ソフトウェア工学　80－14
（１９９１．７．１９）

復元バグ再捕獲法によるデバグ支援ツール評価

若山　博文、　太田　一朗
（ＮＴＴ情報通信処理研究所）

瀬田　徳雄、　北野　和清
（ＮＴＴソフトウェア㈱）

ＣＡＳＥツール評価の一環として復元バグ再捕獲の考え方によるデバグ支援ツール評価法を提案する。提案する方法では、あるデバグ支援ツールを用いて解決されたプログラムのバグが、バグ記録に基づいて、そのプログラムの最新のソースモジュール中に復元・放流される。そのソースモジュールは、比較評価対象の新デバグ支援ツールを用いてデバグされる。これによりデバグ支援ツールとして最も重要な特性であるバグ検出力およびバグ検出効率の評価を、新しいツールの試行のみで効率よく行うことができる。本稿では、復元バグ再捕獲による評価の考え方と共に、単体デバグ支援ツールを対象として実施した試行実験の結果について述べる。

DEBUGGING TOOL EVALUATION THROUGH RECAPTURE OF THE RESTORED BUGS

Hirofumi Wakayama, Ichiro Ohta,　　Norio Seta, Kazukiyo Kitano

NTT Communications and Information Processing Laboratories

NTT Software Corporation

A method for debugging tool evaluation, based on the concept of recapture of the restored bugs, is proposed. In this method the bugs of a program, that were detected and fixed by using the former debugging tool, are restored into the latest source modules of the program according to the bug fixing records. The source modules with the restored bugs are then debugged by using a new tool which is to be evaluated. By following this method, most important features of debugging tools, namely bug detection capability and bug detection efficiency, can easily be measured through only trial use of the new tool. In this paper the method will be described and the result of a trial experiment discussed which is done in the target of a unit testing tool.

1．はじめに

ワークステーション（WS）技術の進展を背景として、各種のCASEツールを導入したWSベースのソフト開発環境が構築・導入が進みつつある[1]-[4]。

これらを、特に大規模な実用化ソフト開発プロジェクトに導入して行くためには、従来の開発作業環境からこれらのCASEツールを導入した作業環境へ移行する気にさせるべく、導入の効果を明確にすることが重要である。しかし、ツールの効果の測定方法については統一的な尺度によるのではなくケースバイケースで評価されているのが現状である。ソフト工学の1つの課題として、共通的な取扱い方を明確にする必要があると考える。

この観点から、本稿では、デバグ支援ツールを対象として、デバグ支援ツールにとって最も重要なバグ検出力、バグ検出効率を簡便に評価する、復元バグ再捕獲による評価方法を提案する。

本稿では、まず、この復元バグ再捕獲によるツール評価の考え方を示す。次にこの方法により実際に行った評価実験の結果を示し、提案する方法がツールのバグ検出力の評価に有効であることを示す。さらに、今回の評価対象としたデバグ支援ツールによるデバグ効率の改善効果についても報告する。

2．ツール評価への要求条件

2．1 従来の問題点

共通のツール評価のガイドラインがないため、個別にケースバイケースで評価が行われている。

以下のような方法がとられているものと思われる。いずれも大規模実用化プロジェクトにおいては禁止的であり、できるだけ簡便に的確な評価結果がえられる方法が望まれる。

①試行錯誤的導入：ともかく導入して使ってみるアプローチである。なかなか答が出ない、えてして無計画に実施されがちであり導入の前後での比較評価ができなくなる、などの問題がある。

②新旧両方法による平行開発：使用条件の一致をはかるのが困難、工数が2倍かかる、両者のペースが合わず（特に実用化プロジェクトの場合）結果がなかなか出ない、などの問題がある。

2．2 評価の基本的考え方

ソフト開発のライフサイクルは、図1に示すように、設計からコーディングに至る前半のバグ作り込みフェーズと、コーディングからシステムテストに至る後半のバグ除去フェーズから成る。一方、各工程の支援ツールは、個別ツールの段階から統合化ツールの段階へと進展してきている[2]-[4]。すなわち、一連のツールがセットで、開発環境として提供されるようになってきている。

