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

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

あらまし　本研究では，プログラムスライスに基づく開発支援システムにおけるプログラムスライス抽出機能が実際の保守作業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．先ず，仕様書とその仕様書に対応するプログラムを用意し，被験者を2つのグループに分ける．次に，仕様書の変更を，一方のグループの被験者はスライス抽出機能を用いて，もう一方のグループの被験者はスライス抽出機能を用いずにプログラムに反映してもらう．最後に，各グループ毎に，プログラム修正に要した時間を比較する．実験の結果，開発支援システムのプログラムスライス抽出機能を用いた場合の方が，プログラムスライス抽出機能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．また，本実験では仕様書の変更として，機能の追加と機能の変更の2種類を用意した．仕様の変更ごとに分析すると，プログラムスライス抽出機能は，機能の変更の方が機能の追加より有効である事が確認できた．

キーワード　プログラムスライス，保守プロセス，プログラム理解

An Experimental Evaluation of Program Slicing on Software Maintenance Process

Akira Nishimatsu, Shinji Kusumoto and Katsurou 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 software maintenance process. In the experiments, we prepare two kinds of program(P1 and P2) and six subjects. The subjects are divided into two groups: G1 and G2. Next, the subjects in G1 and G2 modify P1 and P2 respectively, without using the slicing technique. Then, one in G1 and G2 modify P2 and P1 respectively, using the slicing technique. Finally we compare the time for modification between G1 and G2. The results of the experiment show: (a)that program slicing technique is effective on software maintenance process. (b) that using the slicing technique is more useful in changing the function in the original program than in adding new functionto the original program.

Key words　program slice, maintenance process, program understanding

1　まえがき

ソフトウェア開発におけるコストの大部分は，その保守プロセスに費されている．保守プロセスの主な時間はプログラム理解の時間と指摘されている [9]．そのためプログラム理解を支援する方法の開発が重要となっている．

プログラム理解を支援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いる．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 [10, 11]．スライス技法を利用するプログラム理解では，変更すべき部分を特定するためにスライスを抽出し，そのスライス内を理解することで，プログラム理解を効率良く行うことができる．

これまでに，文献 [6] において，静的スライス (プログラムの依存関係を解析し，それに基づいて求められるスライス) を用いた開発支援システムを開発してきている．このシステムを用いることにより，プログラム中で変更を行うべき部分をスライスとして抽出し，スライスに含まれる部分にのみ注目することで，プログラム理解を効率良く行うことが期待される．

本研究では，文献 [6] で開発された開発支援システムが実際の保守プロセスにおけるプログラム理解の作業に有効であるかを実験的に評価することを目的とする．具体的には，まず 6 人の被験者 (2 つのグループ G1 と G2 に分ける) と 2 種類のプログラム P1，P2 を用意する．まず，G1 が P1 を，G2 が P2 を，それぞれ支援システムのスライス抽出機能を用いずに保守作業を行う．次に，G1 が P2 を，G2 が P1 を，それぞれ開発支援システムのスライス抽出機能を用いて保守作業を行う．最後に G1，G2 の間で保守作業に要した時間についてスライス抽出機能を用いた場合と用いなかった場合での比較を行う．実験の結果，ツールのスライス抽出機能を用いた方が，スライス機能を用いない方よりも効率よく保守作業が行えることが確認できた．

以降，2. では開発支援システムの基礎となっているプログラム依存関係解析，スライスについて述べる．3. では開発支援システムの機能について紹介する．4. では大学環境で行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　プログラム依存関係解析

