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2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．2つの実験では，被験者を2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，複雑化にともな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付けられてきている．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．従って，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2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言われており[6], [9]，フォールトの位置を効率よ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全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[21], [22]．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捜し出す．文献[19]においてHorwitzらは，9種類のCのプログラム(732行～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元のプログラムの約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元のプログラムの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言われているが，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2つのグループ

[†] 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

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[24]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3つの手法(1)Observational methods (2)Historical methods (3)Controlled methods について述べており，本研究のように大学環境で実験を実施する場合には，擬似的に大規模な環境を作り出す(3)をおこなうことが適してい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全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[21], [22]．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(quasi-static slice [20]，conditional slice [10]，interface slice [4]，transform slice [7]，dynamic slice [?] [?]，program dice [8] 等)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[18]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および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 Dependence，略してDD）関係および条件文や繰り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．

```
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の元のプログラム
Fig. 1　Sample Program

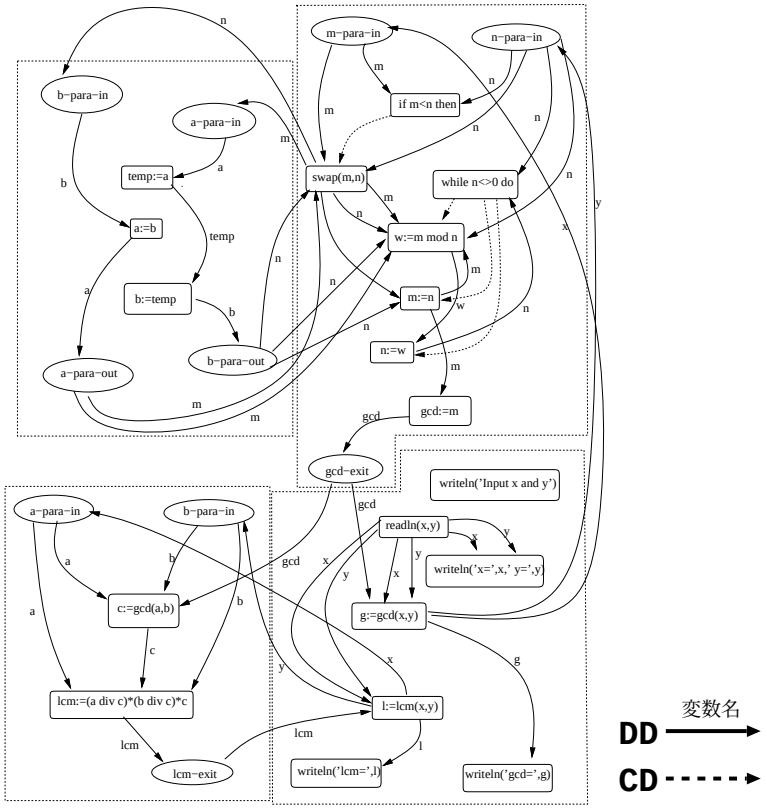

図2　PDGの例
Fig. 2　Example of PDG

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関係の辺を辿って文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[18]．特に，PDGを与えられた節点から辺の順方向に辿って得られた集合をforwardスライスと呼び，逆方向に辿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[18]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[14]．次節3.節で述べる評価実験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，以下の機能を持っている．

(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．

(機能3)　：スライス

スライスを計算し抽出する機能．

従って，3.節で述べる評価実験を行う上では十分な機能を持っている．

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

(注1)：変数はスカラ型のみで，条件文(if)，代入文，繰り返し文(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．

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[23]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2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．まず，GP1の被験者は2.2節で示した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2.2節で示した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年生の時の演習で，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持っている．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．

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～P1.9	
	(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)

3.2　対象プログラム

実験ではいわゆる酒屋問題[23]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) 配列の初期化洩れ．
(F1.5) 関数処理(関数呼び出し文)の記述洩れ．
(F1.6) データ更新の誤り．
(F1.7) 手続きのパラメータ渡しの誤り．
(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，...，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，1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

実験の手順は次の通りである．

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

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

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)[15]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．また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3個存在す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)[15]を，有意水準5%で行ったが有意な差が見られなかった．しかし，各フォールトごとに見ると**Exp1**と同様に，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2個存在することが確認できた．

4.　評価実験2

評価実験1のExp1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．全ての被験者は，大学でプログラミング演習を受講しているので，プログラミング及びデバッグには慣れている．

- 対象プログラムリスト

フォールトが一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，もっとも一般的に選択されるスライシング基準で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)．

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]

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2種類を含めた[17]．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)，及び限定率(スライスのサイズ/プログラムのサイズ　の割合)を示す．

パスカルの三角形

以下のプログラムは整数Nを入力として読み込み，$(a+b)^N$ までの二項係数をパスカルの三角形として出力する．

参考

パスカルの三角形とは，二項係数，つまり$(a+b)^N$を展開したときの各項の係数を見やすい形で，並べたもので，次のようなものである．

```
          1          N=0
        1   1        N=1
      1   2   1      N=2
    1   3   3   1    N=3
  1   4   6   4   1  N=4
```

入力　正しい出力　誤った出力

