部品変更履歴に基づく重み付き依存グラフを用いた部品の洗練

丸山勝久† 島 健一†

本論文では，過去の部品変更の履歴に基づきプログラム内の依存関係に割り付けた重みを用いることで，既存のプログラムから抽出した部品を将来再利用される可能性が高い部品に自動洗練する手法を提案する．我々は，頻繁に再利用されたソースコード断片は開発者が将来も同じ形で再利用する可能性が高い部分，また，頻繁に変更を受けたソースコード断片は開発者が将来も同様の変更を行う可能性が高い部分であると仮定する．このような仮定に基づき，重み付きプログラム依存グラフにおける2節点間の依存関係の強度を重みで定義し，開発者が変更後に再利用した部品の形および部品変更前後の部品の形の変化に基づき重みを変動させる．本洗練手法では，依存関係矢印の重みを指標として用い，もとの部品ソースコードから重み値が大きい強依存関係で結合したソースコード断片を抜き出す．抜き出したソースコード断片に対して，重みから計算した洗練コストにより洗練の妥当性を判定することで，開発者の再利用ドメインに適した洗練部品の形を決定することが可能である．本手法により部品洗練を行うことで，再利用時に開発者が部品を適時変更する手間が減少する．

Refining Reusable Software Components by Using Weighted Graphs Based on Modification Histories

KATSUHISA MARUYAMA† and KEN-ICHI SHIMA†

This paper presents a method for refining reusable software components by using weighted program dependence graphs, whose edges have weights based on the modification histories of the components. The weight is the connection strength between two nodes with respect to data or control dependences. The method uses two kinds of weight calculated based on 1) the state of the connection between dependent nodes when reusing the components and 2) the change in the connection between dependent nodes before and after modification of the components. Nodes connected by dependence edges with high weight values are included in a refined component but edges with low values are broken when refining it. The component is replaced with the refined component only when the condition defined by our proposed refining costs is satisfied. Therefore, source code components in a user library can be automatically modified into components that may be often reused in the future with few modifications. The experimental results for the proposed method demonstrate the advantages of introducing this method into source code reuse.

1. はじめに

部品を直接再利用するソースコード部品化再利用では，開発者の要求する部品がライブラリに存在するとき，開発者があらたにソースコードを記述する手間が省略される[1]．よって，プログラムの開発コストを減少するためには，開発者の要求する部品があらかじめライブラリに用意されている必要がある．これに対して，既存のプログラムから部品を抽出することで，部品作成の負担を軽減する方法が提案されている[2)~6)]．しかしながら，部品がどのような形で再利用されるのかを部品作成時に予測することは困難であり，抽出した部品が開発者の要求を満たすとはかぎらない．要求する部品が見つからない場合，開発者は部品の再利用をあきらめるか，自分の要求に合わせて類似の部品を変更することになる．よって，ソースコードの部品化再利用を促進するためには，部品作成を容易にするだけではなく，作成した部品が無変更あるいは少ない変更で再利用可能であることが重要である．

部品作成者の負担を増加させずに，部品変更に対する開発者の負担を軽減するため，我々は，開発者の過去の部品変更の履歴から将来再利用される可能性の高いソースコード断片を予測し，ライブラリ内の各部品を自動的に洗練する手法を提案する．本論文では，プログラム依存グラフ（PDG: Program Dependence Graph）[7]における各節点間の依存関係に対して，開発者が再利用した部品の形および部品変更前後の部品

† NTT ソフトウェア研究所
NTT Software Laboratories

の形の変化に基づき変動する2種類の重みを割り付けた重み付き依存グラフ（WPDG: Weighted PDG）を定義する．部品の形とは，部品内部に含まれるプログラム文を指す．また，部品の形の変化とは，変更前の部品内部に含まれるプログラム文と，変更後の部品内部に含まれるプログラム文の違いを指す．

これら2種類の重みを開発者の再利用ドメインの特性に適した部品を決定する指標として用い，もとの部品のソースコードから重み値が大きい強依存関係で結合したソースコード断片を，洗練後部品の候補として抜き出す．ここで，再利用ドメインの特性とは，開発者が部品を再利用してどのような機能を持つプログラムを頻繁に作成するのかを指す．さらに，本論文では，開発者の再利用ドメインにおいて，現在の重みに基づき部品洗練を適用することが妥当であるかどうかを判定するため，洗練前後の部品の重みから計算する洗練コストと洗練条件を定義する．本手法により部品洗練を行うことで，ライブラリ内の部品が開発者の要求する機能を満たす可能性が高くなり，開発者が類似の要求に応じて部品を適時変更する手間が減少する．

以下，2章では，洗練対象とする部品と部品変更を示し，WPDGの定義および変更履歴に基づく本手法の概要を述べる．3章では，部品内部の依存関係に割り付ける2種類の重みを定義し，重みの変動操作と計算式を示す．4章では，依存関係の強度に基づきWPDGから洗練部品に含まれる節点を抽出する手続きを述べ，洗練コストおよび洗練条件を定義する．5章では，評価実験の結果より本手法の有効性について考察し，6章で本手法の課題を述べる．

2. 重みを用いた部品洗練手法

本章では，洗練対象とする部品と本洗練手法が扱う部品変更を示す．次に，重み付き依存グラフを定義し，本洗練手法の概要を述べる．

2.1 部品洗練における仮定

本研究では，個人あるいは小規模プロジェクトでの部品化再利用によるプログラム開発を支援対象とする．このような開発では，開発者が作成するプログラムは類似しており，自分あるいはプロジェクト内で作成した部品を再利用する機会は多く，過去の部品変更を模倣する可能性は高い．よって，開発者が変更した後の部品を開発者の要求を満たす部品と見なすと，部品の洗練に関して，次に示す2つの仮定を導くことができる．

仮定1 部品内部で過去に頻繁に再利用されたソースコード断片は，開発者のライブラリにおいて，将来も同じ形で再利用される可能性が高い．

仮定2 部品内部で過去に頻繁に変更されたソースコード断片は，開発者のライブラリにおいて，将来も同様の変更を受ける可能性が高い．

本洗練手法では，仮定1により，部品内部において，頻繁に再利用された部分を特定する．また，仮定2により，頻繁に変更を受けた（依存関係が結合あるいは分断された）部分を特定する．

2.2 洗練対象部品

本洗練手法が扱う部品は，手続き型言語Cのサブセットで記述されたプログラムで，静的依存解析が可能であることが前提である．本手法では，プログラムの依存関係をPDGで表す．

部品は，作成者が再利用対象としてライブラリ内に登録した部品本体と，部品本体内部の文に依存関係を持つ付加ソースコード断片で構成される．本論文では，部品本体を C_{body}，付加ソースコード断片を C_{add}，これら2つを合わせた部品全体を C_{whole} と表す．C_{body} は，開発者がそのままの形で再利用すると予測されるソースコード断片の集まりである．また，C_{add} は，開発者が C_{body} と組み合わせて再利用可能な付加ソースコード断片，あるいは洗練により C_{body} に付加される可能性のあるソースコード断片の集まりである．本洗練手法が対象とする部品は，区間限定スライシングに基づく部品作成手法[6]を用いることで，容易に抽出可能である．

図1 に，洗練対象部品の例を示す．各文の左側に示

```
    print_sum(char *filename)
    /* print the sum of input data */
    {
      FILE *fp
      double x;   /* input data */
      int n;      /* the number of data */
      double sum;
      double mean;
[1]   fp = fopen(filename, "r");
[2]   if (fp != NULL) {
[3]     sum = 0;
[4]     n = 0;
[5]     fscanf(fp, "%lf", &x);
[6]     while (x > 0) {
[7]       sum = sum + x;
[8]       n++;
[9]       fscanf(fp, "%lf", &x);
        }
[10]    printf("sum = %lf\n", sum);
[11]    mean = sum / n;
[12]    printf("mean - %lf\n", mean);
[13]    fclose(fp);
      }
    }
```

$$C_{whole} = C_{body} + C_{add}$$

図1 洗練対象部品の例
Fig. 1 Example of a component to be refined.

