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　楠本 真二　井上 克郎

大阪大学大学院基礎工学研究科情報数理系専攻

〒560-8531 大阪府豊中市待兼山町 1-3 基礎工学研究科

Phone: 06-850-6571　Fax: 06-850-6574
E-mail: {a-nisimt, kusumoto, inoue}@ics.es.osaka-u.ac.jp

あらまし　これまでに我々はプログラムスライスがフォールト位置特定に有効であるかを実験的に評価した．しかし，この評価実験では，被験者数が 6 人と少ないために，スライスの有効性が十分に確認できなかった．そこで，本研究では，これまでの評価実験の問題点である被験者の数を増やし，プログラム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することを目的とする．具体的には，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，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 間で比較を行なった．実験の結果，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特定を行なった方が，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特定を行なった場合より効率良くフォールト位置特定が行なえる事が確認できた．

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

An Experimental Evaluation of Program Slicing on Fault Localization Process

Akira Nishimatsu, Shinji Kusumoto and Katsuro Inoue

Graduate School of Engineering Science, Osaka University

1-3 Machikaneyama-cho, Toyonaka-shi, Osaka 560-8531, Japan

Phone: +81-6-850-6571　Fax: +81-6-850-6574
E-mail: {a-nisimt, kusumoto, inoue}@ics.es.osaka-u.ac.jp

Abstract　This paper aims to evaluate the usefulness of program slicing on fault localization process. In the experiments, we prepare six kinds of program(P1～P6) that included only one fault and thirty-four subjects. The subjects are divided into two groups: G1 and G2. Next, the subjects in G1 specify any fault in six programs successively, using the slicing technique and one in G2 specify any fault in six programs successively, using the slicing technique. Finally we compare the time for fault localization between G1 and G2. The results of the experiment show that program slicing technique is effective on fault localization process.

Key words　program slice, debug process, fault localization

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 [2, 3]，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 [8, 9]．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．

スライスはフォールト位置特定に有効であると言われているが，実際に有効であるかどうかはほとんど確認されていない．そこで我々は，文献 [4] において，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．文献 [4] の実験は，スライスの有効性を実験的に評価しているが，被験者の数が非常に少ないために，スライスが有効であるという結果が十分に得られなかった．

本研究では，文献 [4] での問題点である被験者の数を十分に多くし，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することを目的とする．具体的には，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 間で比較を行なった．実験の結果，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特定を行なった方が，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特定を行なった場合より効率良くフォールト位置特定が行なえる事が確認できた．

以降，2. ではスライスについて述べる．3. では文献 [4] で行なった評価実験とその結果について簡単に述べる．4. では大学環境で行なった評価実験とその結果について述べる．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

これまでに我々は文献 [6] 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．

2.1 プログラム依存グラフ（PDG）

PDG はプログラム内の文の依存関係を表すグラフである．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．

DD は，各節点の到達定義集合（Reaching Definitions，略して RD）を求めることによって得られる．PDG 上でのある節点 t の RD とは，変数 v と節点 s との組 (v,s) の集合である．これは，

- プログラム中の文 s で変数 v を定義している．
- プログラム中の 2 つの文 s から t へのすべての実行パスの中で，v を定義しないパスが少なくとも 1 つ存在する．

ことを示している．t の RD に (v,s) が含まれ，かつ t が v を参照するとき，s から t への DD 関係があるという．

また，ある条件判定部分 s の結果により文 t の実行の有無が決定されるとき，s と t との間に CD が存在するものとする．すなわち CD は if 文や while 文の条件判定部分からそれらの内部ブロックに属する文への影響であり，これはプログラムを解析すれば容易に求められる．

