論　文

フォールト位置特定におけるプログラムスライスの実験的評価

西松　顯[†]　西江　圭介[†]　楠本　真二[†]　井上　克郎[†,††]

An Experimental Evaluation of Program Slicing on Fault Localization Process

Akira NISHIMATSU[†], Keisuke NISHIE[†], Shinji KUSUMOTO[†], and Katsuro INOUE[†,††]

あらまし　プログラムのデバッグや理解を支援するための手法としてプログラムスライス（スライス）が提案されてきている．しかし，実際のデバッグを対象としたスライスの評価はほとんど行われていない．本論文では，スライスがフォールト位置特定に有効であるかどうかを評価するために行った二つの実験（それぞれ評価実験1，評価実験2と呼ぶ）についてまとめる．二つの実験では，被験者を二つのグループに分け，一方のグループはスライスを用いてフォールト位置特定を行い，もう一方のグループはスライスを用いずに行う．評価実験1では6人の被験者が，数百行のプログラムに対して計算機上でフォールト位置特定を行う．評価実験2では被験者を34人に増やし，6種類の数十行のプログラムに対してフォールト位置特定を行う．実験の結果，スライスを用いてフォールト位置特定を行った方が，スライスを用いずにフォールト位置特定を行った場合より効率良くフォールト位置特定が行えることが確認できた．また，フォールトの種類によってスライスの効果が異なることが確認された．

キーワード　プログラムスライス，デバッグ，フォールト位置特定，実験的評価

1. まえがき

ソフトウェアシステムの大規模化，複雑化に伴いソフトウェア開発における生産性，及び，品質向上の実現はソフトウェア工学における研究の主要な目標に位置づけられてきている．ソフトウェアの品質や生産性を向上させるためには，開発されたソフトウェアプロダクトだけでなく，その開発プロセスを対象として作業の改善を行うことが必要である．

一方，現実のソフトウェアプロジェクトではソフトウェア開発コストの50〜80%をテスト工程に費やしているという報告がある．したがって，ソフトウェア開発プロセスの改善を行うためには，テスト工程の改善を行うのが効果的である．テスト工程は故障の検出（テスト）と故障の原因であるフォールトの修正（デバッグ）の二つの作業から構成される．一般に，フォールト位置の特定がデバッグにおいて最も時間がかかる作業であるといわれており[5], [6]，フォールトの位置を効率良く特定する方法の開発が重要となっている．

フォールトの位置を効率良く特定するための方法の一つとして，プログラムスライス技法（Program Slicing．以降，単にスライスと呼ぶ）を利用した手法が提案されてきている[1]．スライス技法はプログラム内のある文の実行に影響を与えるすべての文を抽出する技術であり，抽出された文の集合をスライスと呼ぶ[14], [15]．スライス技法を利用するデバッグでは値の誤っている変数に対して，全プログラムにわたってスライスを求め，そのスライスの中でフォールトとなっている文を探し出す．文献[13]においてHorwitzらは，9種類のCのプログラム（732〜28177行）に対してスライスを計算し，得られたスライスの平均サイズはもとのプログラムの約56%になったことを報告している．また文献[2]において，AtkinsonとGriswoldは100万行のプログラムに対してスライスを計算し，得られたスライスの平均サイズはもとのプログラムの1〜8%になったことを報告している．プログラム全体ではなくスライスとして抽出された文のみを対象とすることで参照範囲が少なくなり，効率良くフォールト位置特定が行えるといわれているが，実際のデバッグ作業を対象としたスライスの有効性の評価はほとんど

[†] 大阪大学大学院基礎工学研究科，豊中市
Graduate School of Engineering Science, Osaka University, Toyonaka-shi, 560–8531 Japan
[††] 奈良先端科学技術大学院大学情報科学研究科，生駒市
Graduate School of Engineering Science, Nara Institute of Science and Technology, 8916–5 Takayama-cho, Ikoma-shi, 630–0101 Japan