す括弧付きの数字は，このソースコードから作成したPDGにおける節点の識別番号である．太字の文（節点1，2，3，5，6，7，9，10，13）はC_{body}であり，「入力データの合計を表示する」機能を持つ．その他の文（節点4，8，11，12）はC_{add}であり，「入力データの個数を求める」機能を持つソースコード断片と「平均値を求め，表示する」機能を持つソースコード断片を含む．このように，本手法では，C_{body}とC_{add}は明確に区別されて，開発者に提示される．

2.3 部品変更

本洗練手法を用いたプログラム開発において，開発者はC_{whole}内部のC_{body}をそのまま，あるいは適時変更して再利用する．本手法では，部品変更前後で同一な節点を識別するために，C_{whole}の各文にPDGの節点番号をタグとして割り付け，無変更な文に対して変更前後で共通のタグ番号を持たせる．タグ付け機構とバージョン管理機構を有するエディタを用いることで，以下に示す5つの部品変更操作を洗練システムが検出する．

(a) C_{body}にC_{add}内のタグを持つ文を付加する．
(b) C_{body}内の文を削除する．
(c) C_{body}にタグを持たない新規文を追加する．
(d) C_{body}内の文をタグを変えずに移動する．ただし，移動した文と他のすべての文との依存関係が変わらないとき，その節点は無変更と見なす．他の文との依存関係に変化が生じたとき，この変更は移動文の削除と新規文の追加として扱う．
(e) C_{body}内の文に対して，タグを変えずに内容だけ書き換える．この変更は書き換え文の削除と新規文の追加として扱う．

図2に開発者が変更後に再利用した部品C_{reuse}の例を示す．A，D，I，M，Rは，それぞれ上記の(a)付加操作，(b)削除削除，(c)追加操作，(d)移動操作，(e)書き換え操作を表す．ここで，本論文が対象とする変更は，開発者の要求に合わせて部品のソースコードを改造することを指し，誤りを含むソースコードの修正は変更として扱わない．

2.4 重み付きプログラム依存グラフ

部品に対する過去の利用および変更の履歴を部品ごとに蓄積するため，C_{whole}のPDGにおける節点間の依存関係矢印に重みを割り付ける．このPDGを，重み付き依存グラフ（WPDG）と呼ぶ．重みは矢印の接続元および接続先節点間の依存関係の強度を表す．

本洗練手法では，依存関係を持つ2つの節点を依存節点と呼び，WPDG上の矢印を接続元および接続先の2つの依存節点により識別する．よって，依存関係矢印の重みを計算する際には，2.3節に示す5つの変更操作から，各矢印の依存節点の状態を調べ，C_{body}に付加する文に対応する付加節点の集合およびC_{body}から削除する文に対応する削除節点の集合を求める．たとえば，図2に示す変更後部品において，付加節点は4，8，削除節点は1，2，10である．また，変更操作(c)，(d)，(e)において，新規文の追加は新規節点と新規矢印の追加として扱い，新規矢印に新たに重みを割り付けることはしない．

本論文では，節点pから節点qへの重みwを持つデータ依存関係と制御依存関係を$p \xrightarrow{w} q$と表す．強度に関係しない場合，節点pとqの依存関係を$p \rightarrow q$と表す．また，$node(G)$，$edge(G)$は，それぞれWPDG（あるいはPDG）G内の節点集合と矢印集合を指す．WPDGにおいてENTRY節点は洗練対象とせず，この節点を接続元とする矢印には重みを割り付けない．

図1に示す部品C_{whole}に対して，部品の再利用と変更を何回か繰り返した後のWPDGを**図3**に示す．

```
    1:    print_sum_num(char *filename)
    2:    {
    3:      double x;
    4:      int n;
    5:      double sum;
    6:      double mean;
R   7: [1]  fp = fopen(filename, "r+");
D   8: [2]
    9: [3]  sum = 0;
M  10:[10]  printf("sum = %lf\n", sum);
A  11: [4]  n = 0;
   12: [5]  fscanf(fp, "%lf", &x);
   13: [6]  while (x > 0) {
   14: [7]    sum = sum + x;
A  15: [8]    n++;
   16: [9]    fscanf(fp, "%lf", &x);
   17:      }
I  18:      printf("num = %d\n", n);
   19:[13]  fclose(fp);
   20:    }
```

A: add I: insert
D: delete M: move
R: rewrite

C_{reuse}

図2 変更後部品の例
Fig. 2 Example of a modified component.

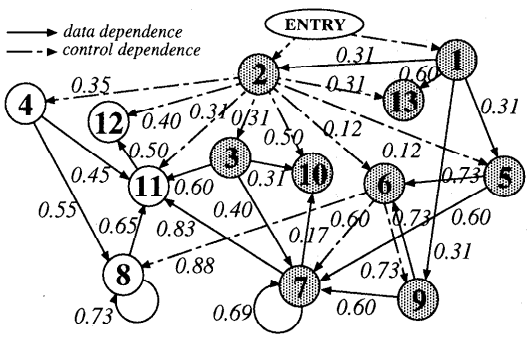


図3 重み付きプログラム依存グラフの例
Fig. 3 Example of a WPDG.

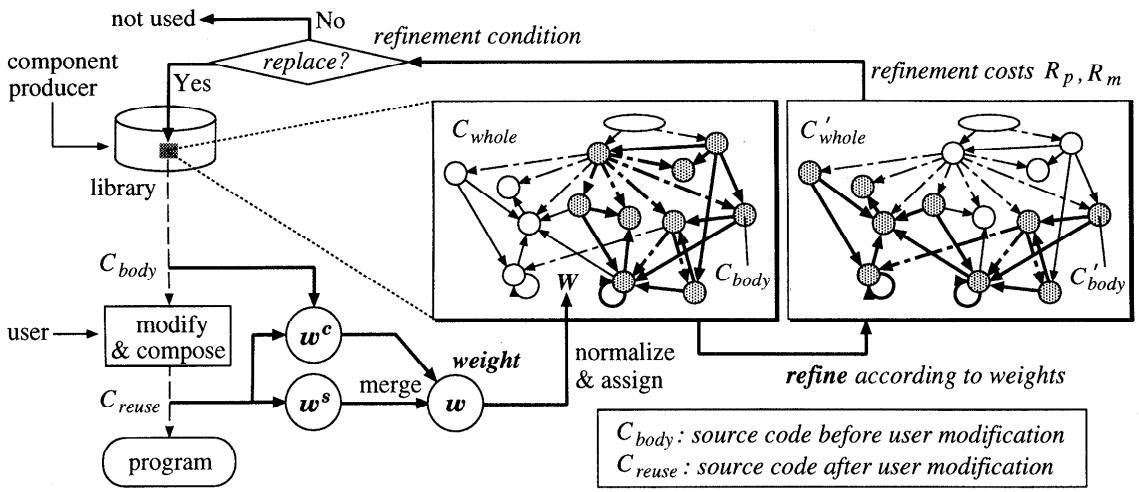


図4 部品洗練手法の概要
Fig. 4 Overview of proposed method.

網かけの節点は部品本体 C_{body} を表す．

2.5 洗練手法の概要

部品洗練とは，部品変更履歴に基づく依存関係の強度に関する重みを指標として，部品本体 C_{body} の形を将来再利用される可能性が高いソースコード断片を含むように変形することを指す．図4に本洗練手法の概要を示す．左側のWPDGは洗練前（現在）の部品 C_{whole}，右側のWPDGは洗練後の部品 C'_{whole} を表す．各WPDGにおいて，網かけの節点は洗練前の部品本体 C_{body} と洗練後の部品本体 C'_{body} を指す．部品洗練の動作は，WPDGの各矢印に対する重み付けと，重みに基づく部品本体の抽出に分けられる．

開発者は，洗練システムの部品検索ブラウザを通して，ライブラリから要求部品を取得する．取得した部品 C_{whole} において，C_{body} をそのまま，あるいは一部変更して再利用した場合を考える．C_{reuse} は，開発者が変更後に再利用した部品を指す．本手法では，変更後の C_{reuse} の形から部品のどの部分を利用したのかに基づき変動する重み w^s と，変更前の C_{body} と変更後の C_{reuse} の形の変化から部品のどの部分を変更したのかに基づき変動する重み w^c をそれぞれ求める．これらの重みに対して，合成演算および正規化演算を行った後の値 W を，C_{whole} のWPDGの各矢印に割り当てる．

重みが変動した際には，依存関係矢印の重みを指標として，C'_{body} に含まれる節点および矢印を C_{whole} から選択し，C'_{body} を含む C'_{whole} を依存関係を保持したまま作成する．C_{whole} と C'_{whole} は，内部に含む C_{body} と C'_{body} が異なるだけである．次に，洗練前後の部品 C_{body} および C'_{body} の形を維持するために必要なコスト R_p，依存関係の状態を修正するために必要なコスト R_m を計算し，これらのコストに関する洗練条件により洗練の妥当性を判定する．洗練条件が成立する場合，ライブラリ内の C_{whole} を C'_{whole} と置換し，洗練が終了する．洗練条件が成立しない場合は，C'_{whole} を破棄し，ライブラリ内の C_{whole} および C_{body} はそのままにする．

