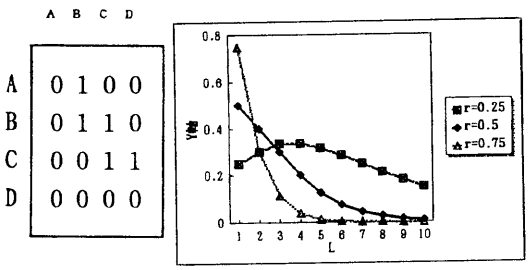


図7　直列のループを含むフローグラフ

図7の例では，r＝0.25のときはエラー数が一時増加してから減少している．ソフトウェアのデバッグのときに，発見バグ数の累積曲線を一時的に増加しまた減少するロジスティック曲線などで近似できること知られているが，それと関連すると考えられる．すなわち試験開始の初期では試験は短いパスで終ってしまうので，エラーの数は多くない．試験がより長いパスの分析に移ると多くのエラーが発見され，その数が増える．さらに進むと発見数が減少するというわけである．

6．3　ループの重合を含む場合

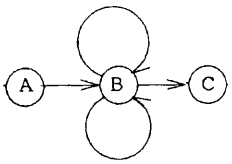
図8に示すような一つのノードに二つのループが付いているフローグラフを考える．これはループと分岐が一つのノードになった（重合）もので，プログラム文でいうとリピート文とＩＦ文が一緒になったものである．

・連結行列の対角要素に重合したループの個数（図8では2）が現れる．

・図8のノードBでのパスの個数PLは，2個のループの中からL個を順序も考えて重複を許して選択する組合せの数であり，2^Lとなる．ここでr≦0.5とすると，ノードBにおける見逃しエラー個数ZLは

$ZL = PL\times(1-r)^L / P1 = 2^L\times(1-r)^L \geqq 1$

となり，L→∞としても0にはならず，r＜0.5のときは逆にLの増加に伴いエラーは増加するという妙なことになる．状態爆発とはこういう現象のことと考えられる．したがって試験でエラーを0にするためにはr＞0.5でなければならない．そのときにGは求められる．



{PL} ={4, 9, 18, 36, 72, 144, 288・・}
PLの近似式＝2.25×2^L

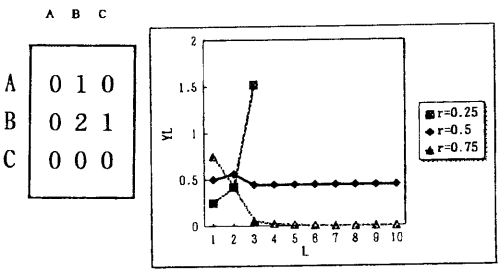


図8　ループの重合を含むフローグラフ

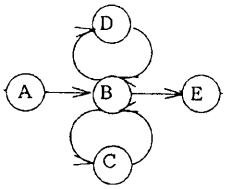
6．4　その他のフローグラフ

一般のフローグラフのエラー総数については，次のようなことがいえる．

（1）フローグラフを並列につないだ場合のPLは，各々のPLの和となる．

（2）ループにノードを追加することによってZLを改善できる．たとえば図8のフローグラフを図9のようにループにノードを挿入すると，Bからスタートして Bに到

るパスは長さＬが偶数のものだけとなり，ＰＬはサイクル２で振動する．リンクの間にノードを入れることは試験のチェックポイントを増やすのでＺＬの０への収束を促すことになる．



{ＰＬ} ＝{6, 11, 12, 22, 24, 44, 48, 88, ・・}

ＰＬの近似式＝Ｌ:奇数　$6\times2^{(L-1)/2}$

Ｌ:偶数　$5.5\times2^{L/2}$

	A	B	C	D	E
A	0	1	0	0	0
B	0	0	1	1	1
C	0	1	0	0	0
D	0	1	0	0	0
E	0	0	0	0	0

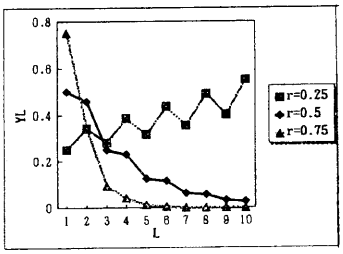


図９　図８の改良例

（３）フローグラフを合成した場合は最も複雑なフローグラフの影響にしたがう．ループの重合があればその影響が大きい．

（４）スパゲッティプログラムはＰＬを増大させエラーを増やす．したがって構造化プログラミングはＰＬの増加を抑えるのに有効である．ＧＯＴＯ文は連結行列の要素を増加させる．

（４）システム複雑度を表すサイクロマチック数は（リンク数－ノード数＋２）で表されるが，連結行列で考えると（要素数－マトリックス行数＋２）となり，これが少ない方がべき乗したときにＰＬの増加の可能性が少ないことがいえる．

７　まとめ

ソフトウェアの試験のフォールトモデルを考察し，パスの分析によって次のような問題について理論付けができたと考える．

- パス数ＰＬ，リンク当りエラー発見率ｒ，パスカバレジ，エラー数ＥＬなどの関係
- エラー数の推定とその収束条件
- エラー数がステップ数に比例しない理由，また発見エラー数の累積曲線がロジスティック曲線で表せたり，状態爆発が起きたりする理由
- パス数とＧＯＴＯレスプログラミングやサイクロマチック数との関係
- ビッグバンテストの非効率性

本論文の考えの裏付けデータとしてはトランスポート試験スイートの分析があるが[4]，現実の試験に当てはめ評価するのはこれからの課題である．

参考文献

[1] 若杉忠男：“状態遷移図の同定問題の一解法”，情報処理学会マルチメディア通信と分散処理研究会，(1996-10).

[2] 若杉忠男：“ＯＳＩ適合性試験スイートの評価法－マルチトランジション試験”，日本電子情報通信学会論文誌，VOL. J72-BI No.4 (1996-4)．

[3] 若杉忠男：“有限状態マシン(FSM)で表されるシステムの複雑度の評価について”，情報処理学会マルチメディア通信と分散処理研究会，(1995-9).

[4] 若杉忠男：“ISOで開発したトランスポートプロトコル適合性試験スイートの質の評価”，情報処理学会論文誌，Vol.37 No.3, (1996-3)

[5] Lipow, M:"Number of faults per line of code", IEEE Transactions on Software Engineering 8: 437-439, (1982).