行われていない．

本論文では，スライスが実際のプログラムのデバッグ作業（フォールトの位置特定）に有効であるかどうかを実験的に評価した（本論文では 2 回の評価実験を行い，それぞれ評価実験 1，評価実験 2 と呼ぶ）．具体的には，評価実験 1 では 6 人の被験者を二つのグループ GP1 と GP2 に分け，まず 9 個のフォールトを含んでいるプログラムに対して，GP1 に含まれる被験者はスライスを用いてデバッグを行い，GP2 に含まれる被験者はスライスを用いずにデバッグを行う．次に，GP1 に含まれる被験者はスライスを用いずにデバッグを行い，GP2 に含まれる被験者はスライスを用いてデバッグを行う．最後に，GP1 と GP2 の間でデバッグに要した時間についての比較を行う．評価実験 2 では被験者の数を十分に多くし，被験者 34 人をグループ G1 と G2 に分け，G1 に含まれる被験者には，スライス情報を含まないプログラムリストのフォールト位置を，G2 に含まれる被験者には，スライス情報を含むプログラムリストのフォールト位置を特定してもらい，それに要した時間について G1，G2 間で比較を行った．実験の結果，スライスを用いてフォールト位置特定を行った方が，スライスを用いずにフォールト位置特定を行った場合より効率良くフォールト位置特定が行えることが確認できた．

なお，文献 [17] において，Zelkowitz と Wallace はソフトウェア工学の分野で提案された手法やツールの評価手法についてまとめている．彼らは三つの手法 (1) Observational methods，(2) Historical methods，(3) Controlled methods について述べている．本研究のように大学環境で実験を実施する場合には，(3) Controlled methods を用いることが一般的であることを指摘している．本研究でも，この指摘に従い，Controlled methods を用いている．また，データの収集・評価方法については GQM パラダイム [3] を用いている．

以降，**2.** ではスライスについて述べる．**3.**，**4.** では，それぞれ評価実験 1 と評価実験 2，その結果についてに述べる．最後に，**5.** でまとめと今後の課題について述べる．

2. スライス

スライス技法はプログラム内のある文の実行に影響を与えるすべての文を抽出する技術であり，抽出された文の集合をスライスと呼ぶ [14], [15]．スライスはデバッグを目的とし提案されたが，現在ではプログラム理解，再利用可能なコードの抽出などを目的としたさまざまなスライス [10] が提案されている．本実験で対象とするスライスは Weiser らにより提案された静的スライス（static slice）である．以降，単にスライスと記述する場合には静的スライスを意味するものとする．

2.1 スライシング

これまでに我々は文献 [12] においてスライス抽出アルゴリズムを提案している．このアルゴリズムでは，プログラムの依存関係解析の結果得られるプログラム依存グラフ（Program Dependence Graph，略して PDG）からスライスを抽出する．PDG の節点はプログラム中の各文及び if 文や while 文の条件判定部分を表し，辺は変数の影響を伝えるデータ依存（Data De-

```
program euclid(input,output);
var x,y,g,l:integer;
function gcd(m,n:integer):integer;
forward;
procedure swap(var a,b:integer);
var    temp:integer;
begin
   temp:=a;
   a:=b;
   b:=temp;
end;
function lcm(a,b:integer):integer;
var    c:integer;
begin
   c:=gcd(a,b);
   lcm:=(a div c)*(b div c)*c
end;
function gcd;
var    w:integer;
begin
   if m＜n then begin
      swap(m,n);
   end;
   while n＜＞0 do begin
      w:=m mod n;
      m:=n;
      n:=w;
   end;
   gcd:=m;
end;
begin
   writeln('Input x and y');
   readln(x,y);
   writeln('x=',x,' y=',y);
   g:=gcd(x,y);
   l:=lcm(x,y);
   writeln('gcd=',g);
   writeln('lcm=',l);
end.
```

図 1　PDG のもとのプログラム
Fig. 1　Sample program.

