「バグ捕獲・再捕獲法」を用いたソフトウェア信頼性保証について

伊土誠一†

ソフトウェアの開発において，潜在バグ数を精度よく推定できれば，効率のよい信頼性保証活動が可能となり，適切なソフトウェア開放時期が決定できる．従来は，開発途中までのバグの発生数の経時変化を把握し，それをソフトウェア信頼度成長モデルにあてはめることにより，潜在バグ数を推定する方法が主に開発されてきた．この手法はデバグや試験のプロセス，横軸として何を採用するかが推定精度に大きく影響する．本論文では，バグ摘出工程の途中のプロセスが潜在バグ数推定精度に影響を与えない特徴をもち，野生動物の頭数などを推定する手段としてよく知られている「捕獲・再捕獲法」をソフトウェアのバグ数推定に適用することを考える．それには，2つの問題を解決する必要がある．1つは，対象プログラムに埋め込んだバグをどのように選定するかである．これはバグ数の推定精度に大きく影響する．2番目は，バグ数推定のために埋め込むバグが発生することによるトラブルである．本来の品質保証作業の進捗に影響を与えないような工夫が必須である．本稿では，これらの課題を解決する「バグ捕獲・再捕獲法」を提案する．さらに，ここで提案した方法論とバグ捕獲・再捕獲法のために開発したツールを，実際に商用に供する大規模ソフトウェアに適用した事例を紹介する．最後に，本方式とソフトウェア信頼度成長モデルとの推定精度の比較等の考察により，バグ捕獲・再捕獲法の有効性を示す．

Assuring High Reliability Software Based on the Bug Capture and Recapture Sampling Method

SEIICHI IDO†

It is possible to assure the software system quality and to release the software system in a timely manner, if we can accurately estimate the number of remaining errors during the software system development process. This paper proposes a concrete method and a tool for estimating the number of remaining errors, based on capture and recapture sampling. To illustrate this, two problems are discussed; how to choose seeded errors to ensure the accuracy of the estimation through capture and recapture sampling, and how to avoid the troubles occurring when seeded errors are made to appear during the activity of recapture sampling. The usefulness of a proposed method is verified by showing that the estimated data determined by this method actually coincide with the real data in large-scale software development.

1. はじめに

開発中のソフトウェアの潜在バグ数を精度よく推定できれば，効率のよい信頼性保証活動が可能となる．また，ソフトウェア出荷時の信頼性を保証でき，適切なソフトウェア開放時期を決定できる．

従来，潜在バグ数の推定は，開発途中までのバグの発生数の経時変化を把握し，それに成長曲線モデルをあてはめることにより，その後のバグの発生傾向を予測する方法をとってきた．その一つの流れは，ロジスティック曲線やゴンペルツ曲線というような決定論的モデルを用いる方法である．これは古くから動物の繁殖数予測，電話の需要予測等に用いられている方法であり，1970年代から日本でソフトウェアの残存バグ予測に用いられるようになった．もう一つの流れは，Jelinski と Moranda[1]や Shooman によって提案された方法であり，確率・統計論を用いたモデルをベースにしている．ソフトウェアの信頼度成長は，ソフトウェアに潜在するバグ総数とテスト稼働との関係から説明できるというものである．この方法はその後，Musa[2]，Goel & Okumoto[3]，Littlewood[4]，大場[5]，山田ら[6]など多くの人によって改善されてきた．また，最近ゴンペルツ曲線に確率論を導入するなど，双方の特徴を生かしつつ手法の改善を図ろうという動きもある[7]．

これらの成長曲線あてはめ法は，バグ摘出工程の

† NTT 情報通信網研究所
NTT Network Information Laboratories

途中のプロセスが潜在バグ数推定精度に大きく影響する．また，推定がマクロ的であり，デバグ工程におけるバグ摘出の目標設定や目標達成のおおよそのチェックには有効な方法である．しかし，最終的なソフトウェアの出荷時期を決定するときのように精度を要求される判断—潜在バグ数の予測，サービス後のトラブル予測，それに基づくソフトウェアの開放可否の判断—に用いるには有効な手法とはいいがたい．また，最近発見・除去されたバグ数の増加を表すときの横軸として時間ではなく，テストインスタンス（テスト作業のあるまとまった単位）を用いる手法[9]も提案されている．これも一種の成長曲線あてはめ法といえる．

本論文では，ソフトウェア開発の最終工程においてソフトウェアの高信頼性を保証し，ソフトウェアの出荷の最終判断をするための手段として，野生動物の頭数推定に用いられている「捕獲・再捕獲法(Capture & Recapture Sampling Method)」をソフトウェアの潜在バグ推定向きに改善した方法（バグ捕獲・再捕獲法と呼ぶ）を提案する．この方法は，潜在バグ数を推定するためにバグの発見の履歴をプロットする必要がない点で，成長曲線あてはめ法と異なる．

