ソフトウェア工学 80−15
（１９９１．７．１９）

# 埋込みモデルにおける エラー埋込みの自動化について

中村憲一郎　栗野俊一　深澤良彰　門倉敏夫
早稲田大学理工学部

ソフトウェアの信頼性を評価する基準の一つとして，プログラム中に含まれるエラーの数を用いる方法がある．このための手法の１つに，埋込みモデルがある．この埋込みモデルは，プログラムに故意にエラーを埋め込んでテストを行い，その結果からエラー数を推定する方法である．この時，埋め込むエラーは，真に含まれるエラーと，その分布，性質等が類似していることが必要である．エラーの生成および埋込みの過程を手作業に頼る場合，上記の条件を満たすことは困難である．本研究では，この過程を自動化し，手作業による問題点の解決をはかる．また，同モデルにいくつかの拡張を行い，これを用いたソフトウェア信頼度推定システムを提案する．

# Automatic Generation and Insertion of Errors in a Seeding Model

Ken-ichiro Nakamura, Shun-ichi Kurino, Yoshiaki Fukazawa, Toshio Kadokura
School of Science and Engineering, Waseda University
Okubo 3-4-1, Shinjuku-ku, Tokyo 169, Japan

In order to estimate the software reliability, the number of remained errors seems to be a mile stone. By utilizing this property, some error seeding models have been proposed. In these methods, some errors are intentionally seeded in a given program, the program is tested, and the number of error is estimated based on the testing results. At this time, it is necessary that the distribution, probabilities and properties of the generated errors are similar to those of the originally contained errors. In case of the hand generation and insertion, it is difficult to satisfy these conditions. In this paper, we propose a software reliability estimation system which automates above process and an extention of the seeding model.

## 1 はじめに

システム開発における重要な目標の1つに，ソフトウェアの信頼性向上がある．信頼性の高いソフトウェアを開発するための手法として，いくつかのソフトウェア・テスト手法によって，エラーの検出を行うことがある．このとき，ある特定のエラーに対するテストの有効性が不明であったり，テストを終了する基準が明確でないといった問題が生ずる[1]．これらの問題を解決するためには，ソフトウェアの信頼性を数量的に評価する方法を確立しなければならない．

プログラム中に残存するエラーの数を推定することができれば，それが一種の信頼性評価になると考えられる．プログラム中のエラー数を推定するための信頼性モデルとして，現在提案されているものには以下のようなものがある[2]．

- 成長曲線モデルによる推定
- 構造パラメータによる推定
- 確率モデルによる推定

成長曲線モデルによる推定では，テストに要した時間と検出されたエラーの累積数による回帰分析を行い，その結果から残存エラー数を推定する．このとき用いる曲線には，ロジスティック曲線やゴンペルツ曲線などがあり，他にもさまざまなモデルが提案されている[3]．

また，構造パラメータによる推定は，プログラム中の種類の異なるオペレータの数やオペランドの数などからプログラムの抽象度を求め，これを用いてエラー数を推定するものである．

確率モデルによるエラー数の推定方法は，対象プログラムから2つ以上のエラーの集合を独立に収集し，それぞれの集合で共通なものとそうでないものの比から全体のエラー数を推定するというものである．このモデルでは比較的前提条件が少ないため，良い結果が期待できる．

1つのプログラムから複数のエラーの集合を得るための方法として，同時に独立なテストを行う方法や，最初に既知のエラーを埋め込んでからテストを行う方法などが提案されている．

本研究では，エラーを埋め込む方法を扱う．これは，「エラー埋込みモデル」と呼ばれる．

### 1.1 エラー埋込みモデル

エラー埋込みモデル[4][5]は，統計学における確率論に基づいた信頼性モデルの一つである．その概要を以下に示す．

まず，元のプログラム中に存在する真のエラーの総数を $M_i$ とする．ここに人為的に $M_c$ 個のエラーを埋め込む．このプログラムをテストすると，真のエラーと埋め込んだエラーの両方が検出される．