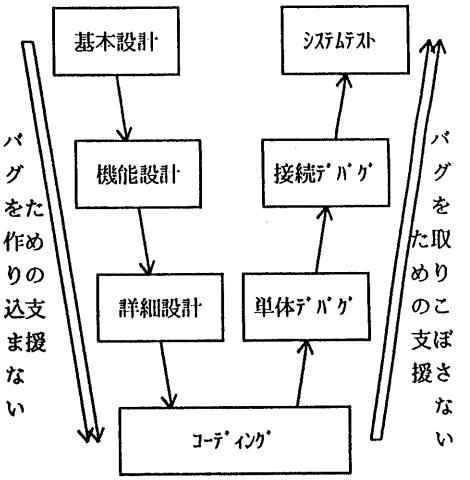


図1．ソフトウェアライフサイクルのモデル

ツールの評価も、個別・単体のツールを評価する段階から、統合化ツールセットとしての評価へ進むと考えられ、ツール間の連携の具合や使い易さ、機能の豊富さ、既存ツール環境からの移行性といった要素が大事になってくる。

しかし、基本的には、前半のフェーズに関してはバグを作り込まないこと、作り込ませないこと、作り込んでもみつけてくれること、等が重要であり、この視点で評価尺度を決める必要がある。後半のバグ除去、すなわちデバグフェーズに関しては、バグの取り逃がしのないこと（バグ検出力）や、できるだけ短時間にバグを検出し退治できること（バグ検出効率）、が基本的な評価尺度である。

以下では、単体デバグフェーズのツールについて、バグ検出力、バグ検出効率の観点での評価法を検討する。

3．復元バグ再捕獲法

3．1 基本概念

復元バグ再捕獲による評価法では、以前の環境におけるソフト開発で得られているデバグ作業記録（バグ及び工数）に基づいて、同じバグを最新のソースプログラム（以下、ソースと略称）中に復元・放流し、新しい環境で解決する。これにより新旧両環境のバグ検出力、バグ検出効率を比較評価するものである。

復元バグ再捕獲の概念を図2に示す。

今回の評価実験の対象としたのは単体デバグ工程である。単体デバグ工程ではフラグ消しが完了したプログラムモジュールについて、モジュール単位での論理デバグを行う。既存環境で見つかったバグが、新環境の単体デバグでどれだけ見つけられるかがポイントである。単体デバグ工程だけを実施すれば評価の答が出るのが特徴である。

3．2 バグ復元放流

復元バグ再捕獲法においては、デバグ開始直前（i.e.コーディング完了直後）のバグ含みのソースを復元する安定な方法が重要である。

次のような方法がある。確実なのは復元法1である。いずれの方法でも、既存環境における正確なバグ記録を入手することが前提である。なお、バグ復元作業はデバグ担当者以外の者が実施する。