「捕獲・再捕獲法」をソフトウェアの品質把握に適用するアイディアは，1972年に IBM の社内誌に Mills が提案している[8]．著者らは文献 10) で捕獲・再捕獲法をソフトウェアの潜在バグ数推定に適用する場合の具体的課題を指摘し，その解決の指針を示した．また，捕獲・再捕獲法に関しては，国内外でいろいろな角度から議論がなされている[11)~15)]が，大規模なソフトウェア開発に具体的に適用して，一貫した方法論として確立されたものはない．

本論文では，2章で捕獲・再捕獲法の基本原理，3章では2章で述べた原理をベースにこれをソフトウェアの信頼度推定用に改善したバグ捕獲・再捕獲法を提案するとともに，これを実現する上での課題について述べる．4章ではバグ捕獲・再捕獲法において最も潜在バグ数の推定精度に影響を与える対象プログラムへ埋め込むバグの選定法について，5章では捕獲・再捕獲法を適用するための支援システムについて，6章では捕獲・再捕獲法の適用例，および，その他の手法との推定精度に関する比較について論ずる．

2. 捕獲・再捕獲法の基本原理

捕獲・再捕獲法は，記号放逐法（Mark & Release Method）とも呼ばれている．最初に捕まえた動物に記号（標識）をつけてもとの場所に放し，他の動物と十分に混ざり合った状態になった時点で，再度捕獲作業を行う．捕らえた動物のうち，再捕獲した個体の再捕獲率によって式(1)により，総個体数（N）を推定する方法である[16)~18)]．

$$N=Mn/m \qquad (1)$$

ここで，

M：記号をつけた個体数

n：再捕獲時の捕獲個体数

m：再捕獲時に捕獲した個体のうち記号のついた個体数

この式が成立する条件は，標識付き個体と標識なし個体の捕獲率に差がないことである．

3. ソフトウェア潜在バグ数推定への適用

3.1 ソフトウェア開発工程との関係

ソフトウェアは通常，**図1**の工程に沿って開発される．品質管理はソフトウェアの計画段階から運用まで一貫して実施されなければならない．ソフトウェア・ライフサイクルにおける製造，デバグ，試験の段階においては，いかに効率よく，かつ，網羅的に信頼性を上げるかの観点から構造テスト，機能テスト，安定化試験などを実施する．また，品質管理者は，これらの品質向上活動を通してバグ発生の累積曲線，バグの内容，試験での MTBF などによってソフトウェアが安定してきたかどうかを判断する．しかし，ソフトウェアを出荷してよいか否かの判断には，どの程度のバグがまだ潜在しているかを精度よく推定する必要がある．

3.2 バグ捕獲・再捕獲法

図2は，2章で述べた捕獲・再捕獲法の原理をベースに，ソフトウェアの潜在バグ数推定用に改善した方式の概要である．以後，この方法は一般的な手法としての捕獲・再捕獲法と区別するために，バグ捕獲・再捕獲法と呼ぶこととする．また，2つの方法の比較を**図3**に示す．バク捕獲・再捕獲法において，式(1)は次のように解釈すればよい．

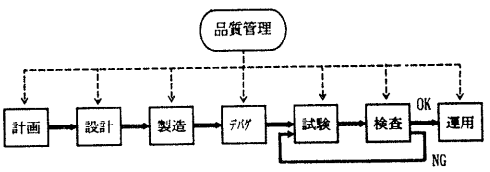


図1 ソフトウェア・ライフサイクルと品質管理
Fig. 1 Software lifecycle and quality control.

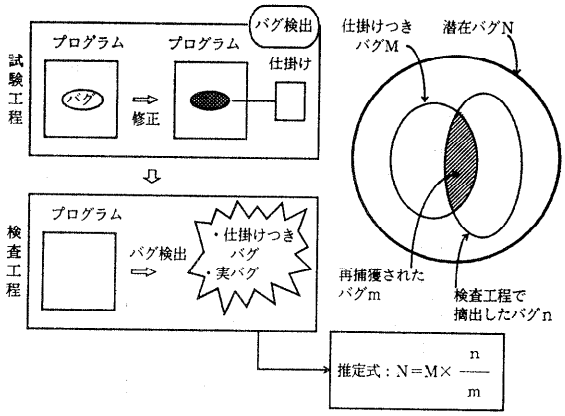


図 2 バグ捕獲再捕獲方式
Fig. 2 Bug capture and recapture sampling method.

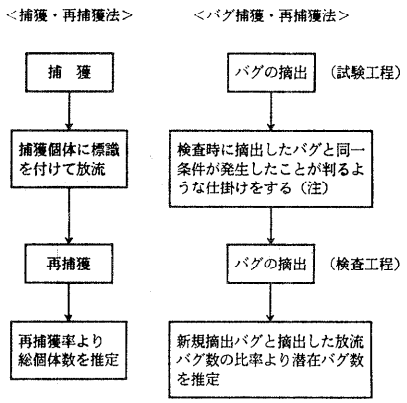


図 3 捕獲・再捕獲法とバグ捕獲・再捕獲法
Fig. 3 Capture & recapture sampling method and bug capture & recapture sampling method.

