プログラムにおけるバグ除去時間の分析

An Analysis of Time Needed to Remove Faults from Program Text

松本健一　楠本真二　井上克郎　菊野　亨　鳥居宏次
Kenichi Matsumoto　Shinji Kusumoto　Katsuro Inoue　Tohru Kikuno　Koji Torii

大阪大学　基礎工学部　情報工学科
Faculty of Engineering Science, Osaka University

あらまし　本稿では，学生実験から収集したプログラムテキストの変更情報をもとに，プログラム中にバグが作り込まれてから除去されるまでの時間(バグ除去時間)を測定し，その分析を行う．ここでは，バグをプログラムの行の単位で識別し，バグ除去時間をバグ検出時間とバグ修正時間の和として定義する．先ず，変更内容に関する分析より，修正時間の長いバグほど除去されるまでの変更回数が多いことが分かった．次に，個人データの分析より，検出時間は平均的には個人差がないが，修正時間は平均的に個人差があることが分かった．最後に，バグの種類の分析より，検出時間は平均的にバグの種類による差がないが，修正時間は平均的に差があることが分かった．

1. まえがき

ソフトウェアの信頼性の向上はソフトウェア工学における重要な課題であり，これまでにも多くの解決法が提案されてきている[3,6,11]．信頼性を測る1つの尺度としてバグ数を採用し，開発されたソフトウェア中に残っているバグ数を推定するモデルも報告されている[7]．更に，ソフトウェアの開発段階や利用段階で発見されたバグの原因や混入過程の分析を行い，その結果に基づいて開発環境を改善し，バグの再発を防止する試みが注目されている[1,2,8,9,11]．

しかし，バグが作り込まれてから，検出され，除去されるまでに要した時間（バグ除去時間）に関する研究はほとんど行われていない．プログラミングの効率を考える上で，バグの除去時間は基本的な要素と考えられ，これを定量的に分析することは重要である．

本稿では，バグ除去過程のモデル化を行い，学生実験から収集したプログラムテキストの変更情報にモデルを適用し，バグ除去時間の分析を行う．ここでは，プログラムテキストの行単位でバグを識別する．分析の都合上，変更された行はバグを含んでいたと仮定する．行の作成をその行へのバグの混入，行に対する最初の変更をバグの発見，最後の変更をバグの除去と考える．バグの混入から発見までの時間をバグ検出時間T_d，発見から除去までの時間をバグ修正時間T_cと呼び，T_d+T_cをバグ除去時間と定める．

本稿の構成は次の通りである．2.では，プログラム開発段階におけるバグの混入，発見，除去の一連の過程のモデル化を行う．次に3.では，本学科の学生実験からツールを用いて自動的に収集したプログラムテキストの変更情報を利用したバグ除去時間の測定，分析について述べる．4.ではプログラムテキストの各行毎に変更内容(変更時刻と変更回数)の分析を行う．引き続き，5.と6.では変更回数が4回以上の行について手作業による測定，分析を行う．5.では各プログラマ毎の個人差について，6.ではバグの種類別の差異について考察する．最後に7.では，測定実験を通じて明らかになった問題点とその改善策について述べる．

2. バグ除去過程のモデル化

2.1 バグ除去時間

ここでは，図1に示す様に，バグの除去過程をバグの混入，発見，除去の3つの事象に分けて考える[4,5]．プログラムを計算機上で実行した時，予想される結果が求まらない(failureが発生した)とき，バグが

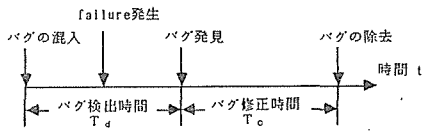


図1．バグの除去過程

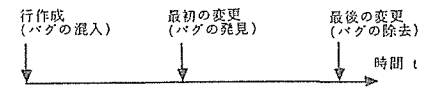


図2　行の変更とバグ除去の対応

混入していたことが分かる．プログラマはそのバグの原因を探し，それを修正する．この一連の操作を何回か繰り返すことによって，最終的にバグを除去することができる．

図1に対応して，バグの混入から発見までの時間をバグ検出時間Td，バグの発見から除去までの時間をバグ修正時間Tcと表わし，Td＋Tcをバグ除去時間と定義する．

2.2　テキストの変更情報[4]