3. 部品変更履歴に基づく重み付け

本章では，部品洗練における2つの仮定に基づく2種類の重みを提案し，それぞれの重みの変動操作と重みの計算式を定義する．

3.1 部品利用および変更に基づく重み

本洗練手法では，部品洗練における仮定1，2に基づき，WPDGの各依存関係矢印に対して，次に示す2種類の重み w^s，w^c を定義する．

(1) **部品利用に基づく重み** w^s: 変更後に再利用した部品 C_{reuse} のPDGにおいて，依存節点の接続関係に基づく重み．依存節点の接続関係とは，依存関係矢印の接続元節点と接続先節点が，変更後部品の内部に存在するのか，あるいは外部に存在するのかという位置関係を指す．

(2) **部品変更に基づく重み** w^c: 変更前の部品本体 C_{body} と変更後に再利用した部品 C_{reuse} のPDGにおいて，依存節点の接続関係の変化に基づく重み．依存節点の接続関係の変化とは，変更前後で依存関係矢印の接続元節点および接続先節点の位置関係がどのように変化したのか，

つまり，依存関係矢印が結合あるいは分断されたのかを指す．

いま，将来再利用される可能性の高い部品の形を決定するという観点だけから部品洗練を行う場合，洗練における仮定1により，開発者が過去に頻繁に再利用したソースコード断片を特定すればよい．つまり，実際に開発者が再利用した変更後部品 C_{reuse} の形に応じて変動する重み w^s を導入するだけで十分である．

しかしながら，再利用した部品の形が同じでも，無変更で再利用した場合と変更をともなって再利用した場合では，開発者の負担に差があるはずである．たとえば，C_{body} の形をまったく変えずに再利用した（C_{reuse} が C_{body} に等しい）場合に比べて，C_{body} に文を追加して作成した C_{reuse} を再利用した場合の方が，開発者の負担は大きい．単純に変更後部品の形だけに注目して重みを変動させた場合，このような開発者の負担の差が部品洗練に反映されない．さらに，我々の目的は積極的に部品洗練を行うことで，開発者の部品再利用を支援することである．しかしながら，相反する再利用を繰り返した場合，たとえば，ある1つの文を含む部品と含まない部品を順番に繰り返し再利用した場合，その文に対応する節点に接続された矢印の重み w^s は同じ値の間を振動する．このため，重み w^s の導入だけでは，開発者が過去に何度も部品変更を行っているにもかかわらず，部品洗練がまったく行われない可能性がある．

そこで，本洗練手法では，同じ形の部品を再利用した場合でも開発者の負担の有無を区別し，洗練における仮定2に基づく重み w^c を導入する．仮定2において，同じ変更が将来も繰り返し行われることは，開発者に対して同じ部品変更の負担が将来も繰り返し生じることを指している．よって，仮定2に基づき，将来変更される可能性が高いソースコード断片を特定し，これらのソースコード断片が他のソースコード断片に比べて，部品洗練時に変更されやすくなるように重み付けを行うことは，開発者の部品変更に対する負担を軽減するという目的を達成するのに妥当である．さらに，開発者の変更を反映した重み w^c を変更後部品の形に基づく重み w^s と合成することで，相反する再利用を繰り返した場合でも，洗練により部品の形を変化させることが可能となる．

3.2 依存節点の位置関係

重み付けの際には，WPDG上の接続元および接続先の2つの依存節点により，各依存関係矢印を識別する．つまり，接続元および接続先節点が変更前後でそれぞれ同一であるとき，それらを接続する矢印も同一

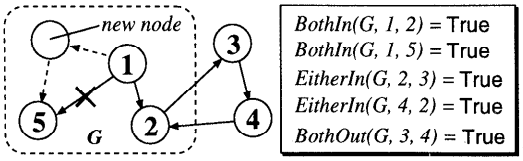


図5 依存節点の位置関係
Fig. 5 Position of dependent nodes.

であると見なす．いま，矢印の方向を無視すると，依存関係矢印の接続関係は，部品内部と外部という2つの領域に対する依存節点の位置関係で表すことができる．依存関係矢印 $p \to q$ における依存節点の接続関係を表す論理式を以下に示す．G はWPDG（あるいはPDG）を指す．

結合 依存節点 p，q が両方とも G に含まれる⇔ $BothIn(G,p,q) \equiv p \in node(G) \wedge q \in node(G)$.

分断 依存節点 p，q が G の内部と外部に分断される⇔ $EitherIn(G,p,q) \equiv p \in node(G) \wedge q \notin node(G) \vee p \notin node(G) \wedge q \in node(G)$.

不定 依存節点 p，q が両方とも G に含まれない⇔ $BothOut(G,p,q) \equiv p \notin node(G) \wedge q \notin node(G)$. 両依存節点とも G に含まれないとき，依存節点の関係は結合あるいは分断のどちらの状態か確定できないため，不定とする．

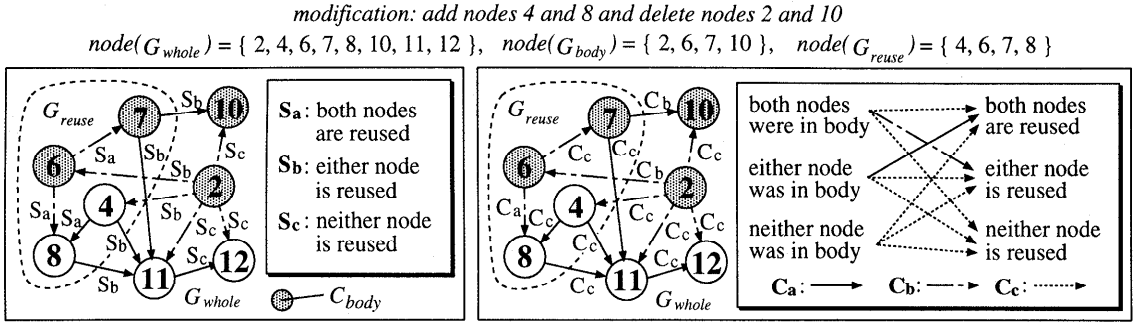
これらの論理式の真偽により，3.1節で述べた2種類の重みに対する変動操作を定義する．

図5 に，依存節点の接続関係の例を示す．WPDG G に存在する各矢印に対して，右側に示す論理式だけが真となる．ここで，矢印 $1 \to 5$ については，部品変更時に新規節点が追加されたため依存関係が破壊されている．しかし，これらの節点は，新規に追加した節点を経由して依存関係を持つので，$BothIn(G,1,5)$ が真となる．このように，本洗練手法では，依存節点が新規に追加した節点を経由して間接的に依存関係を持つ場合ももとの依存節点間に依存関係が存在すると見なし，依存関係矢印の状態を依存節点の位置関係だけから判定する．

3.3 依存節点の接続関係に基づく重み付け

洗練における仮定1に基づき，C_{whole} のWPDG G_{whole} において，過去に頻繁に再利用されたソースコード断片を特定する．このため，依存関係が保存される矢印の重みを増加させ，依存関係が破壊される矢印の重みを減少させる．G_{whole} に存在する依存関係矢印を $p \to q \in edge(G_{whole})$，再利用時の部品 C_{reuse} のPDGを G_{reuse} と表す．G_{reuse} における依存節点の存在位置に基づき，重み w^s を次のように操作する．

操作Sa G_{whole} 内の依存節点の両方とも再利用する



(a) 依存関係矢印の接続関係に基づく重みw^s　　(b) 依存関係矢印の接続関係の変化に基づく重みw^c

図 6 依存関係矢印の状態と状態の変化に基づく重み付け操作
Fig. 6 Operations for varying the weights.