文献 [6] の開発支援システムは，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．

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 [7] を参照されたい．

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　スライス

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 [7]．

```
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

特に，PDG を与えられた節点から辺の順方向に探索して得られた集合を forward スライスと呼び，逆方向に探索して得られた集合を 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．

3 開発支援システム

本章では，文献 [6] で試作した開発支援システムの概要について述べる．本システムの実装には，解析や実行部には C 言語を，インタフェースには Tcl/Tk を用いた．

3.1 言語仕様

本システムが対象とする言語は，以下のような仕様を持つ 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．

3.2 システムの機能と動作

本システムは次のような機能を持つ．

(F1) 通常のデバッガと同様にプログラムのステップ実行，変数の参照，ブレークポイントの設定などが行える．

(F2) スライスとして backward ，forward のどちらでも抽出でき，また，そのときに，PDG のうち制御依存とデータ依存の片方だけを探索したり，希望の段階までを探索することもできる．

ユーザが保守の対象となるプログラムを入力すると，システムは，構文解析を行った後，依存関係の解析を行う．それらの解析が終了すると，PDG および実行に必要な構文木や変数の情報などが生成される．

ユーザはこの段階でシステムにどの動作をさせるかを指定することができる．ユーザがプログラムを実行させた場合 (F1) には，入出力はシステム内で行ない，その際，ユーザは，通常の実行だけでなく，ステップ実行やブレークポイントの設定，変数値の参照などのデバッグに必要な作業も行うことができる．

スライスを抽出する場合には，ユーザは，まずどの文の，どの変数に関するスライスを抽出するのかといった情報を指定する．それに基づきスライスが計算され，そのスライス結果は，元のプログラムの部分プログラムとして出力される．

上記のように，ユーザはシステム内ですべての保守作業を行なう事ができ，その際にスライス抽出機能を用いることが可能である．開発支援システムの全体図は図3に示す．

4 適用実験

4.1 実験概要

実験の目的は開発した開発支援システムのスライス抽出機能が，実際のプログラムの保守作業に有効であるかどうかを実験的に評価する．実験の概要を表1に示す．具体的には，酒屋問題 [12] に対する仕様書 (S1) とレストランの座席予約システム問題 (以降，予約問題と略す) に対する仕様書 (S2) と，それぞれの仕様書に対応するプログラムを 2 種類ずつ用意する (S1 に対応する 2 種

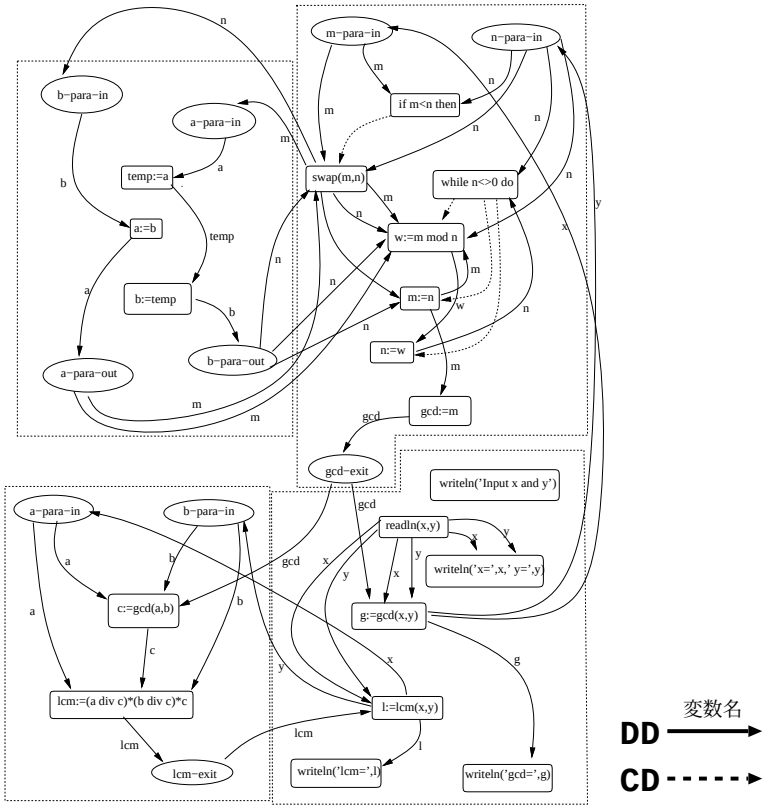