データ収集ツールを用いて，各プログラマがプログラム作成過程で行った全てのプログラムテキストの変更作業を差分リストの形で蓄積する．具体的には，ＵＮＩＸのdiffコマンドを利用する．

今回の実験では，5分おきにプログラムテキストを調べ，テキストが変更されていればdiffコマンドを実行する．また，このコマンドが実行された時間も同時に記録している．差分リストには，追加(append)された行，変更(change)された行，消去(delete)された行が保持される．このリストを見れば，各プログラマのプログラムテキストの変更過程が全て分かる．

2.3　測定に利用したモデル

プログラムの作成はテキストの追加(append)，変更(change)，及び，削除(delete)の繰り返しによって行われる．これをプログラムの行について考えてみると，行の作成(append)，最初の変更(change)，…，最後の変更(change)という履歴をたどる．一方，バグを作り込まない理想的なプログラミングにおいては，全ての行がappendされるだけで作成が終了する．

従って，1回以上変更されている各行に対して，appendの実行をバグの混入，最初のchangeをバグの発見，最後のchangeをバグの除去に対応させることにする(図2参照)．

3．　測定実験

3.1　測定実験の概要

データは大阪大学基礎工学部情報工学科3年生に対し，昭和62年10月から昭和63年2月の間に行われた学生実験より収集した．データの収集方法(詳細は3.3で述べる)，及び，分析方法に基づき，測定実験は次の3つに分類される．

(1)実験1(変更内容の分析)･･･ツールを用いて自動的にプログラムテキストの各行が何回変更されているかを測定する．分析は変更回数と変更時間(バグ検出時間T_dとバグ修正時間T_c)に関して行う．

(2)実験2(個人差の分析)･･･変更回数が4回以上の全ての行に対し，実際に手作業で再調査して，バグ数，T_d，T_cを測定し直す．分析は，バグ数とT_d，バグ数とT_cのそれぞれの関係に個人差が生じているか否かについて行う．

(3)実験3(バグ種類別の分析)･･･実験2で手作業で求めたデータに対し，更に詳細な分析を行う．バグを含んでいる文を分類し，その分類項目ごとにT_d，T_cの値を評価する．

3.2　データ収集源

学生実験ではプログラミング言語PASCALの部分集合に対し，Ｃ言語を用いてコンパイラとアセンブラを作成する．プログラムの大きさはどちらも約1000行になる．学生は2人で1つの班を作り，それぞれ，コンパイラとアセンブラのいずれかを作成する．中間言語の仕様は各班で自由に決める．各プログラムの機能は各学生の判断により仕様を拡張することが許されている．このため問題の難しさは学生によって多少異なる．

全ての学生はＣ言語の使用経験がある．各学生は事前にコンパイラ，アセンブラの講義[10]を受けているが，実際に作成するのは初めてである．被験者となった学生は24人であったが，信頼性を欠くデータは削除した．その結果，5人の学生{Ａ，Ｂ，Ｃ，Ｄ，Ｅで参照する}のデータが最終的に利用可能となった．5人の内訳は，1人{学生Ｂ}がコンパイラを，他の4人{学生Ａ，Ｃ，Ｄ，Ｅ}がアセンブラの作成である．また，1組の班{学生ＢとＣ}が含まれている．

3.3 収集方法

プログラム中の各行の変更情報をとるため，ここでは各行が属するテキストに注目する．行が変更される度に，その行が属するテキストに対する何回目のeditであるかを調べ，その回数を記録する．2.2と同様に，各行について最初のappendが行われたeditをバグの混入，最初のchangeが行われたeditをバグの発見と考える（厳密には，バグの発見とバグの場所の特定には時間的に差があるが，テキストの変更情報だけからはバグ発見の正確な時刻を知ることはできない）．最後の changeが行われたeditをその行に関するバグの除去と見なす．