一般にプログラムには複数の手続きが定義されており，各手続き間には引数や大域変数を通じて DD 関係が生じる．これらの DD 関係を表すため

に，PDG にプログラム中の文とは直接対応しない節点（中継節点と呼ぶ）を用意する．

PDG の作成は，プログラムを解析し，プログラムの各文を PDG の節点に切りわけ，プログラム中の各文における RD を求め，それをもとにして PDG の各辺を生成することによって行われる．詳細は，文献 [6] を参照されたい．

```
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

例として図 1 のプログラムに対応する PDG を図 2 に示す．図 2 の PDG の中で角の丸い四角がプログラム中の各文に対応する節点で，楕円が中継節点である．また，有向辺のうち，実線で名前がついているものが DD 関係の辺で，破線のものが CD 関係の辺である．実線についている名前は，その DD 関係の辺が影響を伝える変数の名前である．また，破線で囲まれている部分はプログラム上の１つの手続きを表す．

2.2 スライシング

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 [6]．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 に述べる．このプログラムの 36 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．

2.3 スライシングツール

これまでに我々は，文献 [6] 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 [5]．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にデバッグを行なってもらった．スライシングツールの対象言語は，以下のような Pascal のサブセットである．

- 文として，代入文，条件文，繰り返し文，手続き呼出文，begin–end で囲まれる複合文を扱う．
- 手続きは再帰呼び出しも扱う．手続きの引数の渡し方については，値渡しと変数渡しの 2 種類がある．
- 変数のデータ型はスカラー型のみでポインタ型は扱わない．具体的には整数型，文字型，論理型およびそれらを要素に持つ配列型とした．

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，以下の機能を持ち，スライシングを行うことによってプログラムの参照範囲を小さくすることにより開発・保守の支援を行う事が可能である．

(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．

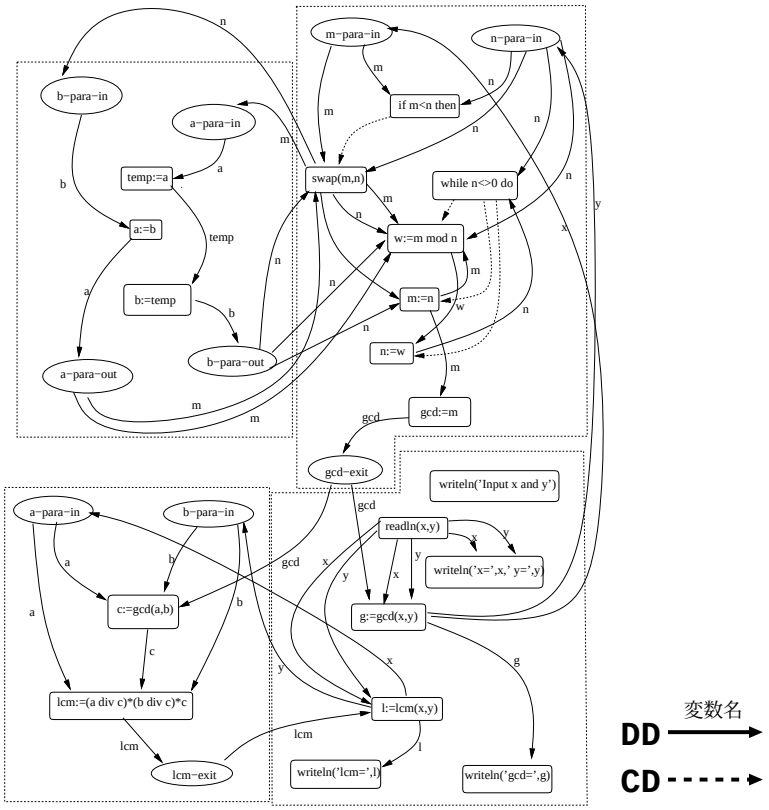

図 2: PDG の例

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 [10] 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 G1 と G2 にそれぞれ 3 人ずつ分ける．G1 に含まれる被験者を A1, A2, A3，G2 に含まれる被験者を B1, B2,B3 と呼ぶ．まず，G1 の被験者は 2.3 節で示したスライシングツールの機能 1 〜 3 を利用して，G2 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なう (これを **Exp1** とする)．次に，G2 の被験者は 2.3 節で示したスライシングツールの機能 1 〜 3 を利用して，G1 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)P2.1 〜 P2.9 のフォールト位置特定を行なう (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

	G1(3 人)	G2(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 [10] 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事ができる．P1 に対して 8 個，P2 に対して 9 個のフォールトを考えた．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 (P1.1) ～ (P1.8) 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 ($Testdata_1$ ～ $Testdata_8$)．尚，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～P2.2 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.

表 2: **Exp1** データ

	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

(単位:分)

表 3: **Exp2** データ

	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

(単位:分)

3.5 分析・評価

3.5.1 Exp1

Exp1 では，スライスを利用しなかった G2 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 G1 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 G1 の方が G2 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)[7] を，優位水準 1%で行うと優位な差が見れた．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事が確認できた．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 個存在する事が確認できた．

3.5.2 Exp2

Exp2 では，スライスを利用しなかった G1 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 G2 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 G2 の方が G1 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)[7] を，優位水準 5%で行ったが優位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 個存在する事が確認できた．

4 評価実験 2

本実験はスライスが，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評価することを目的とする．

また，本実験では，評価実験 1 の問題点である被験者の数を改善するために，被験者の数を大幅に増やしている．

4.1 実験内容

- 被験者
  大阪大学基礎工学部情報科学科の 2 年生の学生 34 人．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事以外は，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