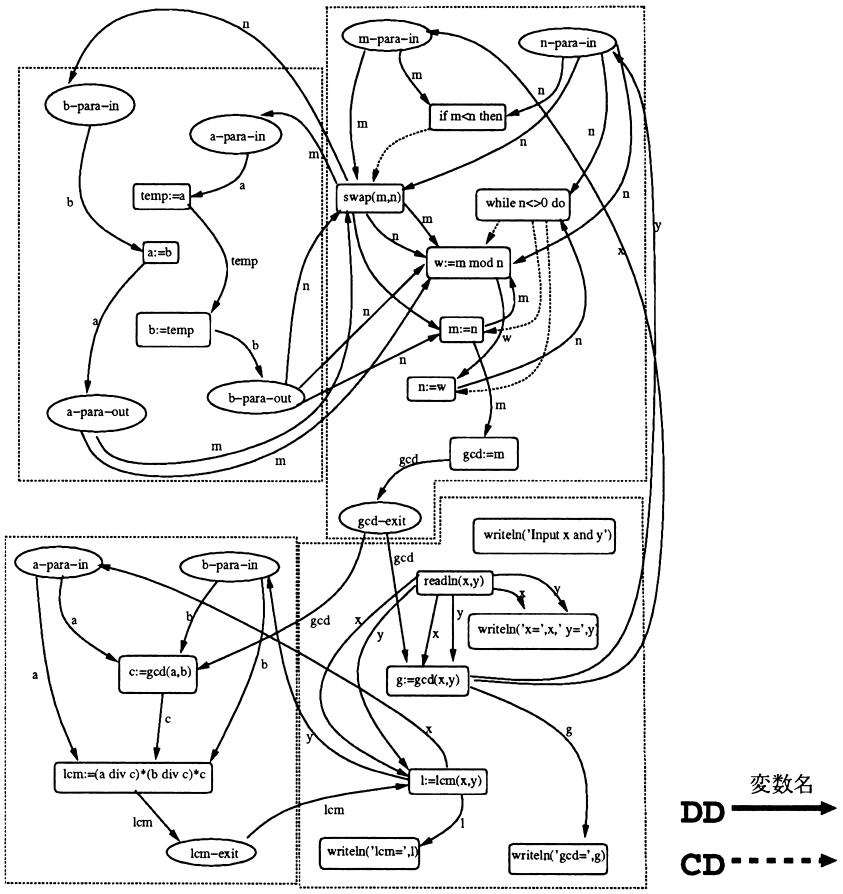

図 2　PDG の例
Fig. 2　Example of PDG.

pendence，略して DD）関係及び条件文や繰返し文の制御の影響を伝える制御依存（Control Dependence，略して CD）関係を表す．図 1 に示すプログラムの PDG を図 2 に示す．

文 s における変数 v に関するスライスとは，PDG 上において，CD 関係の辺または DD 関係の辺をたどって文 s の変数 v に到達できる節点集合に対応する文の集合である [12]．特に，PDG を与えられた節点から辺の順方向にたどって得られた集合を forward スライスと呼び，逆方向にたどって得られた集合を backward スライスと呼ぶ．スライスの例を図 1 に示す．このプログラムの 37 行目で参照されている変数 g に関する backward スライスが，図中の下線部分に示されている．この場合，変数 g に影響を及ぼさない部分，すなわち関数 `lcm` は backward スライスに含まれていない．

2.2　スライシングツール

これまでに我々は，文献 [12] のスライス抽出アルゴリズムを利用したスライシングツールを作成している [8]．**3.** で述べる評価実験 1 では，このスライシングツールを利用して被験者はデバッグを行う．スライシングツールの対象言語は Pascal のサブセット(注 1)である．

また，スライシングツールは，スライス抽出機能だけでなく，以下の機能をもっている．

（機能 1）：プログラムの編集，コンパイル，実行．

（機能 2）：デバッガ

プログラムの連続実行，ステップ実行，変数の参照，ブレークポイントの設定．

(注 1)：変数はスカラ型のみで，条件文（if），代入文，繰返し文（while），入力文（read），出力文（write），手続き呼出し文，複合文（begin-end）からなる．

表 1　評価実験 1 概要
Table 1　Overview of the 1st experiment.

	GP1（3 人）	GP2（3 人）
Exp1	P1.1〜P1.8	
	（スライス利用不可）	（スライス利用）
Exp2	P2.1〜P2.9	
	（スライス利用）	（スライス利用不可）