図 2: PDG の例

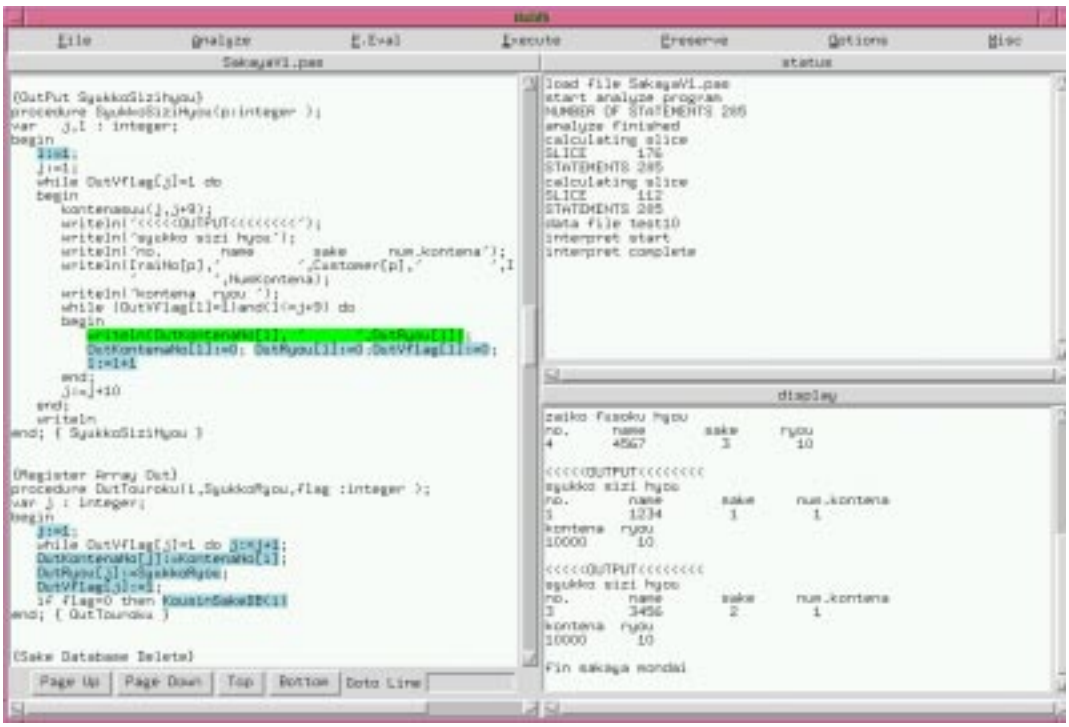

図 3: システム図

類のプログラムを，それぞれ P1.1，P1.2，S2 に対応する 2 種類のプログラムを，それぞれ P2.1，P2.2 とする)．

また，被験者 6 人を 2 つのグループ G1 と G2 に分ける．まず，G1 の被験者 (A1，A2，A3) が P1.1 及び P1.2 を，

G2(B1，B2，B3)の被験者がP2.1及びP2.2を，それぞれ支援ツールのスライス抽出機能を用いずに保守作業を行う．次に，被験者に対してスライスの講義を行ない，実際にスライスの保守への適用例を示す事で，被験者のスライスへの理解を深める．そして，G1の被験者がP2.1及びP2.2を，G2の被験者がP1.1及びP1.2を，それぞれ開発支援システムのスライス抽出機能を用いて保守作業を行う．上記のように被験者は各問題に対して2回，全体では計4回の保守作業を行う．表1の**Exp1.1，1.2，2.1，2.2**の詳細は4.3節 で示す．

表 1: 実験概要

	G1(3人)	G2(3人)
プレ実験	図書管理問題	
適用実験1	**Exp1.1** 酒屋問題 (P1.1,P1.2) (スライス利用不可)	**Exp1.2** 予約問題 (P2.1,P2.2) (スライス利用不可)
講義	スライス	
適用実験2	**Exp2.1** 予約システム問題 (P2.1,P2.2) (スライス利用)	**Exp2.2** 酒屋問題 (P1.1,P1.2) (スライス利用)

なお，6人の被験者は学部3年生の時の演習で，デバッグ支援システムの入力となる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開発支援システムの中で使用する機能についての講習，及び実験で想定する保守作業のプレ実験を事前に行っており支援システムの使用についても慣れている．

4.2 保守作業

本実験における保守作業とは被験者に対してプログラムと仕様書を与え，その仕様書に対する変更要求を被験者にプログラムに対して反映してもらうというものである．保守作業の流れを図4に示す．ここで被験者の行なう保守作業は，まず，プログラム理解として与えられた仕様書の内容を理解し，プログラムがどのような機能を実現しているのかを理解し (仕様書理解)，仕様書に書かれているある機能がプログラムのどの部分で実現されているかを対応づける (仕様書とプログラムの対応)．仕様書を理解した後に，変更部分認識として与えられた仕様書の変更を理解し (変更内容理解)，変更するべきプログラムの部分を認識する (変更内容とプログラム変更部分の対応)．プログラム中の変更すべき部分を認識できれば，被験者は実際にプログラムを変更する．ここでは，プログラムを編集，コンパイル，変更内容を正しく実現できているかを確認するテスト，そして，仮にフォールトが存在する場合はデバッグを行なう．

図4に示す『仕様書とプログラムの対応』,『変更内容とプログラム変更部分の対応』,『デバッグ』の3つの作業は開発支援システムのスライス抽出機能を利用することで，効率良く作業が行なえると考えられる．しかし，実際にはデバッグにおいてはスライス抽出機能を利用した被験者はいなかったためプログラムの対応，変更内容とプログラム変更部分の対応の2つの作業においてスライス抽出機能を利用する場合と利用しない場合とで要した時間に差が生じると考えられる (詳しくは5.3節で述べる)．