①Mは対象ソフトウェアに意図的に埋め込んだバグ（以後，埋込みバグ（seeded bug）と呼ぶ）数である．

②nは再捕獲時に摘出した総バグ数である．

③mは再捕獲時に摘出した埋込みバグ数である．

その結果，捕獲・再捕獲法実施完了時の潜在バグ数(B)は，

$$B=N-M-(n-m) \qquad (2)$$

となる．

また，推定結果の信頼度 $100\times(1-\alpha)\%$ の信頼区間は次の式で求めることができる[19]．

$$\frac{m}{n}-k_\alpha\times\sqrt{\frac{N-n}{N-1}\times\frac{m/n(1-m/n)}{n}}\leqq\frac{M}{N}\leqq\frac{m}{n}-k_\alpha\times\sqrt{\frac{N-n}{N-1}\times\frac{m/n(1-m/n)}{n}} \qquad (3)$$

ここで，k_α は平均値0，分散1の正規分布の α 点（$\alpha=5$ の k_α は 1.96）．

ただし，サンプルの大きさ n はある程度大きいこと．目安としては，$n\geqq20$ で有効．

3.3 捕獲・再捕獲法を適用する場合の課題

捕獲・再捕獲法を実際にソフトウェアの潜在バグ数の推定に適用する場合には，次の課題を克服する必要がある．

課題1：埋込みバグの選定法

潜在バグ数の推定精度を良くするには埋込みバグの選定が重要である．2章でも述べたように式(1)が統計学的に成立する条件は，埋込みバグ（M）と未知の潜在バグ（$N-M$）の捕獲率がどの集合をとっても差がないようにする必要がある．

課題2：埋込みバグによるトラブルの発生対策

検査中に埋込みバグに遭遇すると，トラブルが発生し，本来の検査の進捗に影響する．特に，システム・ダウン，プログラム・ループなどの現象を引き起こす埋込みバグは影響が大きい．また，検査グループは埋込みバグの存在を知らないことが前提であるため，検査工程で埋込みバグが発見されると，検査担当者はバグの1次解析まで行ってしまう．また，バグの埋込みとバグの除去のためにプログラム自体に手が入り，余分な工数を必要とすると共にプログラムの修正ミスを招く恐れもある．

4. 埋込みバグの選定法

3章で指摘した課題1について考察する．バグ捕獲・再捕獲法を用いた潜在バグ推定の理論的根拠となるのは，埋込みバグと潜在バグの捕獲率に差がないことである．そのためには，次のことを考慮する必要がある．

①埋込みバグの分布：検査対象プログラムのどの部分に埋め込めばよいか？

②埋込みバグの摘出難易度：バグ摘出難易度としてどの程度のものを選定すればよいか？

この問題は，再捕獲時に実施する検査の内容に関係する．

再捕獲時の埋込みバグと潜在バグの再捕獲率を同じにするには，**図4**に示すように X, Y, Z 軸に着目し

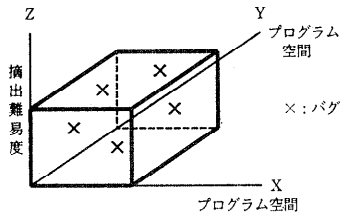


図 4 バグの存在する空間
Fig. 4 Existing space of bugs.

て，埋込みバグと潜在バグの疎密が理想的には同じである必要がある．ここで X, Y 軸が検査対象プログラムの空間，Z 軸がバグ摘出難易度と考えればよい．このことは魚の捕獲時に，漁獲量が網をセットする場所と網のセットする深さに依存することを考えると，その意味するところが容易に理解できる．