この履歴をとる時に注意すべきこととして，1回のeditの実行で複数の行が1度に変更され，変更の前後で行数に変化が生じる場合がある．今回の測定では，変更された行の履歴の和をとって1つにまとめ，それを変更後の各行の履歴と定めた．

次に，バグ除去時間を求めるため，editの回数と時間の対応づけを次の様に定めた．各学生の端末使用時間の累積データと照合することにより，editした時刻（カレンダータイム）を端末使用時間に変換する．これにより，i回目のeditとj回目（i＜j）のeditの時間差が求まり，バグ検出，修正時間も求まる．しかし，実際には端末を使っていない間にもバグとりをする．そこで，ログアウトして次にログインするまでに日付が変わった場合には，ある一定時間（今回は60分）を端末使用時間に加えることにした．

表1　変更回数と行数

変更回数	A	B	C	D	E	合計
0	187	2,119	799	603	399	4,107
1	193	353	202	115	129	992
2	51	88	140	33	204	516
3	60	26	56	26	74	242
4	24	27	21	10	30	112
5	23	9	15	6	8	61
6	8	5	10	3	7	33
7	5	7	4	4	15	35
8	14	23	7	1	11	56
9	3	1	1	0	2	7
10	0	2	2	0	6	10
11	1	0	4	0	2	7
12	0	0	1	0	0	1
13	11	0	1	1	0	13
14	0	0	0	0	0	0
15	0	0	0	0	0	0
16	0	0	0	1	0	1
17	0	0	0	0	0	0
18	0	0	0	4	0	4
19	0	0	0	7	0	7
20	0	0	0	5	0	5
21	0	0	0	1	0	1
22	0	0	0	2	0	2
行数の合計	580	2,660	1,263	822	887	6,212

4. 実験1（変更内容の分析）

4.1 収集データ

5人の学生{A，B，C，D，E}のそれぞれに対し，プログラムテキストの各行の変更回数をツールを用いて測定した．変更回数と行数の関係を表1に示す．

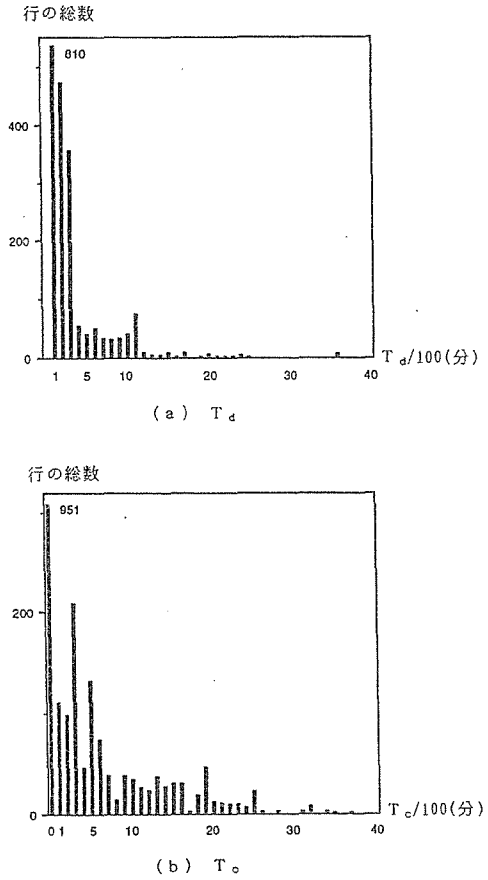


図3　行数とバグ除去時間の関係

4.2　分析

表１より変更回数が0である行数の割合がかなり高い．Aが32.2%，Bが79.7%，Cが63.3%，Dが73.4%，Eが45.0%となっている．更に，変更回数が3回以下までの行とすると実に94.3%にもなる．