検出されたエラーのうち，真のエラーの数を $N_i$，埋め込んだエラーの数を $N_c$ とする．

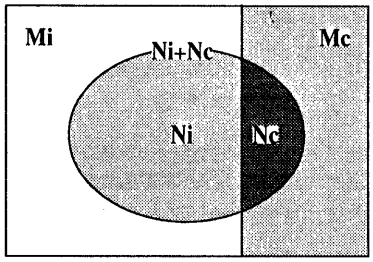
真のエラーと埋め込んだエラーは同様に検出されると仮定すると次式が成立する．

$$N_i : N_c = M_i : M_c \tag{1}$$

$M_i$ について解けば

$$M_i = N_i \cdot M_c / N_c \tag{2}$$

となる（図1）．したがって，テスト終了後にプログラム中に残存している真のエラーの総数の推定値は，$M_i - N_i$ で与えられる．




**Mi:** 真のエラーの総数
**Mc:** 埋め込んだエラー数
**Ni+Nc:** 検出されたエラー数
（ **Ni:** 真のエラー，**Nc:** 埋め込んだエラー）

図 1: エラー埋込みモデル

エラー埋込みモデルは，「人為的に埋め込むエラーと元から存在するエラーの検出率が等しい」という仮定に基づいている．そこで，エラー埋込みモデルを使用する際には，埋め込むためのエラーをどのようにして得るかが問題となる．

従来の方法では，これは手作業で行う，1度検出されたエラーの埋戻しを行うなどで対処していた．

埋め込むエラーを手作業で生成する場合，真のエラーに近い性質を持つエラーを作るのは難しく，また埋め込む位置や数などを決定する手間を考慮すると，この方法は現実的とは言い難い．

1度検出したエラーを埋め戻す方法は，テストによってエラーがランダムに検出されることを前提としている．しかし現実のエラーは，検出が容易なものから非常に困難なものまで広い範囲にわたり，検出しやすいものは誰にでも容易に検出されるという傾向がある．このため，実際には見つかりにくいエラーを扱うことが困難になる．

そこで，エラー埋込み作業を自動化し，埋め込むエラーの性質や分布を一定の基準に基づいて決定することで，前述の問題点の解決をはかることが考えられる．

## 2 本研究の特徴

本研究の目的は，埋込みモデルにおけるエラー埋込み作業の自動化を行い，その有効性を調べることである．

この評価を行うため，ソフトウェア信頼度の推定システムを構築した．

前述のように，従来埋込みモデルを用いる際には，検出されるエラーの種類や分布に偏りが生じ，このために，真のエラーに近い性質のエラーを生成することが困難であった．本システムでは，以下に述べるいくつかの工夫によって，これらの問題点の解決をはかっている．

**エラーの分類**　従来の埋込みモデルは，対象プログラムに含まれるエラー全体の数を推定するものであった．しかし，同モデルを適用する場合，実際には何種類ものエラーが存在し，各々が異なる性質をもつという点を考慮しなければならない．このためにはエラーの分類を行い，各々の分類について別々に推定を行う必要がある．ただし，これによって統計処理の際の母集合が小さくなり，統計的な性質を失わないように注意する．

また，埋め込むエラーの種類や分布は，対象プログラムに含まれる真のエラーに近いものでなければならない．そこで本システムでは，統計および過去の履歴によるエラー発生分布表を参照し，各分類について埋め込むエラーの分布をより現実的なものとしている．

**エラーの重要度の考慮**　ソフトウェア信頼性の評価を行う場合，単にエラーの個数を示すだけでなく，それぞれのエラーの重要度を示すことができると，より実用的である．そこで，本システムでは推定対象をエラー個数からエラー重要度へ拡張する．

エラーの重要度としては，エラーの種類による重要度と，エラーの位置による重要度の2種類を考慮する．例えば，前者には「条件判定部のエラーはメッセージのつづりの誤りよりも重要である」というものがあり，後者には「同じ条件判断部のエラーであればより大きなブロックの制御を行うものの方が重要である」といったものがある．