($BothIn(G_{reuse},p,q)$ が真) とき，依存関係の強度を強くする（重み w^s 増加）．

操作 Sb G_{whole} 内の依存節点の片方だけ再利用する（$EitherIn(G_{reuse},p,q)$ が真）とき，依存関係の強度を弱くする（重み w^s 減少）．

操作 Sc G_{whole} 内の依存節点の両方とも再利用しない（$BothOut(G_{reuse},p,q)$ が真）とき，依存関係の強度を変えない（重み w^s 変動なし）．

図 6 (a) に，重み付け操作 Sa, Sb, Sc を示す．C_{whole}, C_{body} は図 3 の WPDG の一部である．網かけは C_{body} 内部の文に対応する節点を指す．

重み w^s は，開発者がどのような形で部品を再利用したかに影響を受ける．重み w^s の値が大きいほど，それらの依存節点を同時に再利用した割合が多いことを指す．逆に，重み w^s の値が小さいほど，それらの依存節点を片方だけ再利用した割合が多いことを指す．

3.4 依存節点の接続関係の変化に基づく重み付け

洗練における仮定 2 に基づき，部品 C_{whole} の WPDG G_{whole} において，過去に頻繁に変更されたソースコード断片を特定する．このため，依存関係が分断から結合に変化する矢印の重みを増加，依存関係が結合から分断に変化する矢印の重みを減少させる．G_{whole} に存在する依存関係矢印を $p \to q \in edge(G_{whole})$，変更前の部品本体 C_{body} の WPDG を G_{body}，再利用時の部品 C_{reuse} の PDG を G_{reuse} と表す．G_{body} と G_{reuse} における依存節点の存在位置の変化に基づき，重み w^c を次のように操作する．

操作 Ca G_{whole} において G_{body} の内部と外部に分断していた依存節点を両方とも再利用する（$EitherIn\ (G_{body},p,q) \wedge BothIn(G_{reuse},p,q)$ が真）とき，依存関係の強度を強くする（重み w^c 増加）．

操作 Cb G_{whole} において G_{body} 内部で結合していた依存節点の片方だけ再利用する（$BothIn(G_{body},p,q) \wedge EitherIn(G_{reuse},p,q)$ が真）とき，依存関係の強度を弱くする（重み w^c 減少）．

操作 Cc 部品変更前後で依存節点の位置関係が変化しない，または，依存節点の両方とも再利用しない（$BothIn(G_{body},p,q) \wedge BothIn(G_{reuse},p,q) \vee EitherIn(G_{body},p,q) \wedge EitherIn(G_{reuse},p,q) \vee BothOut(G_{reuse},p,q)$ が真）とき，依存関係の強度を変えない（重み w^c 変動なし）．

図 6 (b) に，重み付け操作 Ca, Cb, Cc を示す．C_{whole}, C_{body} は図 3 の WPDG の一部である．網かけは C_{body} 内部の文に対応する節点を指す．

重み w^c は，開発者が部品に対してどのような変更を行ったかに影響を受ける．重み w^c の値が大きいほど，変更前の部品本体 C_{body} において内部と外部に分断していた依存節点を結合する変更を行った割合が多いことを指す．逆に，重み w^c の値が小さいほど，C_{body} において結合していた依存節点を分断する変更を行った割合が多いことを指す．依存節点の接続関係が変化しない，あるいは変更前後の接続関係が不定のとき，重み w^c は変化しない．

このように，重み w^c は，現在（変更前）の部品本体 C_{body} に対して，開発者が依存節点の接続関係を変更した場合にだけ変動する．現在の接続関係が結合のとき，重み w^c は負方向（結合 → 分断）にのみ変動する．逆に，現在の接続関係が分断のとき，重み w^c は正方向（分断 → 結合）にのみ変動する．よって，相反する再利用を繰り返した場合でも，重み w^c は必ず接続関係を変化させる方向にだけ変動し，部品洗練により接続関係が実際に変化するまでの間において，重み w^c が互いに打ち消し合うことはない．重み w^c の絶対値が大きい依存関係矢印は過去に頻繁に変更を受けた部分，その符合は変更の方向（分断 → 結合，あ

るいは，結合 → 分断）を表す．

3.5 重みの計算式

重みの変動定数を δ とすると，重み w^s，w^c の変動操作に対する式は，次のようになる．

(a) 重み増加： $w_n^t = w_{n-1}^t + \delta$

(b) 重み減少： $w_n^t = w_{n-1}^t - \delta$

(c) 変動なし： $w_n^t = w_{n-1}^t$

w^t は，w^s あるいは w^c を指す．また，n は重み w^t の変動回数を指し，$n \geq 1$ の整数である．重みの初期値は，$w_0 = 0$（中心値）に設定する．

本洗練手法では，部品を洗練する際あるいは部品が開発者の再利用ドメインの特性に適しているかどうかを判定する際，重み w^s と重み w^c を組み合わせた指標を用いる．依存節点の接続関係に基づく重み w^s は，過去の部品再利用における依存関係の強度を示している．また，接続関係が変化するときに変動する重み w^c は，部品変更により依存関係矢印の結合あるいは分断が行われた部分に対して，重み w^s の変動を調整する役割を持つ．本手法では，それぞれの重みの変動を δ の加算あるいは減算で達成している．よって，これらの重みを組み合わせた指標は，依存関係矢印の強度を示す主要素 w^s に，変動の調整要素 w^c を加算あるいは減算したものとなる．さらに，本論文では，w^s が依存関係矢印の強度に与える影響の大きさと w^c が依存関係矢印の強度の変動を加速させる大きさを等しいと見なす．以上より，依存関係矢印の強度を表す重み w は，重み w^s と w^c を 1 対 1 の比で加算したものとなる．

$$w = \alpha \times w^s + \beta \times w^c = w^s + w^c \quad (\alpha = \beta = 1)$$

添字のない重み w, w^s, w^c は現在の値を指す．本論文では，$\alpha = 1$ および $\beta = 1$ と設定したが，本来は各ライブラリについて調整が必要である．

変更前後の部品において，依存節点の接続関係と重み w^s および w^c の変動の意味を**表 1** に示す．+ は重み増加，− は重み減少，0 は重み変動なしを表す．表 1 より，重み w^c が増加あるいは減少しているとき，w^s は必ず増加あるいは減少することが分かる．また，w^c は依存節点に関する矢印が結合あるいは分断されたときのみ + あるいは − となる．よって，無変更で依存節点を再利用した場合に比べて，変更をともない依存節点を再利用した場合の重み w の変動は 2 倍になる．本論文における部品洗練の目的は部品変更の負担を軽減することであるので，過去に頻繁に変更された部分に対して，より早く洗練が行われるように重み w の変動を加速することは妥当である．

表 1 重み w^s および w^c の変動値と変動の意味
Table 1 Variations in w^s and w^c and thier summaries.

変更前	変更後	w^s	w^c	変動値	意味
BI	BI	+	0	$+\delta$	無変更で再利用する
EI	BI	+	+	$+2\delta$	結合して再利用する
BO	BI	+	0	$+\delta$	変更は不定，再利用する
BI	EI	−	−	-2δ	分断して再利用しない（する）
EI	EI	−	0	$-\delta$	無変更で再利用しない
BO	EI	−	0	$-\delta$	変更は不定，再利用しない
BI	BO	0	0	0	変更は不定，再利用しない
EI	BO	0	0	0	変更は不定，再利用しない
BO	BO	0	0	0	変更は不定，再利用しない

BI（BothIn）は依存関係矢印が結合，EI（EitherIn）は矢印が分断，BO（BothOut）は矢印の状態が不定を指す．

このように計算した重み w の変動範囲は $-2\delta n \sim +2\delta n$ となり，変動回数 n によって異なる．このため，再利用した回数が異なる部品どうしの重みを単純に比較できない．重みがライブラリ内の全部品に対して共通の指標となるためには，重みの変動が特定の範囲におさまるように正規化する必要がある．本論文では，重みの変動の速度がパラメータによって設定できるという理由から，次の関数[8]を用いて重み w を正規化する．

$$W = N(w) = \frac{1}{1 + e^{(-w/T)}} \quad (T > 0)$$

T は正規化関数のグラフの形を決定する定数である．正規化した重み W の変動範囲は $0 < W < 1$，重みの中心値は初期値 $W_0 = 0.5\,(w_0 = 0)$ に等しい．定数 δ と T は，各ライブラリにおいて調節が必要である．

本洗練手法では，重み w^s と w^c の現在の値を蓄積し，これらを加算し正規化した W を部品 C_{whole} の WPDG G_{whole} の各依存関係矢印に割り付ける．図 3 において，WPDG の各矢印に割り付けられている重みは，正規化後の値 W である．

4. WPDG を用いた部品の洗練

本章では，WPDG の各依存関係の強度に基づき，部品 C_{whole} から強依存関係を持つソースコードを抽出することで，洗練部品を作成する手続きを示す．その際，部品内部の依存関係を維持するコストと洗練により依存関係を修正するコストを定義し，洗練の妥当性を判定する洗練条件を用いる．

4.1 強依存部分グラフの抽出

過去の部品変更履歴に基づき，一定期間重みを蓄積した部品を考える．部品洗練に関する仮定と重みの定義より，開発者の再利用ドメインの特性に適した部品を作成する際，強度が強い依存関係を結合し，強度が弱い依存関係を切り離す．本論文では，洗練後の部品

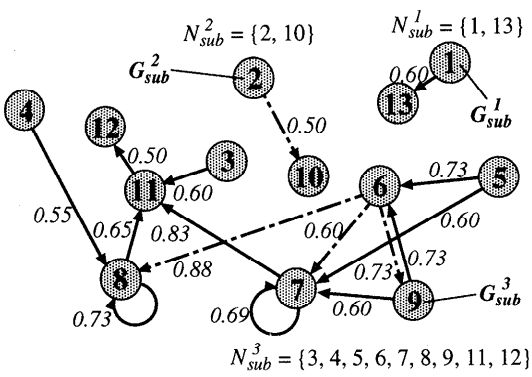


図7 強依存部分グラフと強依存節点集合
Fig. 7 Extracted subgraphs and and node sets.