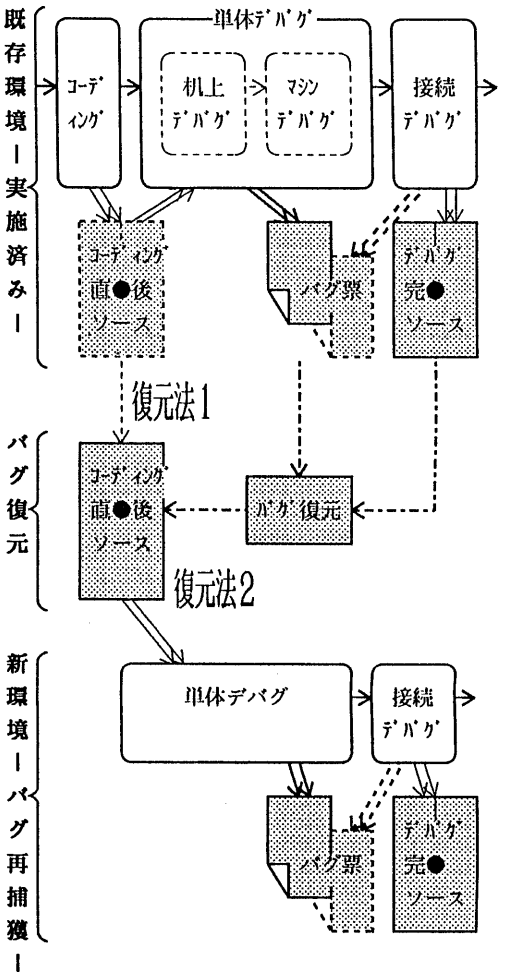


図2．復元バグ再捕獲の概念

・復元法1：コーディング直後のソースライブラリから取得する

・復元法2：バグ記録を参照して、最新のソースにバグを復元・埋め込む（但し、修正ミスによるバグは除く）

いずれも不可の場合、従来と同様、新旧2つの方法で平行開発、などの対策を取る。

復元作業をいつも実施するのは大変であるから、適当なモジュール群について、コーディング済みソース、バグ記録を、ツール評価用の標準テストベッドとして蓄えるようにするのが効果的である。

3.3 復元バグ再捕獲

復元バグ再捕獲は、新しいツール／デバグ環境によって、バグが復元され放流されたソースのデバグを行う（放流したバグを再び見つけ出す）過程である。

この復元バグ再捕獲の実施条件（原則）は以下の通りである。

- デバグ環境：新規環境
- テスト項目：対象モジュールに対し、比較対象の既存環境におけるデバグで実施したものと同じテスト項目であり、かつ全てのテスト項目を用意 → 品質保証レベルの観点で同一の目標を設定
- テスト実施手順：環境に合わせて作成 → 新旧デバグ環境の特徴がでることの保証
- テストデータ：環境に合わせて作成 → 新旧デバグ環境の特徴がでることの保証
- テスト実施者：1人（バグ復元者以外）

なお、テスト実施者のスキルレベルへの依存性を緩和するため、テスト項目、テスト実施手順はできるだけ細かく、具体的に指定しておく。

3.4 評価項目

(1) 評価項目

復元バグ再捕獲におけるバグ検出のモデルを図3に示す。

基本的な評価項目は次の通りである。

- バグ検出力
  - 捕獲力＝捕獲バグ数／全バグ数
  - 発見力＝新規バグ数／全バグ数
  - 見逃し率＝見逃しバグ数／全バグ数
- バグ検出効率＝捕獲バグ数／デバグ工数