**エラー履歴の管理**　実際のテストおよびデバッグ作業を分析してみると，プログラマによって引き起こすエラーの傾向が異なる．また，エラーの種類やその数は，複雑なアルゴリズムを要求するかどうかといったアプリケーションの性質にも大きく依存する．

本システムでは，テストによって検出されたエラーの履歴を次回のエラー埋込みの際に参照することにより，以下の2点を埋め込むエラーの分布に反映させる．

- プログラマの癖などによるエラーの傾向
- アプリケーションの種類によるエラーの傾向

以上の機構により，本研究では，プログラムのより実用的な信頼度を推定することを目標とする．

## 3　本システムの概要

本システムは，埋込みモデルを用いたソフトウェア信頼度推定システムである．全体の構成を図2に示す．

本システムによるプログラムの高信頼化手順は，以下の段階に分けられる．

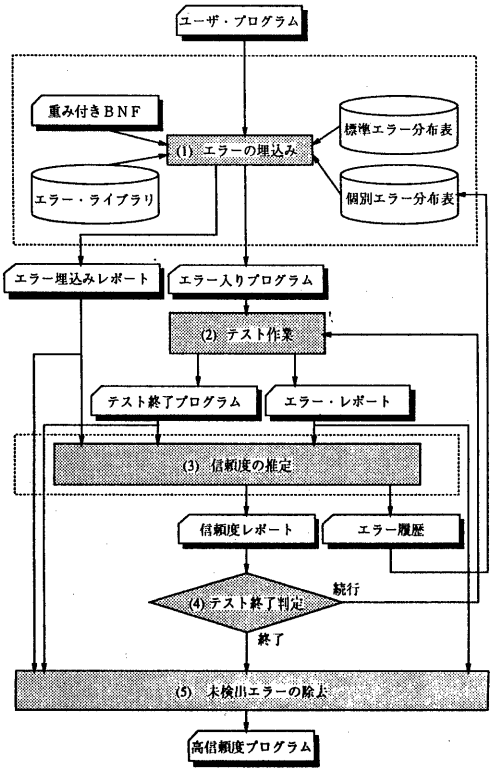



図 2: 本システムによるプログラムの高信頼化手順

(1) 対象プログラムにエラーを埋め込む
(2) エラーの埋め込まれたプログラムをテストし，エラーを検出する
(3) テストの結果を用いて信頼度推定を行う
(4) 推定された信頼度を評価し，テスト終了の判断を行う
(5) 埋め込んだエラーで未検出のものを除去する

本システムでは，上記において (1) の対象プログラムにエラーを埋め込む過程および (3) の信頼度推定の過程を自動化する．図2では点線で囲まれている部分がそれにあたる．

本システムでは，与えられたプログラムに対し，(1) のエラーの埋込みを行うためのソフトウェアを，Software Bebugger と呼ぶ．Software Bebugger の動作は，エラーの分布表と対象プログラムの行数から，埋め込むエラーの種類や数を決定し，実際にエラーを埋め込むことである．

Software Bebugger は，エラーを埋め込んだプログラムと，埋込みの内容に関するレポートを出力する．このエラー埋込みレポートを，信頼度の推定時に利用する．また，検出されたエラーは履歴として個別のエラー分布表

に記録し，次回の信頼度推定時に埋め込むエラーの分布に反映させる．

本システムの対象言語として，試験的に“Pascal-F”の言語仕様を定めた．“Pascal-F”の言語仕様は，本システムの実現を容易にするために，Pascal の言語仕様[6]からレコード型やポインタ型，case 文や repeat 文などを取り除いたサブセットである．

## 4 実現方法

### 4.1 埋込みモデルの拡張

前述の埋込みモデルの拡張を，本研究では以下のように実現する．

#### 4.1.1 エラーの分類

本システムでは，埋め込むエラーの分布を決定するために，対象プログラムの種類によっていくつかの標準エラー分布表を持つ．この分布表と，後に述べるエラー履歴表に基づき，埋め込むエラーの種類や数を決定する．

ここで，全てのエラーを完全に重複なく分類することは困難であるが，なるべくあいまいさを避け，互いに重複しないように工夫している．

表1に，本システムで用いるエラーの分布表を示す．表の数字はエラーの分布を表している． 分類は文献[3]に基づいて行い，“Pascal-F”の言語仕様に合わせて修正および簡略化がなされている．