本体に含む節点を選択する際，依存関係の強度にしきい値 θ を設定し，強結合（重みがしきい値より大）と弱結合（重みがしきい値より小）のしきい値を正規化後の重みの中心値とする（$\theta = W_0 = 0.5$）．

C_{whole} の WPDG G_{whole} において，正規化後の W が θ を超える（$W \geq \theta$）矢印だけを残し，W が θ を超えない（$W < \theta$）矢印を消去する．これにより，強結合を持つ依存節点，つまり強く影響を及ぼし合う節点だけを含み，弱結合を持つ依存節点を分断した複数の部分 WPDG が G_{whole} から抽出できる．抽出した部分 WPDG を強依存部分グラフ G_{sub}，G_{sub} 内に含まれる節点の集合を強依存節点集合 N_{sub} と表す．**図7** に，図3 の WPDG から抽出した3つの G_{sub} と，それらの N_{sub} を示す．

本洗練手法では，N_{sub} 内の節点を洗練後の部品本体 C_{refine} 内部の節点とする．その際，部品本体の機能が単純になることを避けるために，これらの節点を結合する弱結合の矢印を C_{refine} に追加する．洗練後の部品本体の WPDG G_{refine} を次のように定義する．

$$node(G_{refine}) = N_{sub} \quad (\#N_{sub} > 1)$$
$$edge(G_{refine}) = \{p \overset{W}{\rightarrow} q \in edge(G_{whole}) \mid p \in N_{sub} \wedge q \in N_{sub}\}$$

$\#N$ は集合 N の要素数を指す．G_{whole} に存在する N_{sub} 内の依存節点を結合する依存関係矢印は，たとえ依存関係の強度 W がしきい値 θ より小さくても，洗練後の部品本体に必ず現れる．図7において，N^1_{sub}，N^2_{sub}，N^3_{sub} に対する G^1_{refine}，G^2_{refine}，G^3_{refine} が洗練後部品本体 $G_{body'}$ の候補となる．

4.2 重みの誤差による維持コストと修正コスト

依存関係の強度に基づき抽出した G_{refine} が洗練後の部品本体として妥当であるかどうかを判定するために，部品内部の依存関係に対して維持コストと修正コストを定義する．本手法では，これら2つのコストを，依存関係の強度に関する二乗誤差の和（残差平方和）[9]で定義する．二乗誤差の和は，ニューラルネットワークにおける学習[8]などで誤差を見積もる際に用いられ，本手法のように重みの誤差でコストを定義するのに適する．

維持コストとは，部品本体の C_{body} が現在の形を維持するために必要なコストである．このコストは，洗練前の G_{body}，あるいは，洗練後の部品本体の候補 G_{refine} において，各矢印の現在の強度が現在の結合状態を維持する強度とどの程度異なるのかを指す．

いま，$0 < W < 1$ に正規化されている依存関係の強度 W に関して，G_{body} の WPDG 上の各矢印の状態はしきい値 $\theta = 0.5$ を境界として，結合と分断に分かれる．よって，部品内部に含まれる弱結合の矢印に対して，結合するために補う強度は $\theta - W = 0.5 - W$ である．また，部品内外部に分断されている強結合の矢印に対して，分断するために奪う強度は $W - \theta = W - 0.5$ である．本手法では，弱結合に対して補う強度および強結合に対して奪う強度を状態を維持するために必要なコストと見なし，それぞれの強度の二乗誤差の和で維持コストを定義する．C_{body} あるいは C_{refine} の WPDG を G とすると，維持コスト R_p の定義は次のようになる．

$$R_p(G) = \sum_{p \overset{W}{\rightarrow} q \in E1(G)} (0.5 - W)^2 + \sum_{p \overset{W}{\rightarrow} q \in E2(G)} (W - 0.5)^2$$
$$E1(G) = \{p \overset{W}{\rightarrow} q \in edge(G_{whole}) \mid \wedge BothIn(G, p, q) \wedge W < 0.5\}$$
$$E2(G) = \{p \overset{W}{\rightarrow} q \in edge(G_{whole}) \mid \wedge EitherIn(G, p, q) \wedge W > 0.5\}$$

$E1(G)$ は部品内部に含まれる弱結合の矢印の集合，$E2(G)$ は部品内外部に分断されている強結合の矢印の集合を指す．

修正コストとは，C_{whole} において，依存関係矢印の状態を結合および分断する際に必要なコストである．このコストは，各矢印の現在の強度が洗練による結合および分断をいう修正をどの程度妨げるのかを表す．

いま，$0 < W < 1$ に正規化されている依存強度 W に関して，結合している C_{body} 内部の矢印の強度は1（強度最大），分断している C_{body} 内部の矢印の強度は0（強度最小）であることが望ましい．よって，強度 W を持つ依存関係に対して，$|1 - W|$ は結合状態への変化を妨げる強度，$|W - 0|$ は分断状態への変化を妨げる強度である．本手法では，洗練時に結合状態に変化する矢印に対して結合を妨げる強度および洗練時に

分断状態に変化する矢印に対して分断を妨げる強度を修正に必要なコストと見なし，それぞれの強度の二乗誤差の和で修正コストを定義する．C_{body} のWPDGを G，C_{refine} のWPDGを G' とすると，修正コスト R_m の定義は次のようになる．

$$R_m(G,G') = \sum_{p \xrightarrow{W} q \in E1(G,G')} (1-W)^2 + \sum_{p \xrightarrow{W} q \in E2(G,G')} (W-0)^2$$

$$E1(G,G') = \{p \xrightarrow{W} q \in edge(G_{whole} \mid \wedge EitherIn(G,p,q) \wedge BothIn(G',p,q)\}$$

$$E2(G,G') = \{p \xrightarrow{W} q \in edge(G_{whole} \mid \wedge BothIn(G,p,q) \wedge EitherIn(G',p,q)\}$$

$E1(G,G')$ は洗練時に分断状態から結合状態に変化する矢印の集合，$E2(G,G')$ は洗練時に結合状態から分断状態に変化する矢印の集合を指す．

4.3 洗練条件

部品洗練の妥当性を考えたとき，重みが変動するごとに依存関係矢印の結合あるいは分断を行うと，ライブラリ内の部品の形は頻繁に変化することになり，部品検索や部品の機能に対する理解が困難になる．そこで，同様の部品変更が繰り返し行われる場合にのみ洗練が行われるように，維持コストと修正コストに基づく洗練条件を導入する．いま，洗練前の C_{body} が現在の結合状態を維持するのに必要な維持コスト $R_p(G_{body})$ が，洗練後の G_{refine} が結合状態を維持するのに必要な維持コスト $R_p(G_{refine})$ より大きいとき，部品の維持コストが減少するので洗練を行う方がよい．しかし，実際に洗練を行う際には，依存関係矢印の状態を変化させるための修正コスト $R_m(G_{body},G_{refine})$ が必要である．よって，洗練後の維持コストが洗練前より小さく，依存関係矢印の状態を変化させることが可能であるためには，以下の洗練条件を満たす必要がある．

$$R_p(G_{body}) - R_p(G_{refine}) > R_m(G_{body},G_{refine})$$

本手法では，上記の洗練条件が成立する際に洗練が妥当であると判定する．また，複数の候補から洗練後の部品本体を選択するために，部品洗練時の余り R を次のように定義し，R の値が大きいほど開発者の再利用ドメインの特性に適していると見なす．

$$R = R_p(G_{body}) - R_p(G_{refine}) - R_m(G_{body},G_{refine})$$

4.4 部品の洗練手続き

本手法では，次に示す手順で C_{whole} を洗練し，洗練後の部品本体 C'_{body} に含まれる文を決定する．

Step 1. C_{whole} のWPDGの各依存関係矢印の重みに基づき，強依存部分グラフ G_{sub} を抽出し，強依存節点集合 N_{sub} を求める．N_{sub} から，G'_{body} の候補 G_{refine} を求める．G_{refine} が1つしか存在しないとき，Step 5へ．

Step 2. 洗練による急激な形の変化を避けるため，洗練前の G_{body} と洗練後の候補 G_{refine} において，共有する節点の数が多いものを選択する．つまり，集合 $S = node(G_{body}) \cap node(G_{refine})$ に対して，要素数 $\#S$ が最大となる G_{refine} を G'_{body} の候補として選択する．選択した G_{refine} が1つしか存在しないとき，Step 5へ．