（機能 3）：スライス

スライスを計算し抽出する機能．

したがって，**3.** で述べる評価実験を行う上では十分な機能をもっている．

3. 評価実験 1

3.1 実験概要

実験の目的はスライスのフォールト位置特定に対する有効性を確認することである．実験の概要を表 1 に示す．具体的には，酒屋問題 [16] に対するプログラムを 2 種類用意し（それぞれ P1，P2 とし，フォールトは含まれない），P1 に 8 個の，P2 に 9 個のフォールトを含めたプログラム（それぞれ P1.1〜P1.8，P2.1〜P2.9 とする）を用意する．実験は大阪大学基礎工学部情報科学科の学生 6 人に対して行った．まず，6 人の被験者を二つのグループ GP1 と GP2 にそれぞれ 3 人ずつ分ける．GP1 に含まれる被験者を A1，A2，A3，GP2 に含まれる被験者を B1，B2，B3 と呼ぶ．被験者は 1 人 1 台のワークステーション（**2.2** で示したスライシングツールが使用できる）と，テストに用いるテストデータが与えられる．まず，GP1 の被験者はスライシングツールの機能 1〜3 を利用して，GP2 の被験者はスライシングツールの機能 1，2 を利用して（つまり，スライス抽出機能を用いずに）P1.1〜P1.8 のフォールト位置特定を行う（これを Exp1 とする）．次に，GP2 の被験者はスライシングツールの機能 1〜3 を利用して，GP1 の被験者はスライシングツールの機能 1，2 を利用して P2.1〜P2.9 のフォールト位置特定を行う（これを Exp2 とする）．

なお，6 人の被験者は学部 3 年生のときの演習で，スライシングツールの対象言語である Pascal のサブセットに対するコンパイラを開発しており，言語に対する知識は十分にもっている．また，各グループに対してスライシングツールの中で使用する機能についての講習を事前に行った．

3.2 対象プログラム

実験ではいわゆる酒屋問題 [16] に対する 2 種類のプログラムを用いた．2 種類のプログラム（P1，P2）は，アルゴリズム，データ構造が異なるため，別のプログラムであると考えることができる．P1 に対して 8 個，P2 に対して 9 個のフォールトを作り込んだ．以下に，P1 に含めたフォールトの例を示す．

(F1.1)　出力処理の不足．
(F1.2)　変数の代入誤り．
(F1.3)　条件文の誤り．
(F1.4)　配列の初期化もれ．
(F1.5)　関数処理（関数呼出し文）の記述もれ．
(F1.6)　データ更新の誤り．
(F1.7)　手続きのパラメータわたしの誤り．
(F1.8)　関数の実行位置の誤り．

ここで，これら (F1.1)〜(F1.8) のフォールトを用いて，Exp1 で使用する 8 種類のプログラム P1.i ($i = 1, 2, \ldots, 8$) を作成した．P1.i には，(F1.i)〜(F1.8) のフォールトが含まれている（例えば，P1.1 には (F1.1)〜(F1.8) が，P1.5 には (F1.5)〜(F1.8) が，それぞれ含まれている）．また，これらのフォールトはプログラムの基本的な機能に対して作り込まれており，番号の小さいものから順に検出できるようなテストデータを 8 種類用意した（$Testdata_1$〜$Testdata_8$）．なお，フォールト位置の特定時間を正確に計測するために，一つのテストデータで発見されるフォールトは一意に決まっており，その他のフォールトは被験者に発見されないようにマスクされている．

Exp2 で使用する P2.1〜P2.9 も上述と同様に P2 に 9 個のフォールトを含め（それぞれ F2.1〜F2.9 とする），作成した．P1 に含めた 8 個のフォールトと P2 に含めた 9 個のフォールトとは関連性がない．

3.3 実験プロセス

実験の手順は次のとおりである．

Step0: $i = 1$ とする．

Step1: $Testdata_i$ を用いて，プログラム P1.i のフォールト位置の特定を行う．

Step2: 特定したフォールトとその位置を実験監督者に申告する．正しい場合は Step3 へ．間違った場合には，Step1 へ戻る．

Step3: $i == 8$ の場合実験終了．$i < 8$ の場合，$i = i + 1$ として Step1 へ戻る．

プログラム P1.i において，$Testdata_i$ を用いて発見できるフォールトは F1.i のみである．これを被験者が発見した後に与えられるプログラム P1.$i+1$ はフォールト F1.i が既に修正されており，プログラム P1.i と

表 2　Exp1 データ（単位：分）
Table 2　Data of Exp 1. (min)

	スライス利用			スライスなし		
	A1	A2	A3	B1	B2	B3
F1.1	31	26	17	17	28	23
F1.2	8	10	20	10	18	12
F1.3	15	15	13	26	36	28
F1.4	25	20	22	27	17	32
F1.5	14	26	18	35	25	41
F1.6	15	10	10	17	23	17
F1.7	4	12	8	7	16	7
F1.8	7	9	12	15	12	6
合計	119	128	120	154	175	120

