ソフトウエア工学 19－1
（1981. 7. 17）

「Capture & Recapture 法」による潜在バグの推定法とその応用

伊土誠一　村田紀男　中川勉
（電電公社　横須賀電気通信研究所）

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定量的に推定できれば，①デバグ，試験時に最適な工数を投入でき，効率のよい品質保証が可能となる，②ソフトウェア出荷品質の予測ができ適切なソフトウェア開放時期を設定できる．

従来，潜在バグ数は，開発途中までのバグの発生傾向からゴンペルツ曲線，ロジスティック曲線をあてはめて予想する方法，あるいはソフトウェア開発者が経験的に予想したバグ数を逐次補正してゆく方法等が用いられてきた。しかし，これらの方法はマクロ的であり精度が悪いため単体デバグ工程における目標設定と目標達成のためのマクロな工数計画には有効な方法であるが，最終的なソフトウェア出荷時におけるミクロな判断材料 ── 潜在バグ数予測とこれに基づくソフトウェアの開放可否の判断 ── としては有効であるとは云えない。

筆者らは，ソフトウェア開発の最終工程頃における潜在バグ数を精度良く推定するために，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 法）」をソフトウェアのバグ数推定用に改善した方法を提案するとともに，この方法を大規模オペレーティングシステムの開発工程に適用し，その有効性を検証した。

この「Capture & Recapture 法」と同じ様な手法をソフトウエアの品質把握に適用するアイディアは，1972年にMills[1]によって提案されており，近年国内外で「Capture & Recapture 法」に関連した議論がなされはじめている。[2]~[5]

本稿では，2章で本方式の位置付けを，3章で「Capture & Recapture 法」をソフトウェアのバグ推定に適用した場合の問題点とその改善策を述べる。更には，「Capture & Recapture 法」の推定を最も左右すると考えられる埋め込みバグ（放たれた既知のバグ）の選定方法について論ずる。4章では，3章で述べた考え方に従って本方式を商用システム用の大規模オペレーティングシステムの開発時に適用した結果について述べる。

2. ソフトウェアの開発工程と本方法の位置付け

ソフトウェアは，通常，図1の工程に沿って開発される。この間，品質管理は，計画段階からそのソフトウェアが商用に供されライフサイクルが終了するまで一貫して実施されなければならない。通常，ソフトウェア・ライフサイクルの中の設計・製造・試験の段階においては，いかに効率よく，かつ網羅的にテストを行なうかの観点から，

① パス解析による未テストパスの抽出とそのテスト[6]　② 限界値・境界値に着目したテスト　③ MTSなどのテスターによる負荷テスト[7]などが採り入れられており，品質安定の努力が続けられる。しかし，これらのテストの結果，一定の数のバグが摘出されても商用サービスに供するに足る程度十分に安定したかどうかの判断は，バグ発生の累積曲線，試験でのMTBFなどに頼っての総合判断という主観的なものであった。この判断のために商用システムの実際の呼を使って新システム上で再実行し，実使用形態にチューンして，安定性を確認しよう

という試み[8]もあるが，システム更改時或いは旧システムから新システムへの上方向互換のある機能に対しては有効であるとしても，新規の機能については，この方法も有効ではない。　また，ソフトウェアのバグはテストをしなければ検出されず，或る管理単位（たとえば，コンポーネント，モジュールなど）に対し，製造段階で誤って品質良好と判断し，その後その管理単位に対し特別なテスト工程が計画されなければバグ累積曲線は極めて良好の呈を示し，後にまで判断を誤らせることとなる。　筆者らは，ソフトウェアの開放時期の制御を潜在バグ数の多少で行なうという観点から，この推定を精度良く行なうために Capture & Recapture 法をソフトウェア品質保証の道具として用いる改善手法を提案する。