また，推定結果を統計的に扱うには，

③埋込みバグの数：何件くらい埋め込めば，精度が十分といえるか？

という問題もある．しかし，これは純粋に統計上の問題であり，ここでは考察を省略する．

4.1 埋込みバグの分布に関する考察

プログラムに混入するバグ数は，プログラム規模，設計者やプログラマのスキル，プログラムの複雑度，使用言語などによって異なり，バグの分布は**図5**（a）に示すように疎密ができる[20]．6章の実験対象プログラムの実例を**図6**に示す．ソフトウェアの品質を管理する場合は，プログラムの作成者，機能，作成言語を考慮して，プログラム全体をある大きさに分割して管理するのが一般的である．われわれはこの単位を品質管理単位（以下 PU : Product Management Unit）と呼んでいる．バグの疎密を考慮しないで図5（b）のように，各 PU のプログラム規模に比例させてバグを埋め込む（プログラム規模比例方式）と，再捕獲時の検査が検査対象プログラム空間に対し，網羅的に行われない限り，埋込みバグと潜在バグ数の捕獲比率が同じにならない．検査は一般に期間および投入工数に制約があり，検査対象プログラム空間に対して網羅的に実施できないことが多い．そのため，検査が検査対象プログラム空間の一部に偏った場合には，バグ数推定結果が真の値と大きくかい離することになる．このことは，図5（b）における B-6 と B-7 の埋込みバグ数と潜在バグの比率を考えてみれば，明らかである．このため，検査対象プログラム空間において，埋込みバグと潜在バグとの分布ができるだけ一様になることが望ましい．しかし，潜在バグ数は未知数である．この解決策としては，近似的に図5（c）に示すように，デバグ，試験工程で摘出したバグ数に比例して，埋込みバグを検査対象プログラムに埋め込む方式（摘出バグ数比例方式）がよい．この考え方は，バグが多く摘出された PU はその後も引き続き多くのバグが摘出されるという前提に基づいている．この仮説はわれわれの長いソフトウェア開発経験に基づいた経験則である．次にその一例として，6章の実験対象のプログラムの場合を示す．

図7が PU に着目してデバグ工程で摘出した Ks 当たりに正規化したバグ数と試験工程以降で摘出した Ks 当たりに正規化したバグ数との散布図である．相関係数は 0.62 であった．また，同様に単体デバグと結合デバグを X, Y 軸とした散布図の相関係数は 0.78，結合デバグと総合試験を X, Y 軸とした散布図の相関係数は 0.61 であった．統計学上相関係数が 0.6 以上は相関ありということであり，現時点では摘出バグ数比例方式を採用するのがよいといえる．さらによい手法があるかというサーベイは，今後の課題である．

4.2 埋込みバグ摘出の難易度

埋込みバグと潜在バグの捕獲率に差がないようにするには，埋込みバグの分布に加え，再捕獲時のバグの摘出難易度も埋込みバグと本来潜在しているバグとで同じレベルにする必要がある．

バグ捕獲・再捕獲法における埋込みバグとしては，実際に摘出したバグ（実バグ）の発生した条件に着目し，その部分に実バグ発生と同じ条件が成立したことがわかる仕掛けを埋め込む（実バグは修正する）方法—放流バグそのものではなく，その発生条件だけを擬似する—と実バグが発見されていない部分に故意に仕掛けを埋め込む（疑似バグ）方法とを用いる．前者の場合，再捕獲時点と時間的に近いところで検出された実バグ発生と同じ条件を仕掛けにセットすることにより，埋込みバグの摘出難易度はその後検出される実バグの摘出難易度とほぼ同じにできる．後者については，推定時点の実バグの摘出難易度に十分類似させる必要がある．摘出のあまりにも容易な疑似バグばかりを埋め込むと，再捕獲時に疑似バグばかりが摘出されて推定結果が望ましくないものになるからである．この難易度の設定については，他の実バグの摘出難易度を分析し，同じレベルの疑似バグとする必要がある．例えば，エッジに着目した構造テスト実施済のプログ

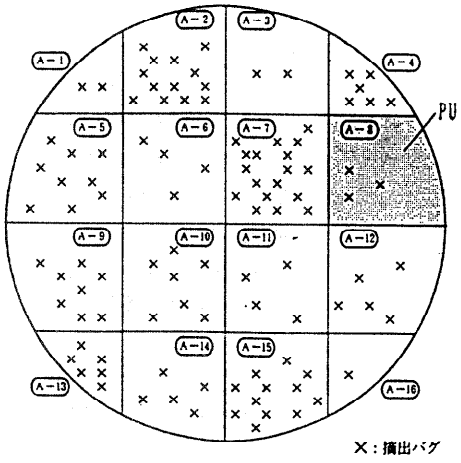


図 5(a) バグの疎密
Fig. 5(a) Condensation and rarefaction of bugs.

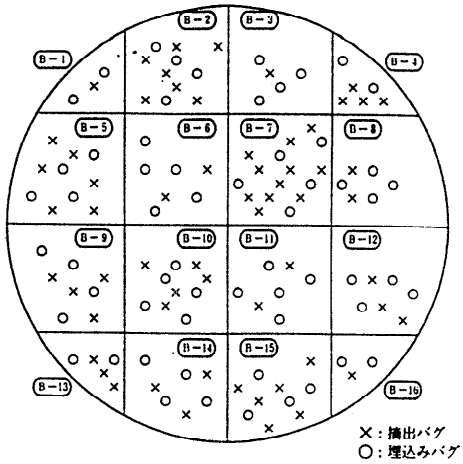


図 5(b) 一様な埋込み法（プログラム規模比例方式）
Fig. 5(b) Method of setting seeded bugs uniformly.