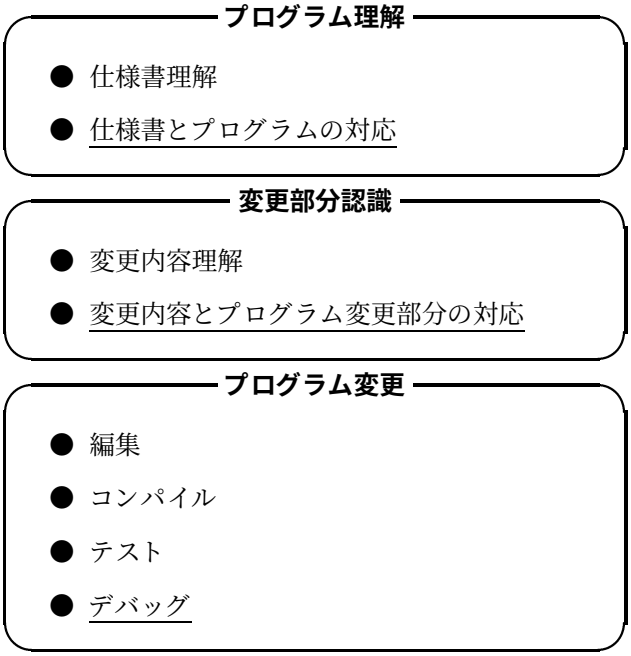

図 4: 保守作業

4.3 実験手順

各被験者が実験 (**Exp1.1，1.2，2.1，2.2**) で行なう保守作業の手順を以下のStep1～Step4に示す．仕様書S(i)に対する変更をM(i)，仕様書S(i)を実現したプログラムをP(i)，仕様書の変更M(i)をプログラムP(i)に反映させたプログラムをP'(i)，P'(i)が正しく変更M(i)を反映していることを保証するテストデータの集合をT(i)とする．下記のStep1～Step4を適用実験1ではG1は酒屋問題，G2は予約問題をスライス抽出機能を用いずに行ない，適用実験2ではG1は予約問題，G2は酒屋問題をスライス抽出機能を利用して行なう．

Step1 仕様書S(i)を被験者に与える．ここでは被験者は仕様書S(i)を理解する．

Step2 プログラムリストP(i)，仕様書の変更M(i)を被験者に与える．ここでは被験者はプログラムリストP(i)と仕様書の変更M(i)を理解し，プログラムリストP(i) の変更部分を認識する．

Step3 被験者は仕様書の変更M(i)をプログラムP(i)に反映させ，プログラムP'(i)を作成する．

Step4 プログラムP'(i)が正確に仕様書の変更を実現できた場合に変更終了とする．以下の2つの条件を満たした場合に変更が実現できたとする．

条件1 字句解析・構文解析・意味解析におけるフォールトがない．

条件2 T(i)に属する全てのテストデータに対して正しい出力が得られる．

4.4 用意した仕様変更

本実験で用意した仕様変更は各問題ごとに2種類用意した．それぞれの仕様変更は元の仕様書への新たな機能の追加と元の仕様書に既にある機能の変更を実現するものに対応する．具体的に以下に示す．

- 酒屋問題
  - □ 機能の追加 出力データの操作
  - □ 機能の変更 入力データフォーマットの変更
- 予約問題
  - □ 機能の追加 新たな入力データの処理
  - □ 機能の変更 入力データフォーマットの変更

各仕様変更について見ると，機能の追加は，既存のプログラムのコードはほぼ利用できないが，機能の変更は既存のプログラムのコードが再利用できるという特徴がある．

4.5 実験結果

本実験で計測するデータは4.3節で述べたStep2～Step4に要した時間である．本来，本実験ではスライス抽出機能の利用により保守作業が効率良く行なえるStep2の部分のみの時間を計測し，評価するべきである．しかし被験者がプログラムの理解，仕様書の変更の理解，プログラムの変更部分の認識を正しく行なったかどうかを正確に判断できないと考えたため(例えば，非常に短い時間でプログラム理解を終了した被験者が，プログラム変更に多くの時間を費した場合，被験者のプログラム理解が不十分である事が言える)，実際にプログラムを正しく変更する作業までの時間を計測，評価する事にした．

各仕様書の変更のプログラムへの反映に要した時間(単位:分)に関するデータを表2と表3に示す(表2が酒屋問題，表3が予約問題に関するデータである)．

表 2: 酒屋問題

	P1.1	P1.2	合計
A1	75	50	125
A2	133	89	223
A3	188	301	489
平均	132.0	146.0	279.0