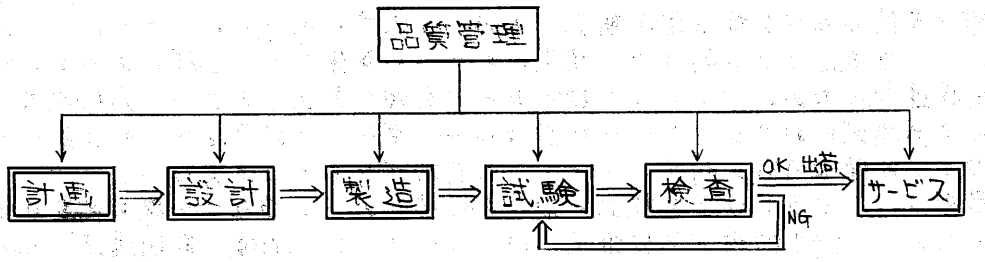


図1　ソフトウェア開発工程

3. ソフトウェア品質推定のための Capture & Recapture 法

3.1 Capture & Recapture 法とは

Capture & Recapture 法とは，記号放逐法（Mark & Release method）とも呼ばれ，最初捕えた動物に記号（標識）をつけてもとの場所に放し，その再捕率によって総個体数（N）を推定する方法である。[9]~[11]もし記号個体と未記号個体の捕獲率に差がなければ

$$N = Mn/m$$

ここで，M＝記号個体数，n＝才2回目の捕獲個体数，m＝才2回目の捕獲記号個体数である。

(1) ソフトウェアにおける推定法

① 試験で発見されたバグ，あるいは故意に作ったバグをプログラムに埋め込み，そのプログラムを検査グループに検査させる。　この様なバグを埋め込みバグと呼び，前者の手順で埋め込まれたバグを放流バグ，後者を移植バグと呼ぶ。

② 埋め込みバグの所在を知らない検査グループが検査しバグを摘出する。

③ その結果から総バグ数（N）を算出する。

$$N = Mn/m \quad \cdots\cdots (1)$$

ここで　M＝埋め込みバグ件数

n = 検査で摘出された総バグ件数
m = 検査で摘出された埋め込みバグ件数

故に，その時点の埋め込みバグを除いた潜在バグ数(l)は，

$$l = N - M - (n - m) \quad \cdots\cdots\cdots (2)$$

信頼度95%のNの範囲は，次式で表わされる。[12]

$$\frac{m}{n} - 1.96 \times \sqrt{\frac{N-n}{N-1} \times \frac{m/n(1-m/n)}{n}} \leqq \frac{M}{N} \leqq \frac{m}{n} + 1.96 \times \sqrt{\frac{N-n}{N-1} \times \frac{m/n(1-m/n)}{n}} \quad \cdots\cdots (3)$$

(2) 本方法の問題点

Capture & Recapture 法をソフトウェアの品質評価尺度として使用する際に次の問題がある。

(問題点 1)

埋め込みバグに遭遇するとトラブルが発生し，検査の進捗に影響を及ぼす。検査グループは，埋め込みバグの所在を知らないのが前提であるのでトラブルの一次解析までは行ってしまうと考えられる。　特に，システムダウン，ループとなる埋め込みバグは影響が大きい。

(問題点 2)

潜在バグ数の推定精度を良くするような埋め込みバグの選定が難しい。　統計学的にこの推定が成立するためには，埋め込みバグと未知のバグ（実バグと呼ぶ）の捕獲率がどの集合をとっても差がない様にする必要がある。

3.2　改善 Capture & Recapture 法（パッチディレール方式）

改善 Capture & Recapture 法は，前述の問題点1を解決し，Capture & Re-Capture 法をソフトウェア向きに改善した方式である。

(1) 改善点

本方式は，埋め込みバグとして放流バグ，移植バグをそのまま使うのではなく，擬似バグを使う方法である。　擬似バグ方式は，バグ自体は修正してしまい，バグの代りとなる標識としてディレールを埋め込み，ディレールを通過するときにその旨のメッセージをコンソールに出力する方式である。　そのロジックを図2に示す。