表 3　Exp2 データ（単位：分）
Table 3　Data of Exp 2. (min)

	スライスなし			スライス利用		
	A1	A2	A3	B1	B2	B3
F2.1	17	17	36	18	11	14
F2.2	6	5	24	6	8	14
F2.3	12	24	12	27	10	19
F2.4	30	13	41	18	16	36
F2.5	6	18	8	20	16	10
F2.6	11	16	15	20	13	5
F2.7	5	19	5	8	7	17
F2.8	5	5	4	2	7	1
F2.9	26	9	10	12	5	2
合計	118	126	155	131	92	118

異なっている部分はその修正部分のみである．上記は Exp1 の手順を示したものであるが，Exp2 においても同様である．

3.4 実験結果

Exp1，Exp2 における各フォールトの位置特定に要した時間（単位：分）に関するデータを表 2，表 3 に示す．

3.4.1 Exp1

Exp1 では，スライスを利用しなかった GP2 の被験者のフォールト位置特定に要した平均時間は 165 分（B1: 154 分，B2: 175 分，B3: 120 分），スライスを利用した GP1 は 122 分（A1: 119 分，A2: 128 分，A3: 120 分）となっており，平均時間を見ると GP1 の方が GP2 よりも 43 分短くなっている．また平均値の差の検定（ウェルチの検定）[9] を，有意水準 5%で行うと有意な差が見れた．この結果から Exp1 においては，スライスを利用した方が効率良くフォールト位置特定が行えることが確認できた．

3.4.2 Exp2

Exp2 では，スライスを利用しなかった GP1 の被験者のフォールト位置特定に要した平均時間は 133 分（A1: 118 分，A2: 126 分，A3: 155 分），スライスを利用した GP2 は 114 分（B1: 131 分，B2: 92 分，B3: 118 分）となっており，平均時間を見ると GP2 の方が GP1 よりも 19 分短くなっている．しかし，平均値の差の検定（ウェルチの検定）[9] を，有意水準 5%で行ったが有意な差が見られなかった．

Exp1，Exp2 ともに，各フォールトごとに見ると，スライスが有効であるような，すなわち 5%の有意水準で有意な差があるフォールトがそれぞれ 3 個と 2 個存在することが確認できた．これらのフォールトは，主に，条件文の誤りや代入文におけるデータ更新の誤りであった．

4. 評価実験 2

評価実験 1 の Exp2 ではスライスを用いた場合と用いなかった場合で有意な差が確認できなかった．その一つの原因は被験者が少なかったことにある．ここでは被験者を大幅に増やして行った評価実験 2 について述べる．

4.1 実験概要

- 被験者

大阪大学基礎工学部情報科学科の 2 年生 34 人である．すべての被験者は，1 年生のときに Pascal プログラミングの講義を受講しており，数百行のプログラムを作成する能力をもつ．また，次に述べる 6 種類のプログラムについても演習で作成したり，講義でそれについて学習している．1 年生で学んだ知識は全員まだ十分覚えているため，被験者は同程度の能力であると考えられる．

- 対象プログラムリスト

フォールトが 1 個だけ含まれるプログラムリストを 6 種類（それぞれ，P1，P2，P3，P4，P5，P6 とする）と，それぞれのフォールトに対して，「誤った値を出力している出力文で，誤った値を保持していた変数」をスライシング基準としてスライスを抽出した際に，スライスに含まれる文に下線が引いてあるプログラムリスト（図 3 参照）（それぞれ，P1'，P2'，P3'，P4'，P5'，P6' とする）を用意する（P1 と P1' は，スライス情報が含まれていること以外は，同じプログラムリストであり，残りの 5 個の対応するプログラムリストも同様である）．

P1〜P6（P1'〜P6'）の 6 個のプログラムは，Pascal で記述されている．各 P1〜P6，及び P1'〜P6' が含むフォールトとしては，(a) 変数名の誤り（wrong variable name），(b) 文の誤り（wrong statement）の