(2) 測定項目、測定方法

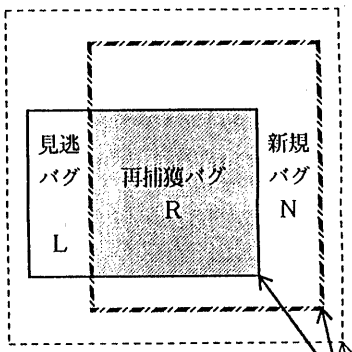
上記項目について評価値を求めるために測定項目と測定方法を次のように定めた。

- バグ件数：再捕獲バグも含め、デバグにより検出したバグの件数である。
- デバグ工数：以下の分類で測定する。

(a) 実施／確認：テスト項目の消化と結果の確認に要した時間

(b) 解析／修正：バグの解析と修正、および修正結果の確認のための時間

(c) その他：デバグ作業に必要な時間であるが上記(a)(b)いずれにも該当しない時間



- 復元バグ＝見逃しバグ＋再捕獲バグ
  ＝L＋R
- 捕獲バグ＝再捕獲バグ＋新規バグ
  ＝R＋N
- 全バグ ≒復元バグ＋捕獲バグ
  ＝L＋R＋N

図3．復元バグ再捕獲におけるバグ検出モデル

4. 評価実験

4.1 実験環境

今回の実験で比較評価の対象とした2つのソフト開発環境を図4に示す。評価項目は、バグ検出力、バグ検出効率の2つとした。

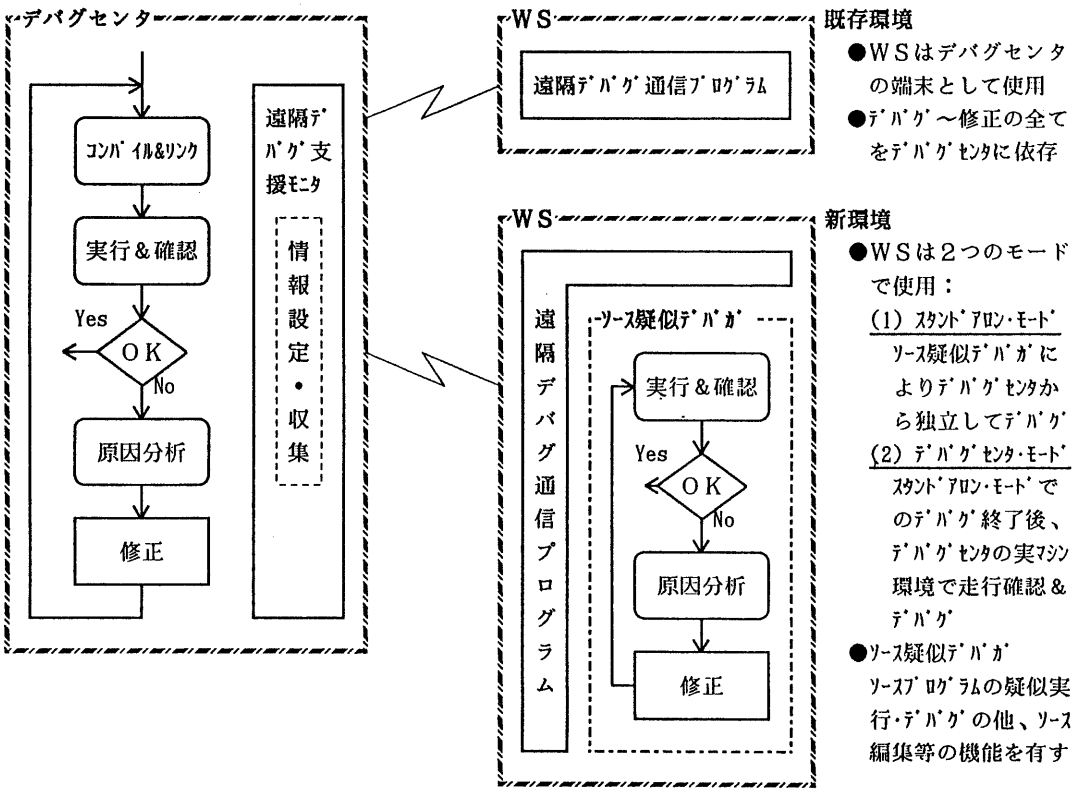


図4．実験環境とデバグ実施手順

バグ検出効率の評価のために必要なデータのうち既存環境でのデバグ工数に関するデータが保存されていなかったため、結果的には図4に示した新旧2つの環境で平行デバグすることにした。

4．2 デバグ対象プログラム － バグ復元

デバグ対象プログラムの概要を以下に示す。

- 分類：応用プログラム
- 規模：490step（PL/I likeの高級言語で記述）
- 構成：1モジュール構成
- 動作環境：汎用大型コンピュータOS上
  （デバグセンタと同じ）
- 機能概要：コンパイルリストファイルを入力し、ソース部とオブジェクト部をソースステートメント対応に並べてリスト出力する。

デバグ対象プログラムのソースは復元法2で復元した。復元バグの密度は約46件／Ksである（22件埋め込み）。

次項で述べるように、デバグ担当者は1人とした。したがって、別々の環境とは言え基本的に同じソースを1人でデバグすることになる。それにも拘らず学習効果などによりデバグ効率が変わるのを防ぐため、復元法2で復元したソースと等価的なバグを別の形で放流したソースをもう一本作成した。すなわち、2つのデバグ対象プログラムを作成し、各環境でそれぞれ1本をデバグすることにした。