表１中の変更回数を，3.3の方法に従って時間に変換し，バグ検出時間T_dと修正時間T_cを計算する．A～Eの5人の学生に関する合計をとって，行数とT_d，行数とT_cのそれぞれの関係をまとめたものを図３に示す．図３(a)より$1 \leqq T_d/100 \leqq 5$の間では行数が単調減少の傾向がある．図３(b)より，$T_c/100=0$の行数が全体の約45.7%を占めている．

次に，変更回数の多い行の詳細な分析のため，変更回数が2，3，及び，4以上の3つに分けて，T_cとT_dの関係を図示した(図４参照)．図４より，変更回数の多い行ではT_dの値よりもT_cの値の方が大きいことが分かる．

5．　実験２(個人差の分析)

5.1　収集データ

4.1ではツールを用いて各行当たりの変更回数を求めた．一般に変更回数が多い行ほど修正時間T_cが長くかかるため，その行に含まれるバグはプログラム効率を著しく阻害すると考えられる．

そこで，本実験，及び，実験３では変更が4回以上の全ての行について，実際に手作業でバグの数，T_d，T_cを求めた．ツールによる集計ではバグと見なされていた行で，実際にはバグでなかったものがかなり存在していた．その評価データを表２に示す．

次に，本当のバグに対して，各プログラマ毎に集計したバグ全てについてのT_dとT_cの合計時間を表３に示す．なお，同表中の[ ]で表わす数字はT_d，T_c T_d+T_cの平均値(バグ1個当たりの時間)を示す．

5.2　分析

各学生に対し，バグ数とT_dの合計，及び，バグ数とT_cの合計のそれぞれの相関を調べた(図５参照)．図５(a)中の実線は(T_dの合計)の(バグ数)への回帰直線で，点線は(バグ数)の(T_dの合計)への回帰直線を表わす．図５(b)についても同様である．

図５(a)より，バグ数とT_dの合計との間の相関係数は0.96と非常に高くなった．従って，5人の学生A，B，C，D，Eのデータは全て直線上に位置しており，T_dの平均時間(直線の傾き)に個人差がないことが分かる．一方，T_dの合計時間そのものには明らかに個人差がある．