Step 3. 余り R が最大となる G_{refine} を G'_{body} の候補として選択する．選択した G_{refine} が1つしか存在しないとき，Step 5へ．

Step 4. ライブラリ管理者，あるいは，開発者が複数の G_{refine} の中から1つを選択する．

Step 5. 洗練条件の真偽を判定する．選択した G_{refine} が洗練条件を満たす場合，G'_{body} に対応する C'_{body} を G_{refine} に対応する C_{refine} に置き換える．洗練条件を満たさない場合，現在の C_{body} をそのまま C'_{body} とする．

Step 1において抽出した図7に示す3つの N_{sub} を洗練の例として取り上げる．Step 1において，N_{sub} から作成した3つの G_{refine} に対して，それぞれもとの部品本体との共有節点の数は，

$$\#S^1_{refine} = 2,\ \#S^2_{refine} = 2,\ \#S^3_{refine} = 5$$

となる．よって，Step 2により，G^3_{refine} が G'_{body} の候補として選択される．**図8**に，G^3_{refine} を用いて洗練を適用する様子を示す．図8(b)において，○印は洗練により結合される矢印，×印は洗練により分断される矢印を指す．洗練前部品の維持コスト $R_p(G_{body})$，洗練候補の維持コスト $R_p(G^3_{refine})$，修正コスト $R_m(G_{body},G^3_{refine})$ は，それぞれ図8に示すようになる．よって，Step 5において洗練条件は次のように成立し，

$$0.8876 - 0.0125 = 0.8751 > 0.6454$$

洗練対象部品 C_{whole} における C_{body} が C^3_{refine} に置換され，洗練後の C'_{body} となる．

図8の例では，図1の C_{whole} において，無変更で頻繁に再利用された，「入力データの合計を求める機能」を担う文3，5，6，7，9と，「入力データの個数を求める機能」を担う文4，8，「入力データの平均を求め表示する」機能を担う文11，12が洗練後の部品本体に含まれる．また，部品変更時に頻繁に削除された，「入力データのファイルに関する処理」を担う文1，2，13，および「合計を表示する機能」を担う文10が部品本体から取り除かれる．

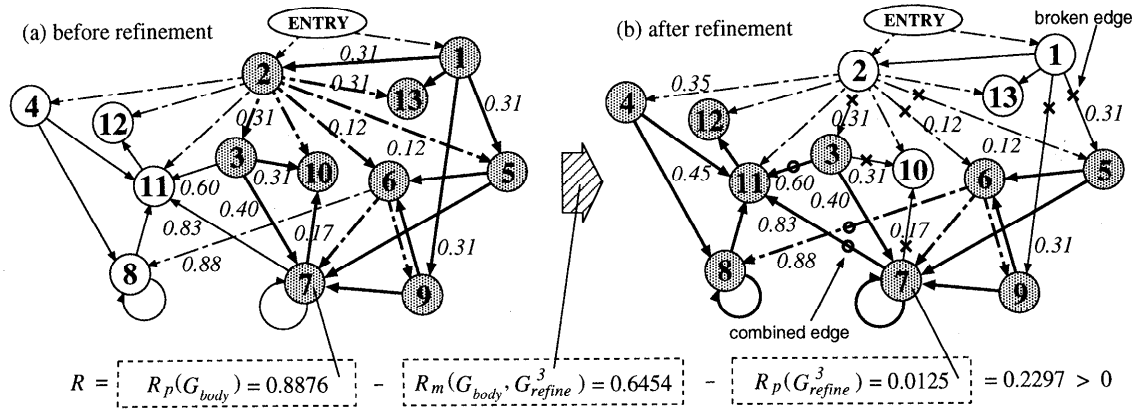


図 8 洗練前後の部品と維持および修正コスト
Fig. 8 Components before and after refinement, and refinement costs.

表 2 洗練対象部品から抽出したソースコード断片の機能と節点
Table 2 Function and nodes of component fragments.

断片	ソースコード断片の機能	節点
f1	ファイルのオープンとクローズ	1 2 13
f2	入力データを読み込む	5 6 9
f3	入力データの合計を表示する	3 5 6 7 9 10
f4	入力データの合計を求める	3 5 6 7 9
f5	入力データの個数を数える	3 5 6 8 9
f6	入力データの平均を表示する	3 4 5 6 7 8 9 11 12
f7	入力データの平均を求める	3 4 5 6 7 8 9 11

5. 評価実験および考察

本洗練手法の有効性を確認するために，本手法に基づいて部品洗練システムを構築し，部品洗練実験を行った．本章では，洗練の特性と洗練の効果という2つの観点から行った実験結果を示し，その実験結果について考察する．

5.1 部品洗練手法の特性

洗練後の部品がどのような形になるのか，および洗練がどのような場合に行われるのかを確認するために，実際に部品を洗練した際の実験結果を用いて，本手法における洗練の特性について考察する．

(1) 同様の変更を適用した際の結果（実験1）

過去の変更に応じて部品が洗練される様子を確認するため，図1に示す部品を用いて，部品洗練に関する実験を行った．本実験では，図1に示す部品から**表2**に示す7つのソースコード断片を抽出し，これらの断片を組み合わせたものを変更後部品 C_{reuse} として用いた．洗練に関するパラメータの値は，$\delta = 1.0$, $T = 5.0$ とした．

本実験において用いた C_{reuse} と，洗練後の C_{body} に含まれる節点を**表3**に示す．表3の C_{reuse}/C_{body} の欄において，記号 + は断片の組合せ（断片に含まれる節点の和集合）を表す．また，付加節点および削除節点の欄の番号は，図1における節点識別番号を指す．初期，最終，洗練後の欄については，付加節点や削除節点ではなく，C_{body} 内部に含まれる節点の識別番号をそのまま記述した．M-1～M-5では，主に「ファイルポインタを入力引数とする（断片f1を削除する）」変更と，「入力データの個数を数える機能を追加する（断片f5を付加する）」変更を適用した．また，M-6～M-10では，主に「入力データの合計を求め，その値は表示しない（節点10を削除する）」形で部品を再利用した．以下，M-1～M-5を変更パターンA，M-6～M-10を変更パターンBと呼ぶ．

表 3 適用した部品変更と洗練後の部品本体 C_{body}（実験1）
Table 3 Fragments and nodes of reused components and refined components.

変更	C_{reuse}/C_{body}	付加節点	削除節点
初期	f1+f2+f3	1 2 3 5 6 7 9 10 13	
M-1	f2+f3	なし	1 2 13
M-2	f2+f4	なし	1 2 10 13
M-3	f2+f5	4 8	1 2 3 7 10 13
M-4	f2+f3+f5+f6	4 8 11 12	1 2 13
M-5	f2+f3+f5+f7	4 8 11	1 2 13
洗練A	f2+f3+f5+f6	3 4 5 6 7 8 9 10 11 12	
M-6	f2+f4	なし	8 10 11 12
M-7	f2+f5	なし	7 10 11 12
M-8	f2+f4+f5+f6	なし	10
M-9	f2+f4+f5+f7	なし	10
洗練B	f2+f4+f5+f6	3 4 5 6 7 8 9 11 12	
M-10	f2+f4+f5+f6	なし	なし
最終	f2+f4b+f5+f6	3 4 5 6 7 8 9 11 12	

本実験では，M-5とM-9の直後に洗練が2回行われる．まず，洗練後の部品の形について考察する．表3において，洗練A直後の C_{body} を C_A，洗練B直後の C_{body} を C_B とおく．また，初期状態の部品本体を $C_{initial}$ とおく．

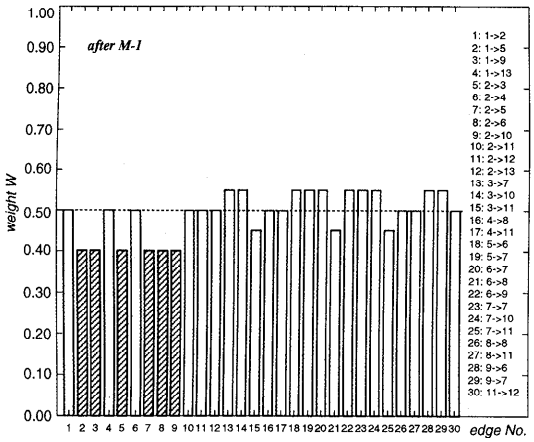


図9 依存関係矢印の重みの値
Fig. 9 Weights at edges in the target component.