B1	126	87	213
B2	110	60	170
B3	90	64	154
平均	108.7	70.3	179.0

表 3: 予約問題

	P2.1	P2.2	合計
B1	105	95	200
B2	99	106	205
B3	112	126	238
平均	105.3	109.0	214.3
A1	44	37	81
A2	82	69	151
A3	78	74	152
平均	68.0	60.0	128.0

5 分析・評価

5.1 酒屋問題

酒屋問題の2種類の仕様変更に要した平均時間は，システムのスライス抽出機能を利用したG2では179.0分(B1:213 B2:170 B3:154)，スライス抽出機能 を利用しなかったG1では279.0分(A1:125 A2:223 A3:489)となっている．平均時間を見るとG2の方がG1よりも仕様変更に要した時間が100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)を有意水準5%で行なったが，有意な差が見られなかった[8]．よって，スライス抽出機能を用いた方が効率良く保守作業を行なえることは確認できなかった．酒屋問題において有意な差が見られなかった事に関する考察は5.4節で詳しく述べる．

5.2 予約問題

予約問題の2種類の仕様変更に要した平均時間は，システムのスライス抽出機能を利用したG1では128.0分(A1:81 A2:151 A3:152)，スライス抽出機能を利用しなかったG2では214.3分(B1:200 B2:205 B3:238)となっている．平均時間を見るとG1の方がG2よりも仕様変更に要した時間が約86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，有意水準2.5%で行うと有意な差が見れた[8]．この結果から，スライス抽出機能を用いた方が効率良く保守作業を行なえることが確認できた．

5.3 スライス抽出機能の利用

一般にスライスはデバッグ時に以下の3つのフォールトの内(1)(2)には有効であるが，存在しないコードはスライスとして抽出できないため(3)のフォールトには有効でないと言われている(我々は，これらの結果を[4]において実際に確認している)．

(1) 変数名の誤り(wrong variable name)

(2) 文の誤り(wrong statement)

(3) 文の欠如(missing statement)

以上を本実験の仕様変更時のスライス抽出にあてはめると，機能の変更においては，既に存在するコードから変更部分をプログラムスライスとして抽出できるが，機能の追加においては，新たにコードを追加する必要があり，追加されるコードはスライス抽出時には存在しないため，変更部分認識時にはスライス抽出機能は有効ではないと考えられる．実際に各仕様変更ごとに分析すると，スライス抽出機能は予約問題においては機能の追加よりも機能の変更の方がより有効であることが確認できた(機能の追加では平均値の差の検定(ウェルチの検定)を，有意水準5%で有意な差が見られたが，機能の変更では2.5%で有意な差が見られた)．

また，被験者A3は表2，表3から他の被験者よりもスライス抽出機能を用いた場合の方が，用いない場合よりも飛躍的に時間が短縮されていることが分かる．これは被験者A3がスライス抽出機能を用いることで，プログラム理解と変更部分の認識を非常に効率良く行なえたと考えられる．