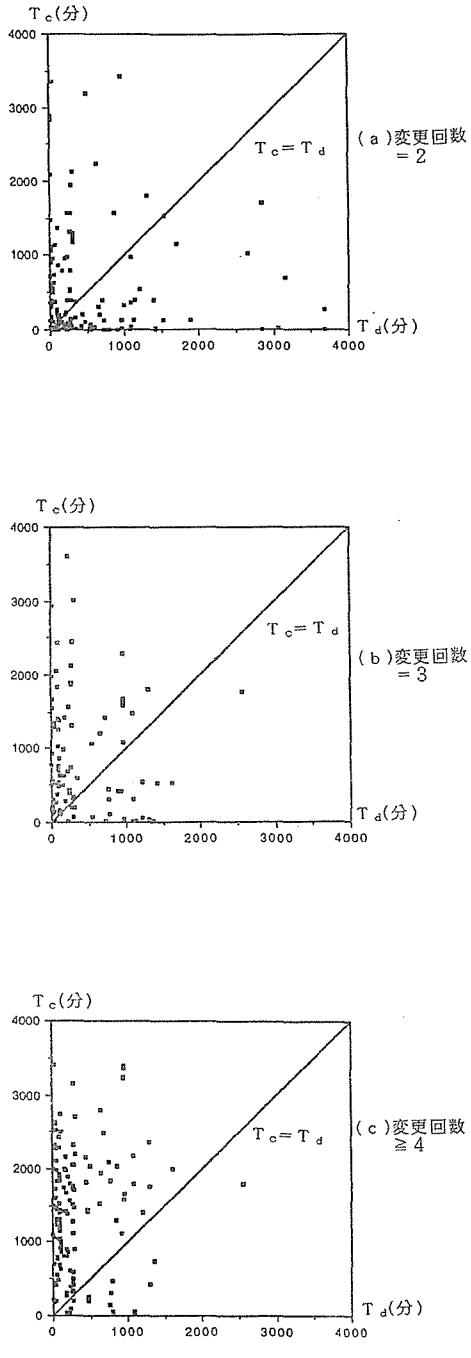


図４　T_cとT_dの関係

表2　集計方法によるバグ数の差

	ツールによるバグ数	手作業によるバグ数	差
A	89	63	26
B	74	6	68
C	66	31	35
D	45	25	20
E	81	22	59
計	355	147	208

表3　バグ除去時間の評価

	バグの個数	T_dの合計 [平均]	T_cの合計 [平均]	T_d+T_cの合計 [平均]
A	63	32394 [514]	34046 [540]	66440 [1055]
B	6	2888 [481]	3628 [605]	6516 [1086]
C	31	13532 [437]	59191 [1909]	72723 [2346]
D	25	4664 [187]	15973 [639]	20637 [825]
E	22	11041 [502]	7696 [350]	18737 [851]

表4　文によるバグの分類

	宣言文	代入文	制御文	関数呼び
A	8	23	21	11
B	0	1	4	1
C	5	10	5	11
D	6	6	9	4
E	12	3	6	1
バグの合計	31	43	45	28

表5　文ごとのバグ除去時間

文の種類	個数	T_dの平均値	T_c の平均値
宣言文	31	504	751
複雑	28	444	1246
単純	15	404	535
代入文	43	430	998
if	30	372	422
while	12	488	719
その他	3	546	330
制御文	45	412	508
関数呼び	28	423	1126

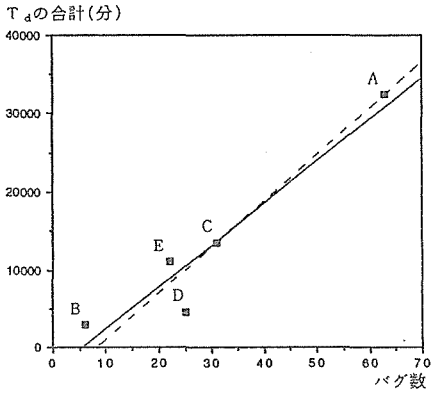
図5(b)より，バグ数とT_cの合計との間の相関係数は0.54と低かった．従って，5人の学生A，B，C，D，Eのデータは1つの直線上には位置しておらず，T_cの平均時間に個人差があることが分かる．また，T_cの合計時間そのものにも明らかに個人差がある．

一方，表3の[ ]で記した平均時間(バグ1個当たりに要した時間)で見ると，T_d，T_c，T_d+T_cのいずれも個人差がある．その中でも，T_cは最大値と最小値の比が5.5倍にもなっており，著しい個人差を引き起こしている．

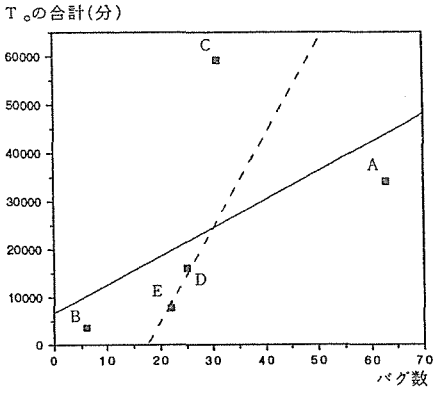
6.　実験3(バグ種類別の分析)