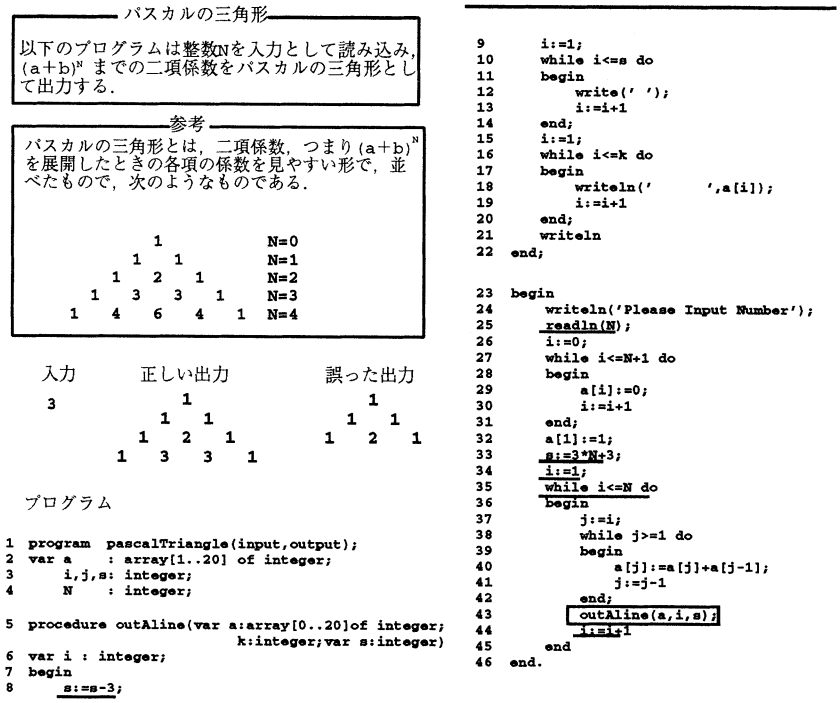

図 3　被験者に与えた問題
Fig. 3　Example of the program. (P3')

表 4　対象プログラムと含まれるフォールト
Table 4　Target programs and faults.

	プログラム	フォールト
P1	素因数分解	文の誤り [F1]
P2	素数	変数名の誤り [F2]
P3	パスカルの三角形	文の誤り [F3]
P4	数値計算	変数名の誤り [F4]
P5	順列	変数名の誤り [F5]
P6	ソート	文の誤り [F6]

表 5　用意したプログラムのサイズ
Table 5　The size of the target programs.

プログラム	P1	P2	P3	P4	P5	P6
サイズ（行）	25	31	46	37	49	35
プログラム	P1'	P2'	P3'	P4'	P5'	P6'
スライス（行）	5	7	7	13	18	18
削減率（%）	80	77	85	65	63	49

2 種類を含めた [11]．また，Pi(Pi') に含まれるフォールトを Fi とする．プログラム及びフォールト内容を表 4 に示す．

表 5 に P1〜P6 のプログラムサイズ（行），P1'〜P6' のスライスに含まれる文のサイズ（行），及び削減率（(プログラムのサイズ−スライスのサイズ）/プログラムのサイズの割合）を示す．

- 被験者の行う作業

34 人の被験者を学籍番号の下 1 けたが奇数であるグループ G1（15 人）と偶数である G2（19 人）に分け，G1 に含まれる被験者はスライス情報が含まれる 6 種類のプログラムリスト（P1'〜P6'）を対象として，G2 に含まれる被験者は 6 種類の単なるプログラムリスト（P1〜P6）を対象としてフォールト位置特定を行う．**3.** で述べた実験 1 では，スライス抽出機能をもったデバッグツール上でフォールト位置特定を行ったが，本実験では，紙にプリントされたプログラムリスト上で机上デバッグを行う．

実際に被験者に与えたプリントを図 3 に示す．図 3 は，スライス情報を含むプログラムリスト P3' である．このプリントには，以下が含まれる．

– プログラムが実現する機能．

– プログラムに与える入力，フォールトが含まれることによる誤った出力，プログラムが本来，出力すべき正しい出力．

表 6　評価実験 2 データ（単位：分）
Table 6　Data of the 2nd experiment. (min)

	G1（15 人）		G2（19 人）
P1'	3.27	P1	3.32
P2'	6.47	P2	8.11
P3'	7.13	P3	11.63
P4'	5.73	P4	4.74
P5'	15.07	P5	16.79
P6'	3.07	P6	4.53
合計	40.73	合計	49.11

– フォールトが 1 個だけ含まれるプログラムリスト（図 3 の例では，35 行目の $while\ i \leqq N\ do$ が誤りであり，正しくは $while\ i \leqq N+1\ do$ である）．