表 1: 標準エラー分布表（抜粋）

| 詳細エラー分類 | 大分類中での相対発生頻度 | | | |
|---|---|---|---|---|
| | 応用ソフトウェア | シミュレータソフトウェア | オペレーティングシステム | PAツール |
| 演算エラー | 18.9 | 24.9 | 3.2 | 0 |
| 式のオペランドが不正 | 43.7 | 40.0 | 100.0 | 0 |
| 符号のエラー | 3.1 | 13.3 | 0 | 0 |
| 不正な式／精度が低い式 | 40.7 | 20.0 | 0 | 0 |
| 演算が抜けている | 12.5 | 26.7 | 0 | 0 |
| 論理エラー | 23.9 | 26.6 | 44.5 | 47.4 |
| 論理式のオペランドが不正 | 23.8 | 7.4 | 7.7 | 4.6 |
| 論理のシーケンスが不正 | 19.0 | 35.1 | 11.5 | 11.3 |
| 不正な変数をチェックしている | 4.8 | 10.0 | 15.4 | 2.3 |
| 論理／条件判定が抜けている | 52.4 | 47.5 | 65.4 | 81.8 |
| データ出力エラー | 4.9 | 5.6 | 6.3 | 0 |
| 出力が不完全で抜けがある | 100.0 | 25.0 | 100.0 | 0 |
| 出力フィールドのサイズが小さい | 0 | 75.0 | 0 | 0 |
| データ取り扱いエラー | 15.3 | 10.7 | 27.0 | 10.2 |
| データの初期化がされない | 15.5 | 22.3 | 27.2 | 0 |
| データの初期化が不正 | 46.2 | 22.3 | 63.7 | 100.0 |
| 変数型が不正 | 7.6 | 22.3 | 0 | 0 |
| 添字付けのエラー | 30.7 | 33.1 | 9.1 | 0 |
| インターフェイス・エラー | 13.7 | 8.5 | 9.5 | 0 |
| サブルーチン引数が矛盾 | 100.0 | 100.0 | 100.0 | 0 |
| データ定義エラー | 10.2 | 17.5 | 9.5 | 4.0 |
| データ／次元が不正定義 | 82.0 | 92.9 | 100.0 | 100.0 |
| データの参照場所が不正である | 18.0 | 7.1 | 0 | 0 |
| その他 | 13.1 | 6.2 | 0 | 38.4 |
| 不要なコード／効率の悪いコード | 100.0 | 100.0 | 0 | 100.0 |

#### 4.1.2 エラーの重要度

埋込みモデルはプログラム中のエラー数を推定するものであるが，本システムでは推定の対象をエラーの個数からエラーの重要度に拡張している．この結果，システムの出力は「エラーが何個」という表現から「エラーの重要度がいくつ」という表現に変わる．

エラーの重要度は，プログラム中でのエラーの位置および構文規則により決定する．このためには，動的に決定する方法と静的に決定する方法が考えられる．前者はプログラムを実際に実行させた場合の，個々の部分の実行頻度に依存し，後者はプログラムの構文木上における位置に依存すると考える．

本システムでは重要度を静的に決定する．すなわち，あるプログラムが与えられた時，その中で構文的に重要な部分とそうでない部分を仮定し，その部分に含まれるエラーの個数を用いて重要度を決定する．これは，プログラムの構文木に重みを付けることによって実現できる．このとき，重みは構文木の形と構文規則に基づいて決定する．

**構文木の形に依存する重み付け** プログラムの構文木に対して，一定の規則で重みを付けることを考える．このとき，重みを付けた構文木は，次の条件を満たすものとする．

- 各節点の重みは子の節点の重みの和となっていること
- 節点の数が変化したときの各節点の重みの変化が現実的であること

木構造に対する重み付けの方法としては，トップダウン法とボトムアップ法がよく知られている．トップダウン法は，根にある重みを与え，子の重みを親の重みの等分とすることで決定する方法である．また，ボトムアップ法は，すべての葉にある一定の重みを与え，親の重みを子の重みの総和とすることで決定する方法である．ただし，これらの手法を用いてプログラムの構文木に重みを付けた結果は，本研究の目的に適さない部分がある．

トップダウン法による重み付けでは，ある節点の子の節点がいくら増えても，元の節点の重みは変化しない．これを構文木に適用すると，例えば大小２つの手続きが並んでいた場合，両者の重みはその大きさに関係なく等しくなってしまう．一方，ボトムアップ法による重み付けでは，ある節点の重みは子供の節点の数に直接比例して大きくなっていく．これを構文木に適用すると，例えば10個の変数からなる変数宣言部は，１個の変数からなる変数宣言部の10倍の重みを持つことになってしまう．