このバグ復元作業は1人で行った。復元バグの特性をできるだけ同じものとし、デバグ対象プログラムに難易度の差が発生するのを防ぐためである。

4.3 デバグ － バグ再捕獲

図4に示した2つの環境で実施した。デバグ担当者は1人とした。

既存環境では、デバグセンタ上でデバグする（WSは端末として使う）。

新環境では、ソースの疑似実行方式によるデバガを使って、WS中心でデバグを実施する。WS上でのデバグ完了後、デバグセンタ上で実マシン依存のテスト項目に関するデバクや走行確認を行う。

- テスト項目：約1件／5stepのテスト項目密度で設定した。
- テストデータ：既存環境むけにはコンパイルリストファイルをそのまま利用し、新環境向けには当該ファイルの疑似ファイル（WS上にスタブを作成して実現）を作成した。

表1．復元バグと新環境での捕獲バグの内訳

復元バグ分類	捕獲バグ － 新環境			同既存環境	復元バグR
	WS	ﾏｼﾝ	計		
処理過不足	2	1	3	4	2
ﾛｼﾞｯｸ誤り	8	2	10	13	7
ｲﾝﾀﾌｪｰｽ誤り	2	1	3	3	3
設定値誤り	7	5	12	12	8
属性誤り	1	0	1	1	1
ﾏｸﾛ用法誤り	0	1	1	1	1
他用法誤り	0	0	0	0	0
その他	0	0	7	0	0
計	27	10	37	34	22
ｶｳﾝﾄ過剰件数（再掲）→				5	

4.4 実験結果と考察

4.4.1 実験結果

(1) バグ検出力

復元放流したバグと、既存環境で再捕獲したバグ、および新環境で再捕獲したバグを表1に示す。

捕獲バグ数が復元放流バグ数よりも非常に多くなっている。これはバグ復元放流以前から潜在していたバグが、今回のきめ細かなテスト項目の設定により摘出されたためである。

また、既存環境で捕獲された34件のうち5件は新環境では捕獲されていないものであるが、その原因は修正方法・内容の違いによる過剰カウントであった。内訳は、既存環境では修正ミスによりバグとカウントしたもの2件、既存環境ではカウントしたが、新環境では他のバグの修正と一緒に修正されてしまいカウントされなかったもの3件である。従って、実効的にはこれら5件は無効バグとして扱うのが適当である。

以上を考慮して、既存環境すなわちデバグセンタのマシンデバグで見つけたバグが実効的な復元バグ集合と考え、バグ復元・検出のモデル（実効）を図5のように補正する。バグ総数は37件（再捕獲バグ29件、新規バグ8件、見逃しバグ0件）である。

これより、新環境、i.e.”WSデバガ＋デバグセンタ”による分担デバグ方式による環境のバグ検出力の評価値は次のようになる（　）内は既存環境のバグ検出力の再評価値である）。

- 捕獲力　＝1．00　　(0.78)
- 発見力　＝0．22　　(0　)
- 見逃し率＝0　　(0.22)

新環境での再捕獲バグの内、WS上のソース擬似実行方式によるデバガでは取り逃がしたバグが7件あった。大部分はメモリアドレス制御やSVC命令といった実マシン依存の機能がデバガで擬似しきれない等の事情によるものであった。逆に新環境かつWS上のデ

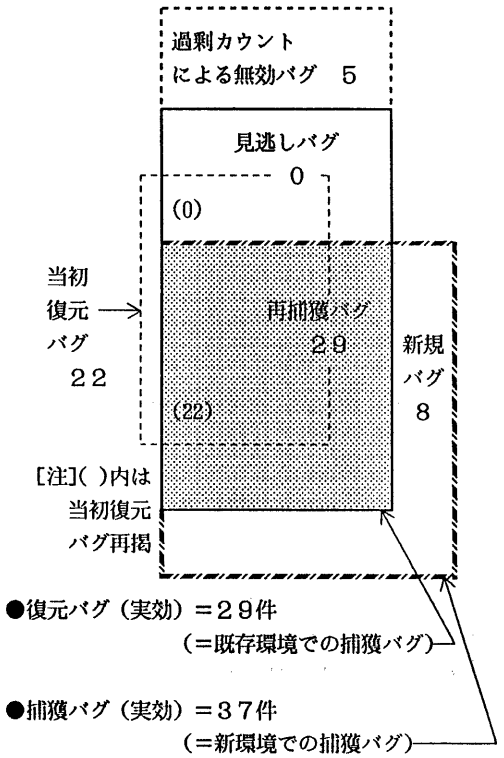


図5．新環境での復元バグ再捕獲の状況

バガでのみ見つけたものも8件あった。多くはデバガがサポートする言語仕様の制約に基づくものであった。ここではしかし言語仕様への適合性チェックに関するデバガの可能性に着目して新規捕獲バグとして扱うことにした。