● 評価方法

各フォールト位置特定に要した時間を計測し，G1, G2 間で統計的に比較・分析を行う．

4.2 実験プロセス

4.1 で示した 6 個のプログラムリストを P1〜P6（P1'〜P6'）の順番で，被験者がフォールト位置特定を行った．実験の流れは，以下のとおりである．

（1） 例題の説明

ここでは，被験者に (3) の Step1〜Step3 で行う作業を十分理解してもらうことを目的とした．

（2） P1〜P6（P1'〜P6'）を含むプリントの配布

（3） 実験開始

P1〜P6（P1'〜P6'）の各プログラムリストごとに以下の作業を行う．

（Step1） プログラムの実現する機能を理解する．

（Step2） 入力とその入力に対する正しい出力，誤った出力からフォールトを認識する．

（Step3） プログラムリストを読み，フォールト位置を特定する．

（4） 実験終了

すべてのプログラムのフォールト位置特定を行った時点で実験終了とする．

4.3 実験結果

本実験で計測するデータは **4.2** で述べた (3) の Step3 に要した時間である．被験者が，各プログラムのフォールト位置特定に要した平均時間（単位：分）を表 6 に示す．

4.4 分析・評価

すべてのプログラムのフォールト位置特定に要した平均時間は，スライス情報を含むプログラムリスト（P1'〜P6'）を対象としたグループ G1 では約 41 分，単なるプログラムリスト（P1〜P6）を対象としたグループ G2 では約 49 分となっている．平均時間だけ見ると G1 の方が G2 より約 8 分短くなっている．また，平均値の差の検定（ウェルチの検定）を行うと，有意水準 5%で有意な差が見れた [9]．この結果から，スライスを利用した方が効率良くフォールト位置特定を行えることが確認できた．

4.5 考　　察

フォールト別に位置特定に要した時間について平均値の差の検定を有意水準 5%で行うと，P3（P3'）に含めた F3 と P6（P6'）に含めた F6 で有意な差が検出された．被験者はデバッグ時には，正しい出力と誤った出力を見比べ，フォールト内容を認識し，プログラムが実現する機能からプログラムのアルゴリズムを考え，大まかにフォールト位置を推定した後に，実際にソースコードを読み，フォールト位置を特定する．上述の F3，F6 は，フォールト内容からフォールト位置を推定する際に，その推定が難しいフォールトであるため，被験者はプログラムを理解する必要があった．このような場合には，プログラム全体を理解するよりもスライスにより範囲を限定し理解する場合が有効であるため，F3，F6 に関しては有意な差が検出されたと考えられる．また F3 は，すべてのフォールトの中で，スライスの利用によるデバッグ対象の削減率が最も高い約 85%となっているために（表 5，図 3 を参照），有意な差が検出できたと考えられる．逆に，F4 に関してはフォールト内容が非常に簡単で，一意にフォールト位置を特定できるようなものであったために，表 6 に示すような結果となっている．

大規模なソフトウェアのデバッグにおいては，F3 や F6 のようにフォールト内容から，フォールト位置を推定するのが困難な場合が多い．このような場合に，スライスを用いることで，デバッグの対象となる範囲を限定し，効率の良いデバッグ（フォールト位置特定）が行えると考えられる．

5. むすび

本研究では，スライスがフォールト位置特定に有効であるかどうかを実験的に評価した．実験の結果，スライスを用いた方が，スライスを用いない場合よりも効率良くフォールト位置特定作業が行えることが確認できた．また，これまでに我々は文献 [7] で保守プロセスのプログラム理解においてスライスが有効であることも確認している．これらの結果から実際の開発/保守現場への適用においてもスライスは有効であると

考えている．新しい技術を実際の開発現場で利用する際には，利用者に対して新しい技術の教育や技術の獲得などに時間を要する場合が多く，更に従来の開発手法を変更する必要が生じる場合もある．しかし，スライスは非常に少ない労力で利用でき，従来の手法に簡単に導入できる技術であると考えられる．実際に，本実験において被験者にはスライスの概念を説明するために非常に少ない時間しか要していない．

本実験で利用したスライシングツール [8] は実験を目的としたものであるために対象言語が Pascal となっているが，現在，Sapid [4] 等を利用し大規模プログラムに対するスライシングツールの開発を行う予定である．更に開発したシステムを利用し，実際に運用されているような実用的なプログラミング言語を対象とした実験を行いたいと考えている．