全体のバランスを考えた場合，これらの重みをエラーの重要度として扱うには不適当である．そこで本システムでは，筆者らが考案した「比付きの重み付け法」[7]を

採用する．比付きの重み付け法は，トップダウン法とボトムアップ法を一般化したものであり，両者の中間的な性質を持つ重み付けが可能である．

具体的な重みの計算方法を表2に示す．

表 2: 重みの計算方法

| 重み付けの種類 | 新たな節点 (=g') | 兄弟 | 先祖 | 兄弟の子孫 |
|---|---|---|---|---|
| トップダウン | $\frac{g_1}{i}$ | $-\frac{g'}{i-1}$ | -- | $*\frac{i-1}{i}$ |
| ボトムアップ | $\frac{g_1}{i-1}$ | -- | $+g'$ | -- |
| 比付き | $\frac{g_1}{i-1+r}$ | $-\frac{g'*r}{i-1}$ | $+g'*(1-r)$ | $*\frac{i-1}{i-1+r}$ |

$g_1$ ... 新たな節点を追加する直前の親の重み
$i$ ... 新たな節点の番号（何番目の子供か）
$g'$ ... 新たな節点に与える重み
r ... 分配比 $(0 \leqq r \leqq 1)$

表2は，既に重み付けが完了している木に，新しく1つ節点を追加した時の，それぞれの重み付け法の実際の計算内容を示している．例えばボトムアップ法では，$g_1$ の重みを持つ節点に $i$ 番目の子を追加するとき，子の節点には $g' = g_1/(i-1)$ の重みを与え，その先祖の節点の重みはすべて $g'$ だけ増加することを表している．

表中の変数 r は分配比と呼ばれるパラメータで，r = 0 の時ボトムアップに，r = 1 の時トップダウンにそれぞれ等しくなる．

このように，比付きの重み付け法の特徴は，ある節点の重みの，親や兄弟への影響の比率を，分配比を変化させることによって調節することができる点である．

図3に示す非常に単純なサンプルプログラムに，トップダウン法，ボトムアップ法，比付きの重み付け法で重みを付けた結果を，それぞれ図4に示す．比付きの重み付け法による結果は，他の2つの手法による結果と比較して，より柔軟な重み付けができることがわかる．

```
program sample1;
  var a,b : integer;
  begin
    a := 1;
    b := 2;
    write(a,b)
  end.
```

図 3: サンプルプログラム

**構文規則に依存する重み付け**　比付きの重み付けの手法は，一般的な木構造に対して定義されており，重み付けの結果は木の形のみに依存する．したがって，木の中で対称な位置にある節点は必ず同じ重みになり，ある意味で公平な重み付けが行われる．

エラーの種類によって重みを変化させるためには，構文木の中では対称な位置の節点同士であっても，それらに異なる重みを与える必要がでてくる．例えば，代入文と出力文の間や，if 文における条件部と実行部などの間で重みに差をつけることなどがこれにあたる．

そこで本システムでは，プログラム構文木のある場所の重みを，その前後との構文的な関係により相対的に変化させる．これらの情報は対象言語である “Pascal-F” の BNF 中に記述し，対象プログラムの構文木を生成する際に参照する．

例えば，while 文の条件部と実行部の重みの割合を 2:3 にしたい場合は，BNF の while 文の部分に次のように記述する．

```
                                    2              3
<while_st> ::= "while" <cond_exp> "do" <stmt>.
```