この方式は，試験対象プログラムにディレール用のSVC命令と擬似バグを識別するための識別詞を埋め込むだけで済む。　また，コンソールへ

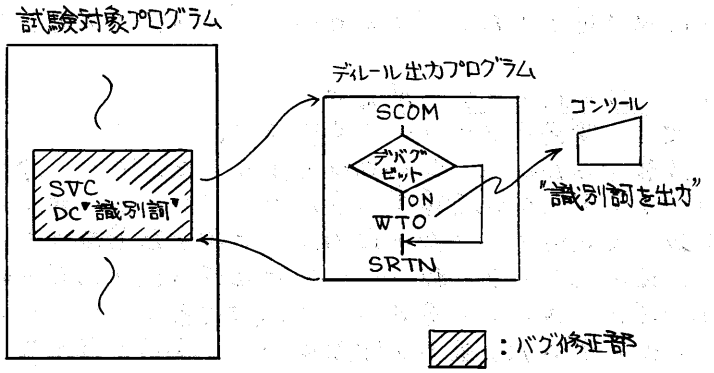


図2　パッチディレール方式のロジック

のメッセージ出力をオプション化することで試験完了後はオプションの無効化（たとえばコンソールコマンド）で取り外しができるし，ＳＤＣを発行すること自体を避けたい時は単にＳＤＣをＮＯＰ化するだけで可能である。　また，ディレールに遭遇しても擬似バグによって試験の進捗を阻害することがなく検査工程での評価方法に適していると云える。

(2) 推定法

潜在バグ数の推定は，検査で摘出された埋め込みバグ件数がコンソールに出力された識別詞の種類である点が異なるのみであり，計算法は前述の式 (2) ～ (3) による。

(3) 考慮すべき点

この方式では，バグは不良パスを通過するときに起こるという前提に立っており，擬似バグも或るパスを不良と見立てることで埋め込まれている。

しかし，実際にはパスのある一部が全面不良という単純なものだけではなく，同一パスでもデータ種別によってバグとなることもある。　たとえば，あるパスでn＝1～5の値をとるとして，n＝5のときのみバグとなる――オーバフローが生ずるとか――ケースである。　一般に，試験工程が進めば，よく取り得る値，たとえばn＝1～4は充分テストされたがn＝5のケースのみ未テストということがある。　この様なとき，このパスにディレールを埋め込むことはn＝1～5の全てのケースを擬似バグ扱いとしてしまうため結果的には，埋め込みバグの検出が実バグの検出より捕獲率で高くなる危険がある。

この危険を避けるために次の様な考慮が必要である。

① 埋め込みバグをどの時点で挿入するか（工程）

挿入時点が，テスト未のパスを通ることによって単純にバグ発見ができる程度の工程（単体デバグ，結合試験初期）であれば，上記の問題は生じない。

② データの条件を考慮したディレールの設定

更に進んだ工程では，データの条件を考慮したロジックを組み込む必要がある。　上記の例ではn＝5のときのみディレールが起こる様にする。

尚，ＯＳでは，殆んどが条件によってパスが分岐するロジックになっており，筆者らが提案しているようにパスに擬似バグを埋め込む方法（そのパスを通過すれば必ず検出されることを意味している）で実用上差支えないと考えられる。

3.3　埋め込みバグの選定法

次に，問題点2について考察する。

改善 Capture & Recapture 法を用いた潜在バグ推定の根拠となるのは，埋め込みバグと実バグの捕獲率に差がないこと，云い換えればどの部分をとっても実バグ数と埋め込みバグ数の比ならびに検出難易度に差がないことである。

(i) 埋め込みバグの数――何件位埋め込めば精度が十分であると云えるか

(ii) 埋め込みバグの分布――試験対象プログラムのどの部分に埋め込めば良いか

(iii) 埋め込みバグ摘出の難易度――摘出の難易度としてどの程度のものを選定すれば良いか

(i) の埋め込みバグ数に関しては純粋に統計学上の問題であり，ここでの考察か

らは除く。

3.3.1 埋め込みバグの分布

(1) バグ分布の疎密

プログラムに含まれるバグ数は，そのプログラムを作ったプログラマのスキル，プログラムの複雑度・使用言語等によって異なり，バグ分布に疎密ができる（図3－1）[13][14]。このため，プログラムの品質管理は，作成者・機能・作成言語を考慮して或る大きさに分割した管理単位に注目して個別に行なうことが必要である。我々はこの単位を品質管理単位（以下PU：Product Unit）と呼んでいる。

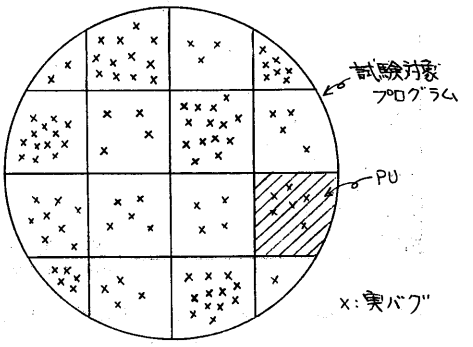


図3-1 バグの疎密

埋め込みバグの選定にあたっては，このPU毎の疎密を考慮する必要がある。すなわち，この発生ローカリティを考慮しないで，各PUの規模比に等しい割合で埋め込みを行なうこと（図3－2）は捕獲率を一定にすることにならない。