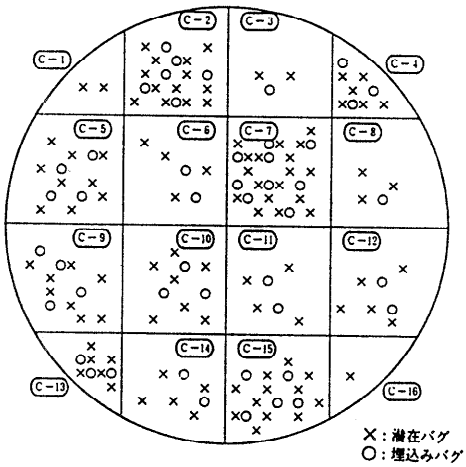


図 5(c) バグ摘出傾向を考慮した埋込み法（摘出バグ数比例方式）
Fig. 5(c) Method of setting seeded bugs in proportion to the detected part of bugs.

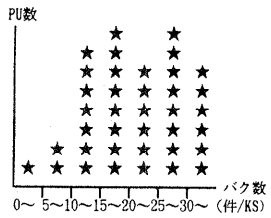


図 6 バグの疎密の例
Fig. 6 Example of condensation and rarefaction of bugs.

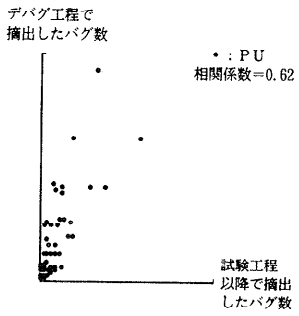


図 7 散 布 図
Fig. 7 Scatter diagram.

ラムが対象であれば，エッジを通過しただけで埋込みバグが発生するような条件を設定することを避けるなどの考慮が必要である．

潜在しているバグの摘出難易度と有意差なく埋込みバグを選定する手法の確立は，今後の研究課題といえる．

5. バグ捕獲・再捕獲法支援ツール (CREDIT-Q)

ここでは，3章の課題2で述べた問題点の解決策とそのための支援ツールについて述べる．

5.1 改善点

すでに図2で示したように，埋込みバグは実際のバグを埋め込むのではなく，本来バグを埋め込みたい場所にディレール・ポイント（仕掛け）を埋め込む方法を採用している．ディレール・ポイントには標識を付与する．検査時，このディレール・ポイントを通過し，かつ，指定した条件が発生したかをチェックする．この条件が満たされたとき，埋込みバグが発生したとみなし統計をとる．この統計データをもとに式(2)，(3)を使って潜在バグ数を推定する．バグはプログラムのある部分を通過しただけでは顕在化しないケースがある．例えば，ディレール・ポイントを1,000回以上通過したときのみバグが顕在化するというように，ある条件と組み合わさって始めて顕在化するものもある．一般に，検査工程のような後ろの工程では，ディレール・ポイントを通過しただけでバグが顕在化する単純バグより，他の条件と組み合わさってバグが顕在化することが多い．

5.2 支援ツール (CREDIT-Q)

CREDIT-Q として次の機能を実現した．

（a） ディレール・ポイント検出用の不当命令（プログラム割込みを発生させる）埋込み機能

（b） バグ発生条件記述機能

（c） バグ発生条件チェック機能

プログラム割込み時のバグ発生条件をチェックし，発生条件を満足していれば，埋込みバグが発生したとみなし，埋込みバグの発生を検査担当者へ通知するとともに，統計用として埋込みバグ検出状況テーブルに情報をスタックする．

（d） 統計情報編集機能

埋込みバグ検出状況を一定時間間隔，または，オペレータの指示に従って編集出力する．

```
*DPT  ID1, PROG-A, 100
*CON  RG, , X"150", GE, GR7
```

図 8 埋込みバグの設定例
Fig. 8 Examples of control statement for setting seeded bug.

（1） 埋込みバグの設定・削除処理

埋込み場所とバグ発生条件は，図8に示すような制御文で記述する．第1ステートメントの *DPT 文では，PROG-A というプログラムの先頭から100バイトの所に ID1 という標識（識別詞）をもつ埋込みバグをセットすることを指示している．その結果，指定された場所に本来の命令コードに代わって，この点を通過したときに割込みが起こるように不当命令が埋め込まれる．元の命令コードは別のエリアに退避される．第2ステートメントの *CON 文は，バク発生条件を指示するステートメントである．バグ発生条件としては，ディレール・ポイント通過時のコンディション・コード，レジスタの内容，メモリの内容，それらの組合せなどの条件が指定できる．図8の例では，第1パラメータでレジスタの内容をチェックせよということを指示している．第2パラメータ以下でチェックの内容を指定する．ここの例では，ジェネラル・レジスタの値が16進数で150以上のとき，埋込みバグが発生したと認識せよということを示している．ディ