```
3          1            1
         1   1        1   1
       1   2   1    1   2   1
     1   3   3   1
```

プログラム

```
1  program  pascalTriangle(input,output);
2  var a    : array[1..20] of integer;
3      i,j,s: integer;
4      N    : integer;

5  procedure outAline(var a:array[0..20]of integer;
                          k:integer;var s:integer)
6  var i : integer;
7  begin
8      s:=s-3;
9      i:=1;
10     while i<=s do
11     begin
12         write(' ');
13         i:=i+1
14     end;
15     i:=1;
16     while i<=k do
17     begin
18         writeln('      ',a[i]);
19         i:=i+1
20     end;
21     writeln
22 end;

23 begin
24     writeln('Please Input Number');
25     readln(N);
26     i:=0;
27     while i<=N+1 do
28     begin
29         a[i]:=0;
30         i:=i+1
31     end;
32     a[1]:=1;
33     s:=3*N+3;
34     i:=1;
35     while i<=N do
36     begin
37         j:=i;
38         while j>=1 do
39         begin
40             a[j]:=a[j]+a[j-1];
41             j:=j-1
42         end;
43         outAline(a,i,s);
44         i:=i+1
45     end
46 end.
```

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

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
限定率(%)	25	23	15	35	37	51

- 被験者の行う作業

34人の被験者を学籍番号の下一桁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持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．

– 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節で示した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

全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節で述べた(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．

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

4.4 分析・評価

全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[15]．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，おお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，全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限定率が最も高い約15%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よ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[13]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，さら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[14]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[5]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．さら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] Agrawal, H., and Horgan, J.: "Dynamic Program Slicing", *SIGPLAN Notices*, Vol.25, No.6, pp. 246–256 (1990).

[2] Atkison, D. C. and Griswold, W. G., "The Design of Whole-Program Analysis Tools", In *Proceedings of the 18th International Conference on Software Engineering*, pp. 16–27, 1996.

[3] Basili, V. R., Caldiera, G., Rombach, H. D., *Goal Question Metric Paradigm*, in John J. Marciniak, editor, Encyclopedia of Software Engineering, vol.1, John Wiley & Sons, pp.528-532(1994).

[4] Beck, J. and Eichmann, D., "Program and Interface Slicing for Reverse Engeneering", Proceedings of ICSE-15, pp. 509–518, 1993.

[5] 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.

[6] Korel, B., and Laski, J., "Dynamic Slicing of Computer Programs", *Journal of Systems Software*, Vol.13, pp. 187–195 (1990).

[7] Lanubile, F. and Visaggio, G., "Extracting Reusable Functions by Flow Graph-Based Program Slicing", IEEE Transactions on Software Engineering, Vol. 23, No. 4, pp. 751-761, April, 1997.

[8] Lyle, R. and Weiser, M., "Automatic program bug location by program slicing", Proceedings 2nd International Conference on Computers and Applications, pp. 877–883 ,1987.

[9] Myers, G. J., "The Art of Software Testing", Wiley-Interscience(1979).

[10] Ning, J., Engerts, A. and Konzaczynski, W., "Automated support for legacy code understanding", Communications of the ACM, vol. 37, no. 5, pp. 50-57, 1994.

[11] 西江, 神谷, 楠本, 井上, "プログラムスライスに基づくデバッグ支援ツールの実験的評価", ソフトウェアシンポジウム97予稿集, pp.142-147(1997).

[12] 西松, 楠本, 井上,"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", 信学技報, SS98-3, May 1998.

[13] 西松, 楠本, 井上, "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, Mar 1998.

[14] 佐藤, 飯田, 井上, "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.

[15] 芝, 渡部, 石塚 編: 統計用語辞典, 新曜社 (1984).

[16] 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.

[17] 下村, "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.

[18] 植田, 練, 井上, 鳥居, "再帰を含むプログラムのスライス計算法", 信学論, Vol. J78-D-I, No. 1, pp. 11–22, 1995.

[19] University of Wisconsin., "The Wisconsin Program-Slicing Tool 1.0, Reference Manual", *Computer Sciences Department, University of Wisconsin-Madison*, August, 1997.

[20] Venkatesh, G., "The semantic approach to program slicing", In Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation, pp. 80-91, 1991.

[21] Weiser, M., "Programmers use slices when debugging", Communications of the ACM, Vol. 25, No.7, pp. 446-452(1982).

[22] Weiser, M., "Program slicing", Proceedings of the Fifth International Conference on Software Engineering , San Diego, CA, pp. 439–449 ,1981.

[23] 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.

[24] Zelkowitz, M. and Wallace, D. R.: "Experimental models for validating technology", *IEEE Software*, Vol.31, No. 5, pp.23-31 (1998).

（平成年月日受付，月日再受付）

西松　顯

平8阪大・基礎工・情報卒．現在同大学院修士課程在学中．プログラムスライスの研究に従事．

西江　圭介

平8阪大・基礎工・情報卒．現在同大学院修士課程在学中..

楠本　真二　（正員）

昭63阪大・基礎工・情報卒．平3同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．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～昭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．工博．ソフトウェア工学の研究
に従事．