即ち，図3－2で試験がランダムに行なわれたとすると明らかにA－1は埋め込みバグに比べて実バグの検出割合が多くA－2には実バグ割合が少ないだろう。このため，製造工程でのバグ発生頻度／PUを考慮した埋め込み法（図3－3）が必要である。図3－3では，どのPUの母集団をとっても埋め込みバグ／実バグの割合は一定であり，試験がどのPUに偏っても捕獲率を一定にできると言える。

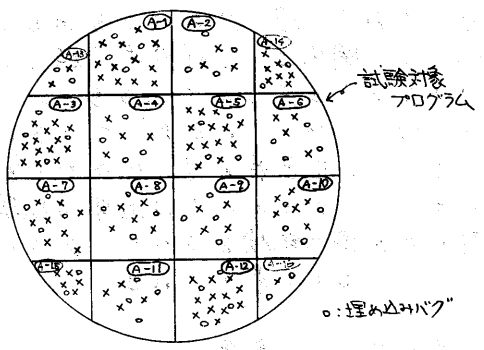


図3-2 一様埋込み法（均等方式）

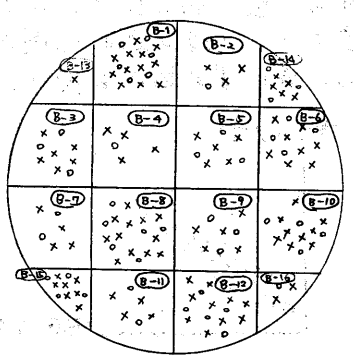


図3-3 発生傾向をみた埋込み法（比例方式）

この考え方は，プログラムの製造サイクルの中で，製造工程で品質の悪いPUはその後の工程でも引続いて良くバグが検出されるという前提に立っている。これは本当だろうか？この前提となる仮説を確めるために次の様な実験を行なった。

(a) 実験

PUに着目して，単体デバグでのバグ摘出状況と総合試験でのそれの相関，

結合試験までと検査期間との相関，これらを順次求める．

(b) 結果

工程間の相関を図4－1，図4－2に示す。

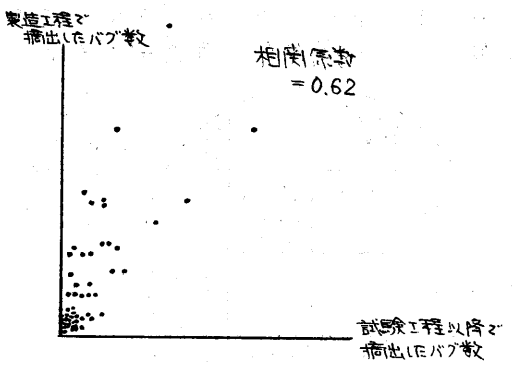


図4-1 PUに着目した製造工程とそれ以降で摘出したバグ数の相関

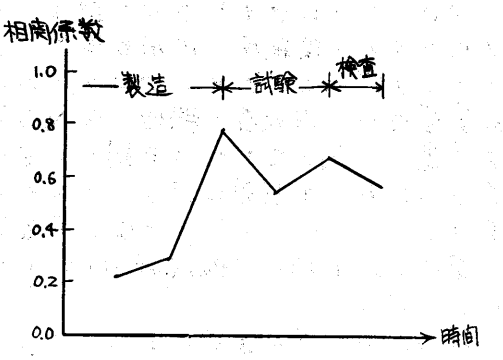
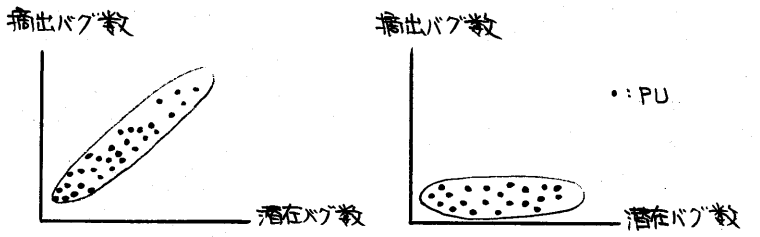