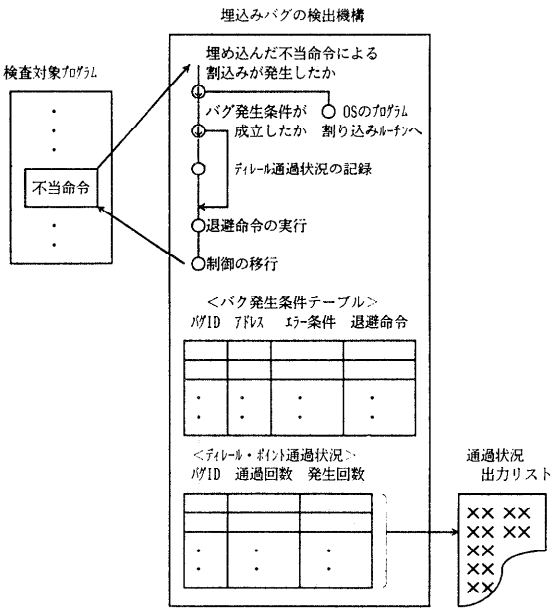


図 9 埋込みバグの検出機構の概略
Fig. 9 The seeded bugs detection mechanism.

レール・ポイントを削除したいときは，*CLR 文で指定する．例えば，識別詞が ID 1 のディレール・ポイントを削除したいときは，*CLR ID 1 と書く．このとき，識別詞 ID 1 に対応するディレール・ポイントに埋め込まれた不当命令は，本来の命令コードに戻される．

（2） 埋込みバグの検出

埋込みバグの検出機構を**図 9** に示す．不当命令が埋め込まれた部分にあった元の命令コードは，バグ ID，埋込みアドレス，バグ発生条件と共に，埋込みバグの検出機構内にあるバグ発生条件テーブルに設定される．検査時に検査対象プログラムに埋め込まれたディレール・ポイントを通過すると，自動的に埋込みバグ検出機構が動作し，バグ発生条件をチェックする．その結果は，埋込みバグ検出機構内にあるディレール・ポイント通過状況テーブルに記録される．

6. 実験と評価

バグ捕獲・再捕獲法を商用大規模オペレーティングシステム開発時の検査工程に適用し，潜在バグ数を推定した．その後，1年にわたりバグの発生状況を追跡し，本方式がソフトウェア出荷時の信頼性保証の手段として十分利用可能であることを検証した．

6.1 実験の内容

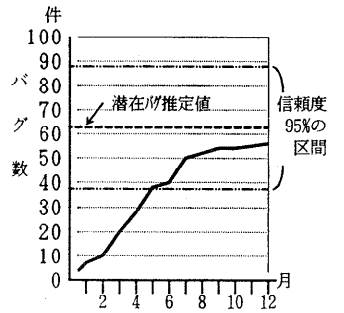
（1） 埋込みバグの選定

埋込みバグとしては，製造・試験グループが摘出したバグの中から 26 件を選択し，その他に疑似バグを 26 件作成した．埋込みバグ分布としては，摘出バグ数比例方式を採用した．

表 1 実験方法に関する諸元
Table 1 Abstract of experiment.

	内　　容
検査対象プログラム	TSS 用大規模 OS の新規作成，改造部分 約 150 *Ks*
埋込みバグ	（1） 総数：52 件 内訳 放流バグ：26 件 疑似バグ：26 件 （2） 埋込みバグ分布：摘出バグ数比例方式
期　　間	（1） 実験期間：6 カ月 （2） 推定結果の確認期間：12 カ月
テスト内容	（1） 製造工程：構造テスト中心 （2） 試験工程：機能テスト （3） 検査工程：機能テスト

表 2 実験結果
Table 2 Experimental result.

	内　　容
摘出バグ	34 件 （内訳）放流バグ：21 件 疑似バグ：13 件
推定潜在バグ数	63 件 $\left[\begin{array}{l} N=\dfrac{52\times 34}{13}=136 \\ \therefore B=136-52-21=63 \end{array}\right.$
確認期間のバグ発見推移	

（2） 実験期間

実験は検査工程でサービス開始後と同様の環境で実施した．その後，推定結果の検証のため約 1 年間バグの発生状況を追跡調査した．実験方法に関する諸元を**表 1** に示す．

（3） 検査グループとの関係

検査側は埋込みバグの発生条件に対して何も知識がないのが前提である．そのため，実験では埋込みバグの内容を一切伏せ，検査グループにバグ捕獲・再捕獲法の実験を行っていることだけを知らせた．

6.2 実験結果の考察

実験結果を**表 2** に示す．

（1） 推定と検証

推定結果の潜在バグ数は 63 件であった．その後，1 年経過するまでに 56 件のバグがフィールドから発見された．この 1 年間のバグ発生傾向，商用サービスでのシステムの使われかたから判断して，実験に供したソフトウェアのバグは十分枯れたものと推測される．また，このことから本方式による潜在バグ推定の精度は高いと判断できる．

（2） 推定精度の評価

バグ捕獲・再捕獲法の推定精度を評価するため，ロジスティック曲線，および，ゴンペルツ曲線による推定法と比較する．

評価法としては，式(4)のようにそれぞれの方法で求めた潜在バグ予測と実際の総バグ数の差分（誤差

率）を用いた．

$$\text{誤差率}=\frac{\text{推定潜在バグ数}-\text{摘出済バグ数}}{\text{総バグ数}}\times 100 \quad (4)$$