表3において，$C_{initial}$ と C_A に含まれる節点を比較すると，洗練Aにより，断片 f4+f6 の節点4，8，11，12が付加（C_A の節点集合 $-$ $C_{initial}$ の節点集合，$-$ は集合の差を表す），断片 f1 の節点1，2，13が削除（$C_{initial}$ の節点集合 $-$ C_A の節点集合）されていることが分かる．この洗練結果は，変更パターンAにおいて，半分以上の割合で節点4，8を付加した，および，すべての変更で節点1，2，13を削除したという変更操作に一致する．すなわち，洗練A前後の C_{body} の変化は，変更パターンAを反映して行われたと考えてよい．また，表3において，C_A と C_B に含まれる節点を比較すると，洗練Bにより，節点10が削除（C_B の節点集合 $-$ C_A の節点集合）されていることが分かる．洗練対象部品において，節点10は入力データの合計値を表示する文であり，この節点が削除されることは合計値の表示が不要，つまり，合計値は平均値を求めるための一時変数として用いられていることを指す．この洗練結果は，変更パターンBにおいて，すべての変更で節点10を削除したという変更操作に一致する．すなわち，洗練B前後の C_{body} の変化は，変更パターン2を反映して行われたと考えてよい．

次に，部品洗練が行われる時期について考察する．M-1の直後の部品 C_{whole} における各依存関係矢印の重みの値を**図9**に示す．また，M-1～M-10を適用した際の，洗練前部品の維持コスト R_p，洗練候補の維持コスト R'_p，修正コスト R_m，余り $R = Rp - R'_p - R_m$ の変動を**図10**に示す．

本洗練手法では，4.3節で定義したコストに基づく洗練条件を導入したことにより，ある節点に関する依存関係矢印の重みがしきい値0.5を超えた，あるいは，しきい値0.5を下まわったからといって，すぐに洗練が行われるわけではない．たとえば，図9を見ると，M-1の適用時の削除節点1，2，13に関する矢印のうち，矢印2，3，5，7，8，9の重みがしきい値0.5より小さい（図9の網掛け部分）．よって，単純に現在の重みに基づいて部品を洗練すると，M-1の直後に節点1，2，13が C_{body} から取り除かれるはずである．しかし，図10を見ると，M-1の直後において R の値は負であり，洗練条件が成立していない．このため，実際にはM-1の直後で洗練は行われない．このように，洗練条件によって洗練の妥当性を判定することで，同様の変更が繰り返し行われた場合にのみ洗練を行うことができ，瞬間的に行われる部品変更に対して C_{body} の形を緩やかに変化させることが可能である．また，図10のM-1～M-5とM-6～M-9において，それぞれ R の勾配を見ると，洗練が行われるまで R は洗練条件が成立する方向（正の値の方向）に単調に増加していることが確認できる．この結果は，本実験において，適用した部品変更操作がそれぞれの変更パターンAあるいはB内で類似していたことを裏付けており，洗練後の部品本体の形が各変更パターンにおいて一意に収束していることを指す．

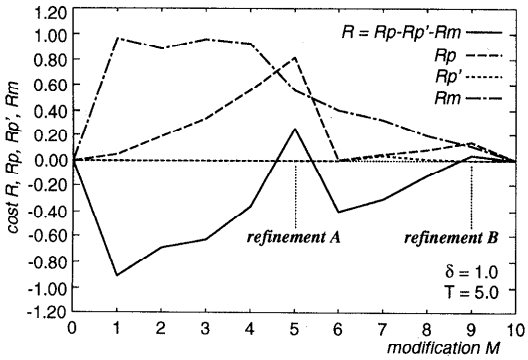


図10 洗練コスト R_p, R'_p, R_m と余り R の変動
Fig. 10 Variation in refinement costs.

以上より，適用した部品変更と洗練に関して次のことがいえる．洗練条件を導入した本洗練手法では，同様の部品変更が繰り返し適用された場合に，その部品変更操作を反映して，緩やかに部品洗練を行うことができる．

(2) 相反する変更を適用した際の結果（実験2）

本実験では，依存節点の接続関係の変化に基づく重み w^c を導入した際の効果の1つとして，相反する変更を繰り返し適用した場合でも部品洗練が行われることを確認する．このために，w^c を導入する場合（$\beta = 1$）としない場合（$\beta = 0$）について，同じ部品を洗練する実験を行った．表3に示すM-1～M-5に

対して，断片f1（節点1，2，13）を付加する変更M–1′～M–5′を適用した部品を用意する．これら相反する変更を含む10個の部品をM–1, M–1′, M–2, M–2′, …, M–5, M–5′の順序で5回繰り返し，全部で50回の重み変動を行った．洗練に関するパラメータの値は，実験1と同じ $\delta = 1.0$，$T = 5.0$ とした．

変更により依存関係が結合と分断を繰り返す矢印7（節点2と節点5を結合）の重みと，余り R の変動の様子を**図11**に示す．図11 (a) を見ると，相反する変更を含む部品を適用することで，矢印7の重みが1回の重み計算ごとに増加と減少を繰り返していることが分かる．$\beta = 1$ のとき，1回ごとの振動の他に大きな周期が見られる．これは，重み w^c の導入により依存関係矢印の強度の変動に偏りが存在することに一致する．このように，相反する変更を適用した場合でも重みを少しずつ増加あるいは減少させることで，部品洗練が行われる可能性は高くなる．このことは，図11 (b) において，$\beta = 1$ の場合，M–15で洗練が行われたことから確認できる．

以上より，重み w^c が洗練に与える影響に関して次のことがいえる．本洗練手法では，相反する変更を繰り返し適用した場合でも，重みの変動に一定方向の偏りが生じ，洗練が行われる可能性は高い．

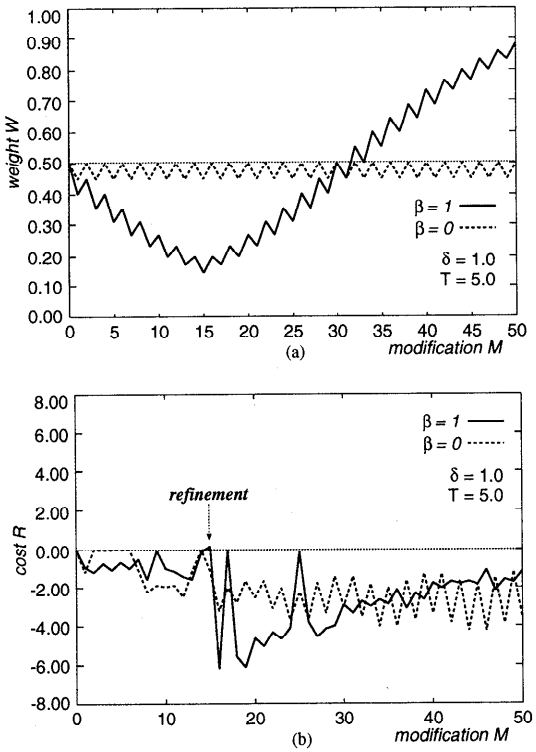


図11 矢印の重みと余り R の変動
Fig. 11 Variations in weight and remaining costs.

(3) 重み付けパラメータの洗練への影響（実験3）

3.5節で定義した重み変動式および正規化関数の式に注目する．n 回の部品変更において，重みが増加した回数を n_i，減少した回数を n_d とすると，$w_0 = 0$ より，

$$w_n = w_0 + n_i\,\delta - n_d\,\delta = (n_i - n_d)\,\delta$$

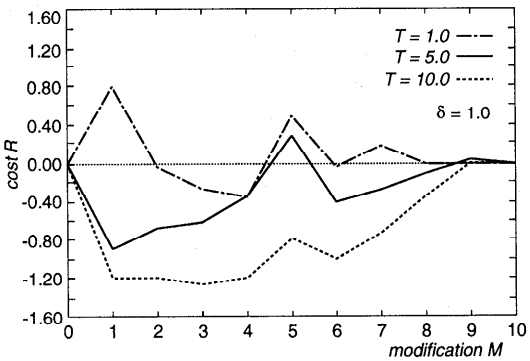
となる．よって，各依存関係矢印に割り当てられる正規化後の重み W は，

$$W = N(w) = \frac{1}{1 + e^{(-(n_i - n_d)\delta/T)}}$$

となる．上記の式より，W の変動はパラメータ δ あるいは T 単独ではなく，これらの比 δ/T に影響を受けることが分かる．さらに，洗練時の余り R の計算式において，R は W^2 の加減算で定義されている．よって，洗練条件が成立する頻度や時期は δ/T に影響を受ける．そこで，重み付けの計算式におけるパラメータが洗練にどのような影響を与えるのかを調べるために，$\delta = 1.0$ と固定し，T の値を変えて，洗練実験を行った．

$T = 1.0, 5.0, 10.0$ において，表3に示す変更M–1～M–10を適用した際の，余り R の変動と洗練後の C_{body} に含まれる節点を**図12**に示す．変更欄の数字は，洗練が行われるまでの変更の回数を指す．$T = 5.0$ のときの R と C_{body} は，実験1のときと同じである．

正規化関数において，T を小さくすると，$w = 0$ 付



変更	節点集合（$T = 1.0$）	節点集合（$T = 10.0$）
初期	1 2 3 5 6 7 9 10 13	1 2 3 5 6 7 9 10 13
1	3 5 6 7 9 10	–
5	3 4 5 6 7 8 9 10 11 12	–
7	3 4 5 6 7 8 9 11 12	–
9	–	3 4 5 6 7 8 9 11 12
最終	3 4 5 6 7 8 9 11 12	3 4 5 6 7 8 9 11 12

図12 $T = 1.0, 5.0, 10.0$ の際の余り R の変動と洗練後部品
Fig. 12 Variation in refinement costs and refined components under $T = 1.0, 5.0, 10.0$.