これは，図5(a) に示すような重み付けを行うことを意味するが，ここで一時的に図5(b) に示すような状態を考え，これに重み付けを行った後に元の状態に戻す．

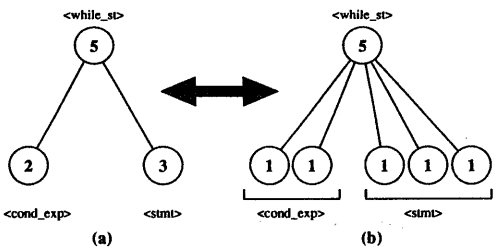


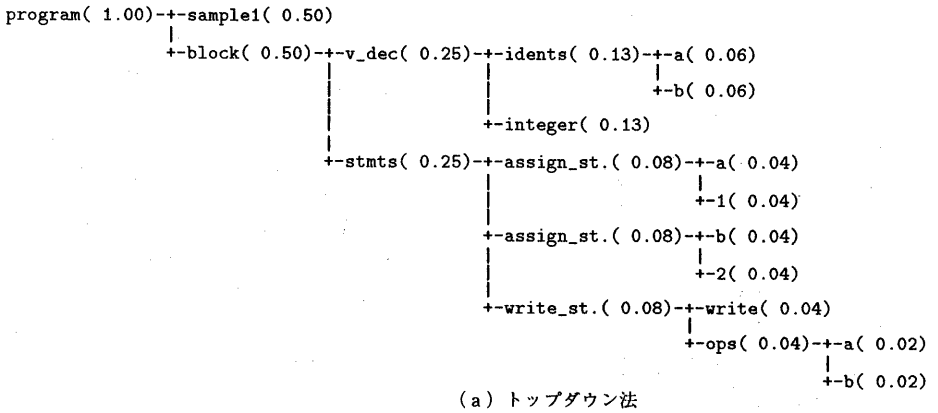

図 5: 不公平な重み付けの実現

#### 4.1.3 エラー履歴の管理

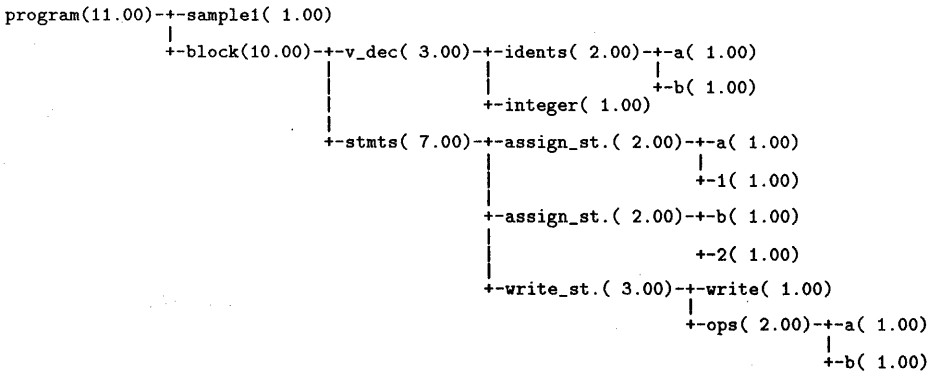
プログラムに含まれる真のエラーは，プログラマやアプリケーションの種類によって異なる分布となる．本システムでは，埋め込むエラーを真のエラーの分布に近付けるために，過去のエラーの履歴表を参照する．

エラー履歴表は，過去に検出されたエラーの重要度を，プログラマおよびアプリケーションの種類ごとに分類して，記録しておくものである．ただし，本システムでは，アプリケーションの種類として，ＯＳ，コンパイラ，事務計算システム，数値計算システム程度の分類を考えている．

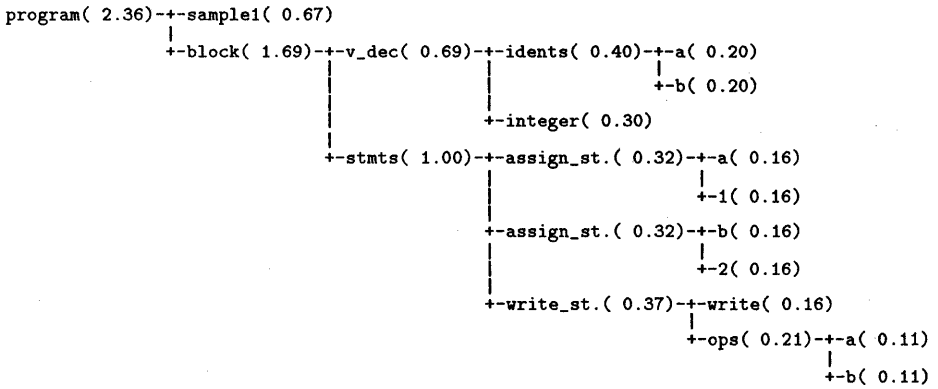
この履歴表から，プログラマおよびアプリケーションの種類によるエラーの傾向を抽出し，標準のエラー分布表と組み合せることによって，ある程度個性が反映された分布表が作成できる．また，全体でどれぐらいの重要