	プログラム	フォールト
P1	素因数分解	条件文の誤り [F1]
P2	素数	変数名の誤り [F2]
P3	パスカルの三角形	条件文の誤り [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	条件文の誤り [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，一般にスライスが有効であると言われてる，(a) 変

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
5	1 1 1 1 2 1 1 3 3 1	1 1 1 1 2 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

数名の誤り (wrong variable name)，(b) 文の誤り (wrong statement) のみを含め，スライスが有効でないと言われている (c) 文の欠如 (missing statement) は含めなかった．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: 用意したプログラムのサイズ

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

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．

- 被験者の行なう作業
  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上 (P1' ～ P6') で，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上 (P1 ～ P6) を対象

として，フォールト位置特定を行なう．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なったが，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なう．

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事による誤った出力，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 $while\ i <= N\ do$ が，正しくは $while\ i <= N+1\ do$ であるフォールトを含んでいる．)

- 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・分析を行なう．

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なった．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) において行なう作業を十分理解してもらう事を目的とした．
(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なう．
(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な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: 実験データ (単位：分)

	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) を行なうと，有意水準 2.5%で有意な差が見れた [7]．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なえる事が確認できた．

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全体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)，有意水準 1%で有意な差が検出できたと考える．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，スライスを用いる事で，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．

4.6 実験 1 との比較

通常，デバッグに要する時間は個人の能力に大きく依存する．そこで，3．の実験 1 では，実験の信頼性を高いものにするために，各被験者がスライスを利用したデバッグと利用しないデバッグを行ったが，本節で説明した実験 2 では，以下の 3 つの理由から，G1 と G2 の条件を入れ替えて (G1:スライスなし，G2:スライス利用)，同様の実験を行なう必要はないと考えた．

1 被験者の数が多い．
2 被験者が同じ学年である (同程度の能力である)．
3 被験者をランダムに G1，G2 に振り分けた．

5 まとめと今後の課題

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保守作業が行えることが確認できた．

今後の課題は，大規模プログラムに対するスライシングツールの開発があげられる．

参考文献

[1] Agrawal, H. and Horgan, J. R.:“Dynamic Program Slicing”, *Proc. ACM SIGPLAN'90 Conference on Programming Language Design and Implementation*, ACM Press, New York, N.Y., pp. 246-256 (1990).

[2] Korel, B. and Laski, J.:“Dynamic Slicing of Computer Programs”, *J. Systems Software*, Vol. 13, pp. 187-195 (1990).

[3] Myers, G. J.: *The Art of Software Testing*, Wiley-Interscience(1979).

[4] 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).

[5] 佐藤，飯田，井上:“プログラムの依存解析に基づくデバッグ支援ツールの試作”，情報処理学会論文誌, Vol. 37, No.4, pp.536-545(1996).

[6] 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”, 電子情報通信学会論文誌,Vol. J78-D-I, No.1, pp.11-22(1995).

[7] 芝，渡部，石塚 編: 統計用語辞典, 新曜社 (1984).

[8] Weiser, M.:“Programmers use slices when debugging”, *Communications of the ACM*, Vol. 25, No.7, pp. 446-452(1982).

[9] Weiser, M.:“Program Slicing”, *IEEE Trans. on Soft. Eng.*, Vol.10, No. 4, pp. 352-357(1984).

[10] 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，25，9, p.934(1984).