式(4)において，真の総バグ数は永久にわからない．ここでは総バグ数として，サービス開始1年後までに摘出したバグ数とした．通常，アプリケーション・プログラム（AP）は随時，日々，月々，年度ごとのサイクルで走行するのが大半である．そのため，商用開始後1年たつとAPは十分使い込まれ，摘出バグ数も十分真の総バグ数に漸近すると考えられる．この実験対象プログラムの総バグ数は2,850件である．

ロジスティック曲線，および，ゴンペルツ曲線を用いた推定法では，**図10**に示すバグ摘出率90%，95%，97%，98%の時点までのバグ発生履歴をそれぞれロジスティック曲線，および，ゴンペルツ曲線に当てはめて，それぞれの時点での推定潜在バグ数を求めた．また，バグ捕獲・再捕獲法による潜在バグ推定時期はバグ摘出率98%の時点であった．これらの結果を式

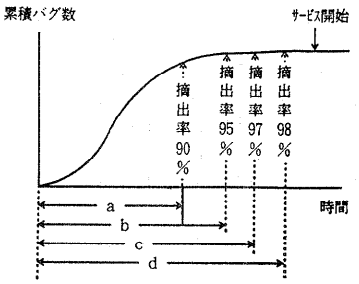


図10 ソフトウェア信頼度成長曲線適用モデル
Fig. 10 Model for applying software reliability growth curve.

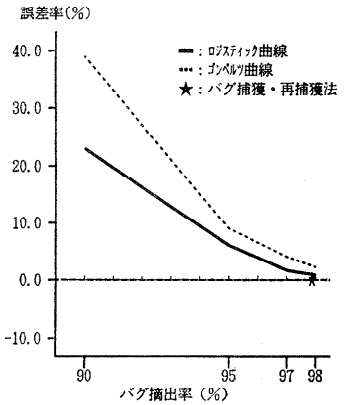


図11 各方式の誤差率
Fig. 11 Error percentage in each method.

(4)に入れグラフ化したのが**図11**である．実線がロジスティック曲線，点線がゴンペルツ曲線，★印の点がバグ捕獲・再捕獲法の誤差率である．ロジスティック曲線，ゴンペルツ曲線とも推定時点がバグ摘出率90%から98%に向かうに従って誤差率が大幅に改善されることがわかる．バグ摘出率98%の時点はロジスティック曲線で0.8%，ゴンペルツ曲線で2.3%の誤差率である．それに比べ，バグ捕獲・再捕獲法の誤差率はバグ摘出率98%の時点で0.2%と非常に低い値である．バグ捕獲・再捕獲法の予測精度が良いのは，①成長曲線当てはめ法がデバグや試験時の投入稼働の時間的ばらつきやデバグ，試験のやり方，チェック項目の実施順番などに影響されやすく，それが無視できないレベルにある．一方，バグ捕獲・再捕獲法はその影響を受けない．また，②今回の実験では推定精度に影響を与えるバグの分布，バグの摘出難易度等に関して，十分に考慮した埋込みバグの選定が行われたことによると思われる．

（3） 出荷品質評価尺度としての適用性

バグ捕獲・再捕獲法は，潜在バグ数をかなり精度良く推定し，出荷時期を決定する上で有力な客観的評価情報を与えるものであり，次のように利用できる．

①推定された潜在バグ数から，商用として十分耐えられる品質となるまでに，あとどの程度の試験工数を投入すべきかの判断が可能となる．通常，机上テスト，マシンテストでのバグ摘出効率（摘出バグ数/作業時間）は統計をとっており，潜在バグ数か予測できれば，投入すべき工数予測が可能である．また，この手法ではどの部分を試験すべきかを指摘はできないが，使用状況，バグの発生ローカリティなどを初期の工程から継続的に把握することでどこを狙うべきかは絞り込みが可能である．

②サービス開始時に求められるソフトウェアの信頼性は，開発するソフトウェアの種類によって異なる．しかし，くり返しソフトウェアの品質管理をしていれば，自分たちの開発するソフトウェアは経験的に商用開始時の潜在バグ数がどの程度だと許容範囲であるかを把握している．本方式による推定値をこの許容尺度と照らし合わせることで適切な商用開始時期を決定できる．