（a）トップダウン法




（b）ボトムアップ法




（c）比付きの重み付け法（分配比 r = 0．5）

図 4: 各手法による重み付けの結果

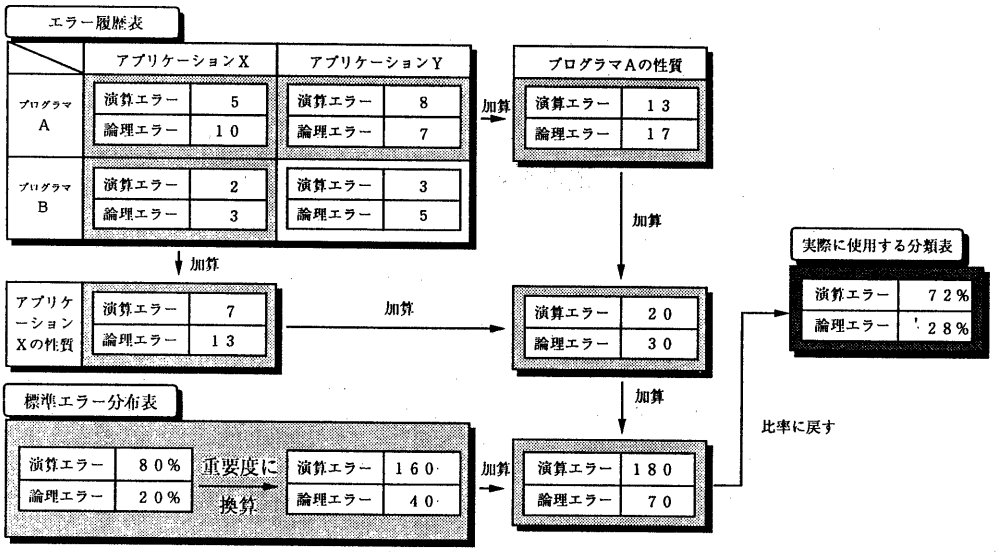



図 6: エラー履歴の管理

度のエラーを埋め込むかは，対象プログラムのサイズにより決定する．

例として，プログラマAが書いたアプリケーションXに本システムを適用する場合について，埋込みに使用するエラー分布を求める手順を図6に示す．

エラーの埋込みは，こうして得られた分布表を参照しながら行う．また，この対象プログラムにテストを実施して検出されたエラーのうち，真のエラーを調べて，エラー履歴表の更新を行う．

### 4.2 エラーの埋込み

埋め込むエラーの種類およびその重要度が決定されると，対応するエラーのパターンがライブラリより呼び出される．エラーのパターンは表1に基づいて分類されており，部分木の形で登録されている．

例として，図3のプログラムにエラーを埋め込む過程を図7に示す．

このようにして，所定の重要度に達するまで，構文木上でエラーの埋込みを行い，エラーの埋め込まれた対象プログラムが得られる．また，同時に埋め込んだエラーの種類や位置が，その重要度と共にエラー埋込みレポートとして出力される．

### 4.3 信頼度の推定

プログラム中に残存するエラー重要度の推定は，式(2)に準ずる．

まず，テストを実施した結果検出されたエラーを，埋め込んだエラーと真のエラーに分離する．次にエラーの分類を行い，各々の分類別に式(2)を適用する．この結果，プログラム中に元から存在しているエラーの分類別の重要度が求められる．ここから，既に検出されたエラーの重要度を引くことにより，プログラム中に残存するエラー重要度が推定できる．

残存エラー重要度は，残存エラーの個々の重要度の総和を表す．したがって，例えば「残存エラー重要度10」は，ただ1つの重要度10のエラーであること，重要度7のエラーと重要度3のエラーの組み合わせであること，重要度1のエラーが10個であることなどが考えられ，その評価はテストの担当者に委ねられる．

## 5 おわりに

本稿では，埋込みモデルを用いてエラー数を推定する際の問題点を示し，その解決法としてエラー埋込みの自動化について述べた．さらに，従来の埋込みモデルに対しいくつかの拡張を行い，その実現方法について述べた．また，これを用いてソフトウェアの信頼度を推定するシステムを提案した．