6.1　収集データ

5.1で求めた各バグに対して，バグが混入している



(a)　バグ数とT_d



(b)　バグ数とT_c

図5　バグ数とバグ除去時間の相関

文の種類を詳細に調べた．ここでは，基本的に4種類の文{宣言文，代入文，制御呼び}に分けて集計した．各学生ごとの分類結果を表４に示す．

次に，代入文を単純な代入文(例えば x←yの様に，左辺，右辺ともに単なる変数である代入文)と複雑な代入文（例えば A[x,y]←B[x,y]＋pの様に，左辺，あるいは右辺の一部に配列等を表わす変数が含まれている代入文)に分け，制御文をif文，while文とその他に分けた．その分類の下に，バグ検出時間T_dの平均値，バグ修正時間T_cの平均値を計算した(表５参照)．

6.2 分析

表４より，バグとなっている文の種類別に見ると，制御文と代入文に関するものが多かった．但し，個人的にはかなりの違いが見られる．Ａが代入文と制御文に，Ｂが制御文に，Ｃが代入文と関数呼びに，Ｄが制御文に，Ｅが宣言文に，多くのバグを混入している．

表５より，バグ1個当たりのT_dは文(バグ)の種類別に見てもほとんど差が見られない．一方，T_cの平均値は文の種類によってかなりの差が見られる．特に，複雑な代入文と関数呼びにバグが混入すると，バグの修正が著しく困難になることが分かる．しかし，バグの総数で最大となっている制御文のT_cの値が最小である．

以上より，バグの総数だけではプログラム作成の時間的な効率についての十分な議論はできないことが明らかとなった．

7. むすび

本稿では，バグ除去過程のモデルを提案し，プログラムテキストの変更情報から見たバグ除去過程にモデルの適用を行なった．先ず，バグ検出時間T_dには個人差がなく，バグの種類別による違いもほとんど見られないという結果を得た．

一方，バグ修正時間T_cについては，個人差がかなり存在し，バグの種類による違いも見られることが分かった．T_cに関する結果に関しては個々のプログラマの能力差[5]が主な要因であると考えられる．

今後の課題について簡単にまとめる．今回の分析では，実験終了時にプログラムテキストとして最終的に残った行をその対象としている．実際には，作成の過程で消去された行も存在する．また，手作業で測定した行は変更回数が4回以上の行だけに限っており，変更回数が3回以下の行は対象としていない．これらの行を含めた分析を行なうことは重要な課題である．

テキストの変更情報の収集についても検討すべき問題点がある．現在，バグ除去時間を正確に測定することができない場合への対処法について検討している．具体的には，段付けに伴う変更，注釈行の挿入，変数名の変更，等の取り扱いである．

文献

[1] V.R.Basili and H.D.Rombach : "Tailoring the software process to project goals and environments", Proc. of 9th ICSE, pp.345-357 (1987).

[2] A.Endres : "An analysis of errors and their causes in system programs", IEEE Trans. Software Engineering, SE-1, 6, pp.140-149 (1975).

[3] K.Kishida et al. : "Quality - assurance technology in Japan", IEEE Software, 4, 5, pp.11-18(1987).

[4] 楠本真二：“プログラムテキストの変更情報から見たバグとその除去時間の分析”，大阪大学基礎工学部情報工学科特別研究報告(1987).

[5] K.Matsumoto et al. : "Error life span and programmer performance", Proc. of 11th COMPSAC, pp.259-265(1987).

[6] 宮本勲：“ソフトウェア・エンジニアリング：現状と展望”，TBS出版会(1982)

[7] M.Ohba : "Software reliability analysis models", IBM J. Res. Develop., 28, 4, pp.428-443(1984).

[8] D.M.Weiss : "Evaluating software developmen by error analysis : The data from the architecture research facility", Journal of System and Software, 1, pp.57-70(1979).

[9] D.M.Weiss and V.R.Basili : "Evaluating software development by analysis of changes : Some data from the software engineering laboratory", IEEE Trans. Software Engineering, SE-11, 2, pp.157-168(1985).

[10] N.Wirth : "Algorithm + Data Structure = Programs", Prentice-Hall(1976).

[11] 保田勝通：“テスト・品質保証技術の現状と課題”，情報処理，28.7, pp.873-879(1987).