(2) バグ検出効率

新旧環境でのデバグ工数の内訳を表2に示す。

デバグに要した時間は、既存環境677人・分、新環境491人・分と、新環境の方が27%少なくて済んだ。実施・確認工数が22%減、解析／修正工数が50%減であり、特に解析／修正工数の削減効果が大きい。この効率アップは、既存環境ではデバグセンタでのバッチ処理的形態でのデバグであるのに対し、新環境では捕獲バグの概ね8割弱をWS上のデバガのインタプリタ機能を使って即時的に解析・修正・結果確認できることによると考えている。

4.4.2 実験方式性能

新規に導入するデバグ支援ツールの性能（さらにはデバグ対象プログラムの種類など）にもよるが、評価方式から考えて、従来の試行錯誤的実施に対しては約1／2、新旧平行実施に比べると概ね1／4以下の工数で評価結果を得ることができると期待できる。

実際、試行錯誤的評価ではすべてを実施するソフト開発のライフサイクルのうち少なくとも前半が省略できる筈である。したがって、平均的な工数の工程配分比率[6]により、コーディング・単体デバグ工程までに70%以上の工数がかけられると仮定すれば、通常の開発で評価結果を出すのに比べ50%程度は少ない工数で済むと期待できる。

表2．デバグ工数の比較

環境	工数種別	工数内訳（単位：分・人）			
		実行・確認	分析・修正	その他	計
既存	ﾏｼﾝ作業	330	260	87	788
新規	WS作業	228	79	15	322
	ﾏｼﾝ作業	30	52	87	169
	計	258	131	102	491

5. おわりに

新規デバグ支援ツールの導入効果を簡便に予測する手段として、復元バグ再捕獲の考え方による評価方法を提案した。

1ポイントではあるが、評価実験の結果、①バグ検出力、②バグ検出効率に関し、非常に効率よく新旧環境の比較評価を行うことができた。実際、バグ検出力、バグ検出効率の比較データは新旧環境の特徴をよく反映したものとなっている。評価結果を出すのに必要な工数は従来の方法に比べ半分以下で済むなど、評価方法としての性能も満足なレベルが期待できる。

今後検討すべき課題としては、より一層効率的に評価実験を実施するための評価試験ベッドの確立や部分デバグによる効果推定、ソフト開発支援ツールの統合化の傾向から複数工程を対象とした効果の測定評価方法への拡張、機能の使いやすさなどの側面からの総合的評価法の確立、などがある。

[謝辞]

日頃ご指導頂くと共に、本稿の内容に関し討論して頂いたＮＴＴ情報通信処理研究所・情報処理研究部拝原部長、川原主幹研究員、ＮＴＴソフトウェア㈱豊嶋部長、石塚部長をはじめ、本実験プログラムを支援して頂いた皆さんに感謝致します。

[参考文献]

(1) 福山他："分散形ｿﾌﾄｳｪｱ開発環境の構成方式"，NTT R&D，Vol.39，No.7，1990
(2) 山本他："設計情報ﾘﾎﾟｼﾞﾄﾘを用いた統合化CASE"，NTT R&D，Vol.39，No.7，1990
(3) Lewis,G.R.："CASE Tool Integration"，IEEE Software，July，1990
(4) 藤野他："特集：ＣＡＳＥ環境"，情報処理，Vol.31，No.8，1990
(5) 国友著："ﾌﾟﾛｸﾞﾗﾑ開発管理"，ｵｰﾑ社，1990