現在，本システムの中心となる Software Bebugger のインプリメント作業を，yacc を用いて行っている．現時点で，以下の機能が実現されている．

- 対象プログラムの構文チェック
- 対象プログラムの構文木の生成
- 構文木上での重要度の分布の生成
- 数種類のエラーの位置の決定および埋込みの実行

今後の課題として，以下のような項目を検討中である．

**対象言語およびエラー分類の妥当性の検討** 一般に，エラーの種類やその数は使用するプログラム言語に依存す

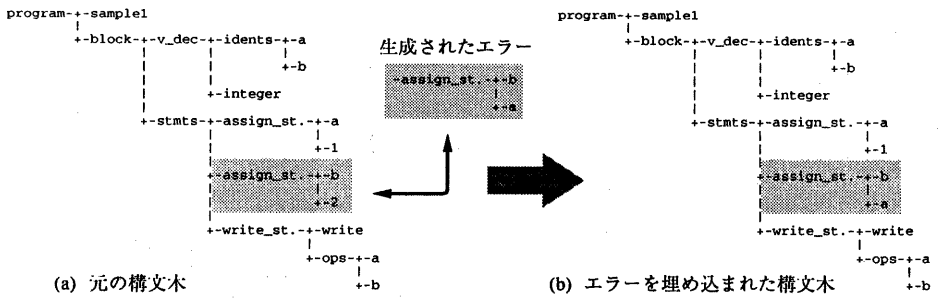



図 7: エラーの埋込み

る．現時点では標準エラー分布として暫定的に表1のデータを用いているが，このデータはPascal-Fとは別の言語によるものであり，本システムに最適なものではない可能性がある．また，Pascal-Fの言語仕様は本システムの実現性を調べるために定めたもので，実用のプログラムの記述に不向きな面がある．

そこで，本研究の有効性を確認次第，対象言語をC言語など実際のアプリケーションの記述に使用されている言語に変更し，本システムをより実用的なものに改良する．また，そこで選択した対象言語におけるエラーの例をなるべく多く調べることによって，表1の分類の利点および欠点を明らかにし，その妥当性を検討する．そして，必要に応じて分類基準の修正・改良を加えていく方針である．

**動的なエラー重要度の考慮**　本システムではエラーの重要度を静的に決定するが，プロファイラ等を利用することにより，実行頻度に基づく動的な重要度の導入が可能になると考えられる．ただし，多数のエラーを含むプログラムのプロファイラ出力をどれだけ考慮するべきかについては検討を要する．

**エラー検出数の変化率の考慮**　現段階では，テストの実施回数によるエラー検出数の変化率を考慮していない．実際には，テスト期間中に検出されるエラー累積数と時間に関係が存在することが経験的に確かめられており，さまざまな信頼度成長曲線のモデルが提案されている[8]．埋込みモデルとこれらの成長曲線モデルを組み合わせることによって，さらに正確な推定ができると思われる．

**エラー重要度の妥当性の検討**　静的なエラーの重要度について，本システムでは新しい重み付けの手法を導入した．この結果，かなり柔軟に重みの分布を変化させられることが確かめられた．今後，さらに多くの例について重みの分布を調べ，その妥当性を検討することで，より自然な結果を与えるようなパラメータの収集を行う予定である．

**エラーライブラリの構造の改善**　現時点で，エラーの埋込みを行う部分は，エラーの種類ごとに直接C言語のプログラムとして実現されており，拡張性に問題がある．この部分をSoftware Bebugger本体と切り離し，本システムで扱うエラーの種類の増大に対応できるようにする必要がある．また，将来的には仕様記述のような形でエラーを記述し，ライブラリ化できるようなシステムへ拡張をしたいと考えている．

## 参考文献

[1] 玉井哲雄他著, ソフトウェアのテスト技法. 共立出版,1987.

[2] 榎本肇著, ソフトウェア工学ハンドブック. オーム社,1986.

[3] 宮本勲著, ソフトウェア品質の現状と改善策.TBS出版会,1982.

[4] G.J. マイヤーズ著, 有澤誠訳, ソフトウェアの信頼性. 近代科学社,1977.

[5] シューマン著, 寺本他訳, ソフトウェアの信頼性. マグロウヒル社,1989.

[6] A.V. エイホ他著, 大野義夫訳, データ構造とアルゴリズム. 培風館 ,1987.

[7] 栗野俊一, 中村憲一郎他: 木の公平な重み付けについて. 情報処理学会アルゴリズム研究会研究報告,90-AL-16-8,1990.

[8] 大場充著, ソフトウェアの開発技術. オーム社,1988.