図4-2. PUに着目した相関係数の推移

この結果が教えるように，工程の前半では相関が強くないが後半ではかなり強くなっており，埋め込みバグ分布も工程後半で推定する場合には比例方式が望ましい。この関係は，工程後半（結合テストを終えた段階）では各PUについて一通りのテストを行なっているものの，最初に作り込まれたバグがこの時点ではまだとり切れてないことを示していると言える。個人ベースのデバッグが中心になっており，担当者の思い込みなどが是正されていないことを示していると推測される。これらのバグは通常は試験工程で取り除かれると考えられるので（取り除かれる様に試験をすべきであるので），試験完了後の状態では除々に相関が低くなってゆくと考えられる。

(2) 埋め込みバグ分布

埋め込みバグの分布は，その時点でのバグ発生ローカリティに比例させるのが望ましい。即ち，図5(a)の様に，バグ分布に疎密がある場合にはバグ摘出履歴を考慮して疎密をつけて埋め込む比例方式が望しく，工程末期に埋め込むものについては図5(b)の様に埋め込みを均一にした均等方式に近づけるのが望しい。



(a) 比例方式が有効な例　　(b) 均等方式が有効な例

図5 PUに着目した潜在バグ数と摘出バグ数の関係

3.3.2 埋め込みバグ摘出の難易度

改善 Capture & Recapture 法では，埋め込みバグとして検出済み実バグの発生箇所に着目し，そのパスにディレールを埋め込む（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――と故意にパスにディレールを起こせる（擬似バグ）方法とを用いる。 前者の場合，推定時点とほぼ時間的に近いところで検出された実バグを基としているので，埋め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特に差異はない。 後者については，推定時点の実バグの摘出難易度に充分類似させる必要がある。 摘出の余りにも容易な擬似バグばかりを埋め込むと，再捕獲時に擬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，同じレベルの枝のパスに埋め込みバグを挿入する必要がある。

4. 実験と評価

改善 Capture & Recapture 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，一年以上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品質評価の判断情報として充分有効であることを検証した。

4.1 実験方法

(1) 埋め込みバグの選定

埋め込みバグは，製造・試験グループが摘出したバグの中から変換したもの26件，移植バグ26件を選定した。 埋め込みバグ分布は比例方式とした。

(2) 実験期間

実験期間は，検査工程6か月間で行ないその最後に潜在バグ数を推定し，その後，検証のため約1年間バグの発生状況を追跡調査した。 以上の実験方法に関する諸元を表1に示す。

(3) その他の留意点

この実験では，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り，検査グループのリーダ以外に情報を知らせなかった。

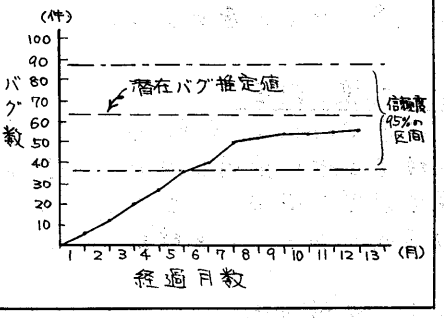
表1. 実験方法に関する諸元

	内容
試験対象プログラム	商用大規模オペレーティングシステムの新規作成，改造部分に当る約150ks
埋め込みバグ	(1) 総数：52件（擬似バグ） 内訳 { 放流バグ：26件 / 移植バグ：26件 } (2) バグ分布：比例方式
期間	(1) 実験期間：6ヶ月 (2) 結果の確認期間：12ヶ月
テスト内容	(1) 製造工程：構造テスト (2) 試験工程：機能テスト (3) 検査工程：機能テスト

4.2 実験結果と考察

実験結果を表2に示す。

表2 実験結果

	内容
摘出バグ	34件 内訳 { 実バグ：21件 埋込みバグ：13件
推定潜在バグ数	$N = \frac{52 \times 34}{13} = 136$ 件 故に 算定時点の潜在バグ数(ℓ)は $\ell = 63$ 件 (=136-52-21)
確認期間のバグ発生推移	

(1) 推定と検証

検査工程の最後に予想した推定潜在バグ数は63件であったが，その後，現局での商用テスト，商用開始後（現在8ヶ月経過）一年経過した時点で56件が検出されている。この一年間のバグ発生傾向，商用によってほとんどのパスが網羅されているだろうという判断からみると，実験に供したソフトウェアの総バグに充分漸近したと考えられる。