謝辞　実験に協力頂いた大阪大学基礎工学部情報科学科 2 回生に感謝します．本研究は，一部文部省科学研究費特定領域研究 (A)(2)(課題番号：10139223) の補助を受けている．

文　献

[1] H. Agrawal and J. Horgan, “Dynamic Program Slicing,” SIGPLAN Notices, vol.25, no.6, pp.246–256, 1990.
[2] D.C. Atkison and W.G. Griswold, “The design of whole-program analysis tools,” Proceedings of the 18th International Conference on Software Engineering, pp.16–27, 1996.
[3] V.R. Basili, G. Caldiera, and H.D. Rombach, “Goal question metric paradigm,” Encyclopedia of Software Engineering, ed. J.J. Marciniak, vol.1, John Wiley & Sons, pp.528–532, 1994.
[4] 福安直樹，山本晋一郎，阿草清滋，“細粒度ソフトウェア・リポジトリに基づいた CASE ツール・プラットフォーム Sapid,” 情処学論，vol.39, no.6, pp.1990–1998, 1998.
[5] B. Korel and J. Laski, “Dynamic slicing of computer programs,” Journal of Systems Software, vol.13, pp.187–195, 1990.
[6] G.J. Myers, The Art of Software Testing, Wiley-Interscience, 1979.
[7] 西松　顯，楠本真二，井上克郎，“保守プロセスに対するプログラムスライスの実験的評価,” 信学技報，SS97-87, March 1998.
[8] 佐藤慎一，飯田　元，井上克郎，“プログラムの依存関係解析に基づくデバッグ支援システムの試作,” 情処学論，vol.37, no.4, pp.536–545, 1996.
[9] 芝　祐順，渡部　洋，石塚智一編：統計用語辞典，新曜社，1984.
[10] 下村隆夫，プログラムスライシング技術と応用，共立出版，1995.
[11] 下村隆夫，“変数値エラーにおける Critical Slice に基づくバグ究明戦略,” 情処学論，vol.33, no.4, pp.501–511, 1992.
[12] 植田良一，練　林，井上克郎，鳥居宏次，“再帰を含むプログラムのスライス計算法,” 信学論（D-I），vol.J78-D-I, no.1, pp.11–22, Jan. 1995.
[13] University of Wisconsin, “The wisconsin program-slicing tool 1.0, reference manual,” Computer Sciences Department, University of Wisconsin-Madison, Aug. 1997.
[14] M. Weiser, “Programmers use slices when debugging,” Communications of the ACM, vol.25, no.7, pp.446–452, 1982.
[15] M. Weiser, “Program slicing,” Proceedings of the Fifth International Conference on Software Engineering, San Diego, CA, pp.439–449, 1981.
[16] 山崎利治，“共通問題によるプログラム設計技法解説,” 情報処理学会誌，vol.25, no.9, p.934, 1984.
[17] M. Zelkowitz and D.R. Wallace, “Experimental models for validating technology,” IEEE Software, vol.31, no.5, pp.23–31, 1998.

（平成 10 年 10 月 26 日受付，11 年 4 月 21 日再受付）

西松　顯

平 9 阪大・基礎工・情報卒．平 11 同大大学院博士前期課程了．同年，NTT データ通信（株）入社，現在に至る．プログラムスライスの研究に従事．

西江　圭介

平 9 阪大・基礎工・情報卒．平 11 同大大学院博士前期課程了．同年，松下通信工業（株）入社，現在に至る．

楠本　真二　（正員）

昭 63 阪大・基礎工・情報卒．平 3 同大大学院博士課程中退．同年同大・基礎工・情報・助手．平 8 同大講師．平 11 同大助教授．工博．ソフトウェアの生産性や品質の定量的評価，プロジェクト管理に関する研究に従事．情報処理学会，IEEE 各会員．

井上　克郎　（正員）

昭 54 阪大・基礎工・情報卒．昭 59 同大大学院博士課程了．同年同大・基礎工・情報・助手．昭 59〜61 ハワイ大マノア校・情報工学科・助教授．平 1 阪大・基礎工・情報・講師．平 3 同学科・助教授．平 7 同学科・教授．平 11 奈良先端大・情報科学研究科・教授併任．工博．ソフトウェア工学の研究に従事．