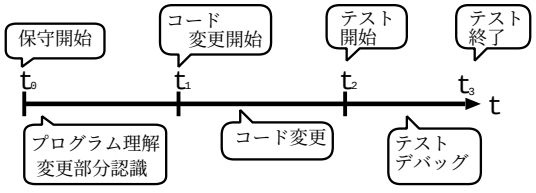

図 5: 保守作業の流れ

5.4 考察

本実験では，4.3節のStep2～Step4に要した時間を評価の対象としたため，プログラムの変更作業の時間も評価対象となった．プログラムの変更作業は被験者のプログラマ能力に非常に依存するため，すべての被験者のプログラマ能力が同程度であることが望ましいが，本実験では被験者A1が突出したプログラマ能力を有していたため，酒屋問題においては有意な差が見られない結果となった．

本実験で用意したプログラムのサイズを表4に示す．各プログラムは300～400行程度と小規模ではあるが，スライスを用いることで参照範囲を表5に示すように45%～52% に限定できる(表5は被験者が実験において抽出した典型的なスライスのサイズである)．大規模なプログラムにスライス抽出技術を適用する場合においても本実験と同様に参照範囲を限定することができ，保守作業を効率良く行なえると考えられる．

表 4: 用意したプログラムのサイズ

プログラム	P1.1	P1.2	P2.1	P2.2
サイズ(単位:行)	387	434	333	424

表 5: プログラムスライスのサイズ

プログラム	P1.1	P1.2	P2.1	P2.2
サイズ(単位:行)	176	210	164	220
限定率(単位:%)	45.5	48.4	49.2	51.9

6 まとめと今後の課題

本研究では，プログラムスライスに基づく開発支援システムが実際の保守作業に有効であるかどうかを実験的に評価した．実験の結果，開発支援システムのスライス抽出機能を用いた方が，スライス抽出機能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，(1)システムの拡張，(2)大規模プログラムへの適用，(3)他言語への適用などがあげられる．

謝辞 本研究は，一部文部省科学研究費補助金重点領域研究(2)「発展機構を備えたソフトウェアの構成原理の研究」(課題番号:09245218)の補助を受けている．

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] 西松, 井上:“依存関係解析に基づく開発支援システムへの動的スライス抽出機能の追加”, 情報処理学会第 55 回（平成 9 年後期）全国大会 講演論文集 (1), page418-419.

[6] 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).

[7] 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).

[8] 芝, 渡部, 石塚 編: 統計用語辞典, 新曜社 (1984).

[9] Robson, D.J et al.,“Approaches to Program Comprehension”, J.System Software, Vol. 14, No. 1, 1991.

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

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

[12] 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).