7. おわりに

捕獲・再捕獲法をソフトウェアバグ推定用に改善した方式を提案し，本方式を具体的に大規模ソフトウェ

アの開発に適用することにより，ソフトウェアの高信頼性を保証し，出荷時期を決定する尺度として有効であることを示した．

本方式の特徴としては，

①バグ捕獲・再捕獲法によれば，実バグをそのまま放流する必要がない．

②バグの発見履歴を収集していなくても，潜在バグの推定ができる．

③潜在バグを推定する上で，精度の高い客観的評価尺度となりうる．

点を挙げることができる．

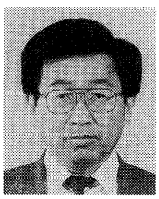
また，バグ捕獲・再捕獲法は人工的に疑似バグを対象プログラムに埋め込んでおき，試験でどの程度除去されたかにより，試験の妥当性を評価する[21]などの応用もでき，今後は，検査工程以外への適用方法等について検討していきたい．

謝辞 本検討にご協力いただいた村田紀男氏（現 NTT ソフトウェア(株)），中川勉氏（現 NTT データ(株)），林孝樹氏（現 NTT データ(株)）をはじめ，実験に協力していただいた多くの方々にこころから感謝いたします．

参 考 文 献

1) Jelinski, Z. and Moranda, P. B.: Software Reliability Research, *Statistical Computer Performance Evaluation*, Freiberger, W. (ed.), pp. 465-485, Academic Press, New York (1972).
2) Musa, J. D.: The Measurement and Management of Software Reliability, *Proc. IEEE*, Vol. 68, No. 9, pp. 1131-1143 (1980).
3) Goel, A. L. and Okumoto, K.: Time-Dependent Error Detection Rate Model for Software Reliability and Other Performance Measures, *IEEE Trans. Reliability*, Vol. R-28, No. 3, pp. 206-211 (1979).
4) Littlewood, B.: Theories of SoftwareReliability: How Good Are They and How Can They Be Improved?, *IEEE Trans. Software Eng.*, Vol. SE-6, No. 9, pp. 489-500 (1980).
5) Ohba, M.: Software Reliability Analysis Models, *IBM J. Res. Dev.*, Vol. 28, No. 4, pp. 428-443 (1984).
6) Yamada, S. and Osaki, S.: Discrete Software Reliability Growth Models, *Applied Stochastic Models and Data Analysis*, Vol. 1, No. 1, pp. 65-77 (1985).
7) 山田：ゴンペルツ曲線を用いた確率的ソフトウェア信頼度成長モデル，情報処理学会論文誌，Vol. 33, No. 7, pp. 964-969 (1992).
8) Mills, H.: On the Statistical Validation of Computer Programs, IBM FSD Report 72-6015 (1972).
9) 当麻：超幾何分布にもとづくソフトウェア残存フォールト数推定モデル，情報処理，Vol. 31, No. 12, pp. 1641-1646 (1990).
10) 伊土ほか：「Capture & Recapture 法」による潜在バグの推定法とその応用，情報処理学会ソフトウェア工学研究会，19-1 (1981).
11) Goodenough, J. B. and McGowan, C. L.: Software Quality Assurance: Testing and Validation, *Proc. IEEE*, Vol. 68, No. 9, pp. 1093-1098 (1980).
12) Duran, J. W. and Wiorkowaski, J. J.: Capture-Recapture Sampling for Estimating Software Error Content, *IEEE Trans. Software Eng.*, Vol. SE-7, No. 1, pp. 147-148 (1981).
13) 坂本ほか：バグ残留法によるプログラムの品質測定，第21回情報処理学会全国大会論文集，7C-8 (1980).
14) 若山ほか：復元バグ再捕獲法によるデバグ支援ツール評価，情報処理学会ソフトウェア工学研究会，80-14 (1991).
15) 中村ほか：埋込みモデルにおけるエラー埋込みの自動化について，情報処理学会ソフトウェア工学研究会報告，80-15 (1991).
16) Lincoln, F. C.: Calculating Waterfront Abundance on the Basis of Banding Returns, U. S. Dept. Agric. Circ., No. 118, pp. 1-4 (1930).
17) Schnabel, Z. E.: The Estimation of the Total Fish Population of a Lake, *Amer. Math. Mon.*, Vol. 45, pp. 348-350 (1938).
18) 吉原：生物資源量の推定法，日本生態学会誌，Vol. 4, pp. 177-182 (1955), Vol. 5, pp. 37-41 (1956).
19) 脇本和昌：身近なデータによる統計解析入門，森北出版 (1973).
20) 伊土ほか：プログラムのスキルを考慮した作成分担に関する一提案，第22回情報処理学会全国大会論文集，1C-2 (1981).
21) Ohba, M.: Software Quality=Test Coverage×Test Accuracy, *Proc. 6th ICSE*, pp. 287-293 (1982).

(平成4年8月24日受付)
(平成5年9月 8 日採録)



伊土 誠一（正会員）

1947年生．1969年北海道大学工学部電子工学科卒業．1971年同大学院修士課程修了．同年日本電信電話公社入社．以来，DIPS 計算機用オペレーティング・システムの研究実用化と大規模計算機システムの開発，ソフトウェア工学，および，データベースの研究等に従事．現在，NTT 情報通信網研究所研究企画部長．電子情報通信学会会員．