この事から，本方式による潜在バグ推定の精度は高いと判断される。

(2) 工数

改善 Capture & Recapture 法を実施するには，

① ディレールを発見し，識別詞を出力するためのマクロルーチンの作成
② 埋め込みポイントの決定
③ ディレールポイントへの埋め込み

が必要であるが，定常作業は簡単なパッチ修正でありこのための稼働増は軽微である。

(3) 出荷品質評価尺度への適用性

従来の出荷品質評価は，バグ累積曲線・MTBFなどのデータから勘と経験で行なっており，判断が主観的になることを避けられなかった。

これに対し改善 Capture & Recapture 法は，潜在バグ数をかなり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，この結果を次の様に利用できると考える。

① 推定された潜在バグ数から，商用として充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も可能となろう。この手法では，どの部分を試験すべきかを指摘はしないが，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ある程度絞り込みが可能であり，机上テストのバグ摘出効率（摘出バグ数／作業時間）などから投入すべき工数予測も可能である。

② OSのように高品質を要求されるソフトウェアでは，商用開始時の潜在バグ数は経験では1件／20KS程度が許容値と考えられる。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よう。

今後，益々高信頼性を要求される大規模ソフトウェアシステムにとって，本方式の適用が有効となることを期待する。

5. おわりに

Capture & Recapture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，ソフトウェア出荷時期を決定する品質評価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 改善 Capture & Recapture 法によれば，実バグをそのまま放流する必要がない。

② 潜在バグ推定の工数が少なくて済む。

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得る。

点を挙げることができる。

今後は，評価データの充実により事例としての証明を更に重ねること，擬似バグの難易度決定を容易にする手法の検討，検査工程以外への適用方法等について検討してゆく必要がある。

終りに，本研究の機会を与えられ，また有益な御指導を賜った横須賀通研 伊吹デ処部長，高村統括調査役，川野辺研究室長に深く感謝の意を表する。 又，本実験に協力された研究所，データ通信本部ならびに共同研究各社（日本電気株式会社，株式会社日立製作所，富士通株式会社）の関係各位に感謝する。

[参考文献]

1) Mills H. : On the Statistical Validation of Computer Programs, IBM FSD Report 72-6015, 1972
2) JOHN B. GOODENOUGH and CLEMENT L. McGOWAN : Software Quality Assurance : Testing and Validation, Proc. IEEE, 68, 9, pp. 1093-1098 (Sept. 1980)
3) JOE W. DURAN and JOHN J. WIORKOWASKI : Capture-Recapture Sampling for Estimating Software Error Content, IEEE Trans. Software Eng., SE-7, 1, pp. 147-148 (Jan. 1981)
4) 篠田他： 制御欠陥と機能試験の妥当性評価，情報処理学会 第22回全国大会 5C-7, 1981
5) 坂本他： バグ残留法によるプログラムの品質測定，情報処理学会第21回全国大会 7C-8, 1980
6) L.G. Stuchi : New Directions in Automated Tools. For Improving Software Quality, Current Trends in programming Methodology. Vol. 2, Prentice-Hall.
7) 拝原他： 多端末試験システム，通研実報，28, NO.12, pp. 2725-2743, 1979
8) 中川豊他： TSSにおけるデグレード試験方法，電子通信学会 昭和54年度部門全国大会，379, 1979
9) Lincoln, F.C. : Calculating Waterfowl Abundance on the Basis of Banding Returns, U.S. Dept. Agric. Circ., NO. 118, 1930, pp. 1-4
10) Schnabel Z.E. : The Estimation of the Total Fish Population of a Lake, Amer. Math. Mon., Vol 45, 1938, pp. 348-350
11) 吉原友吉： 生物資源量の推定法，日本生態学会誌 Vol. 4, 1955, pp. 177-182, Vol 5, pp. 37-41
12) 脇本和昌： 身近なデータによる統計解析入門，森北出版K.K., 1973
13) 伊土誠一他： プログラマのスキルを考慮した作成分担に関する一提案，情報処理学会 第22回全国大会 1C-2, 1981
14) 田村悦夫： 通信制御ソフトウェア開発におけるプログラマ個人差の分析，情報処理学会研究会 ソフトウェア工学 3-3, 1977