表4 洗練前部品と洗練後部品の節点および矢印の数
Table 4 Number of nodes and edges in crude and refined components.

部品	FINGER	FINGER′	FTP	FTP′	TELNET	TELNET′	WHOIS	WHOIS′
節点数	21	20	22	16	37	19	26	22
矢印数	48	42	51	32	83	34	62	51

近での傾きが急になり，1回の重み変動操作に対して正規化後の重みの変動が大きくなる．よって，1回の重み変動で洗練条件が満たされる可能性が高くなり，洗練が頻繁に行われると推測できる．逆に，T の値を大きくすると，$w=0$ 付近での傾きが緩やかになる．よって，洗練が行われるために多くの部品変更が必要となり，洗練の頻度は減少すると推測できる．これらの推測が正しいことは，図12において，余り R の変動の形がほぼ等しいにもかかわらず，洗練の回数が $T=1.0$ のとき3回，$T=5.0$ のとき2回，$T=10.0$ のとき1回と，T が大きくなるほど洗練の回数が減少していることから確認できる．特に，$T=1.0$ のとき，実験1の変更パターンAの1回目直後，および，変更パターンBの2回目（全体では7回目）直後という早い段階で洗練が行われている．逆に，$T=10.0$ においては，変更パターンBの4回目（全体では9回目）直後という遅い段階で洗練が行われている．さらに，$T=10.0$ のとき，変更パターンAにおいて，変更を繰り返し行ったにもかかわらず洗練が行われていない．

本実験だけから，δ/T の有効な値を導くことはできないが，重み付けパラメータ δ/T を設定する際の指針として，次のことがいえる．洗練条件を導入した本手法では，洗練の行われる頻度や時期を重み付けパラメータ δ/T の値によって調整可能である．δ/T を小さくすると，洗練の頻度は増加，かつ，洗練の時期は早くなる．逆に，δ/T を大きくすると，洗練の頻度は減少，かつ，洗練の時期は遅くなる．重み付けパラメータによる影響を明確にするためには，さまざまな部品変更パターンを用意し，実験と検証を数多く行う必要がある．

5.2 部品洗練の効果（実験4）

部品洗練の効果を確認するために，実在するソースコードから部品を抽出し，本手法に基づき洗練を行う場合と行わない場合における再利用時の変更操作数を測定した．洗練対象として，UNIX☆の finger，ftp，telnet，whois のプログラムから文献6)の手法により抽出した「TCP ソケットのコネクションを確立する」という共通の機能を持つ部品FINGER，FTP，TELNET，WHOISを用意した．本実験では，これら4つの部品から1つを選択し，残りの3つの部品を変更後の部品と見なして，1つの部品に対して90回（3部品×30回）の変更を適用した．たとえば，部品FINGERを選択した場合，残りのFTP，TELNET，WHOISをFINGERの変更後部品と見なし，FINGER→FTP，FINGER→TELNET，FINGER→WHOISという変更を順番に合計90回適用した．また，洗練後部品の形が変更の順序から受ける影響を少なくするため，1回の重みの変動幅を小さく設定し（$\delta=0.1$，$T=5.0$），数多くの部品変更を行うことで洗練が行われるようにした．洗練は各部品とも1回ずつ行われた．洗練前と洗練後の部品本体の節点数と矢印数を**表4**に示す．FINGER′，FTP′，TELNET′，WHOIS′は洗練後部品を指す．

洗練前および洗練後の各部品を再利用することで，もとの4つのプログラム *finger*，*ftp*，*telnet*，*whois* を作成するのに必要な変更操作数を**表5**に示す．表中のA，D，Iは，2.3節の部品変更における記号を指し，Aは節点の付加，Dは節点の削除，Iは新規節点の追加を表す．たとえば，洗練前のFINGERを再利用して *ftp* を作成するためには，FTP/*finger* の欄のA0 D6 I6より，節点を0個付加，6個削除，6個追加の全部で12操作が必要である．節点の移動Mおよび節点におけるラベルの書き換えRは，節点の削除Dと新規節点の追加Iに分けて数えた．

もとのプログラム *finger*，*ftp*，*telnet*，*whois* を作成する際，洗練前部品を再利用した場合の変更操作数の合計と洗練後部品を再利用した場合の変更操作数の合計を比較する．FINGERとFINGER′の変更操作の合計数はどちらも39回で，本実験から洗練の効果は確認できない．FTP，TELNET，WHOISに関しては，洗練前より洗練後の変更操作の合計数が，それぞれ32%，41%，21%減少している．

さらに，本実験では，「TCP ソケットのコネクションを確立する」機能を持つプログラムとして *GNU finger*☆☆ と *ncftp*☆☆☆ を用意した．これらのプログラム

☆ UNIX は X/Open カンパニリミテッドがライセンスしている米国ならびに他の国における登録商標である．

☆☆ Copyright 1990, 1991, 1992 Free Software Foundation, Inc.

☆☆☆ Copyright 1995 Mike Gleason, NCEMRSoft.

表5 洗練前部品と洗練後部品に対する変更操作の数
Table 5 Number of modified nodes when reusing crude and refined components.

プログラム＼部品	FINGER	FINGER′	FTP	FTP′	TELNET	TELNET′	WHOIS	WHOIS′
finger	− − −	A1 D0 I0	A0 D9 I6	A0 D3 I6	A0 D19 I1	A1 D3 I2	A0 D5 I0	A0 D1 I0
ftp	A0 D6 I6	A0 D6 I7		A6 D0 I0	A0 D24 I6	A2 D4 I5	A0 D10 I5	A0 D6 I4
telnet	A0 D4 I18	A0 D2 I17	A0 D8 I21	A0 D3 I22	− − −	A18 D0 I0	A0 D6 I16	A0 D2 I16
whois	A0 D0 I5	A1 D0 I5	A0 D8 I10	A0 D1 I1	A0 D19 I6	A1 D2 I6	− − −	A4 D0 I0
合計	39	39 (100%)	62	42 (68%)	75	44 (59%)	42	33 (79%)
減少率	−	0%	−	32%	−	41%	−	21%
GNU-finger	A0 D12 I10	A0 D10 I10	A0 D13 I11	A0 D7 I12	A0 D25 I7	A5 D9 I5	A0 D15 I9	A0 D11 I10
合計	22	20 (91%)	24	19 (79%)	32	19 (59%)	24	21 (88%)
減少率	−	9%	−	21%	−	41%	−	12%
ncftp	A0 D10 I14	A0 D8 I14	A0 D7 I10	A0 D3 I13	A0 D28 I12	A0 D22 I13	A0 D14 I13	A0 D10 I13
合計	24	22 (92%)	17	16 (94%)	40	35 (88%)	27	23 (85%)
減少率	−	8%	−	6%	−	12%	−	15%

は，部品FINGER，FTP，TELNET，WHOISの洗練にかかわっていない．これらの部品に対しても，洗練前部品と洗練後部品を再利用した場合を比較すると，どの部品に対しても変更操作数が減少している（6〜41%）ことが確認できる．

表5に示す実験4の結果から，実在するプログラムに対しても，部品洗練により変更操作数が減少することが確認でき，本洗練手法が有効であることが示された．特に，部品の規模（節点数と矢印数）が他の部品より大きいTELNETに関しては，洗練前後の変更操作数の減少率が比較的大きい．本実験の一部の結果において，洗練前部品に対する洗練後部品の変更操作数の減少率がそれほど大きくならなかった（0〜21%）原因として，部品および部品を再利用して作成したソースコードの規模が小さかったことと，洗練に用いた変更パターンの種類が少なかったことが考えられる．

6. 課　　題

本章では，本手法に関する課題について考察する．

6.1 洗練後部品の実行可能性

部品を再利用する際，そのままの形で組み込み可能であることや動作確認ができることより，ライブラリ内の部品は実行可能であることが望ましい．しかし，本洗練手法では，実際に依存関係が存在するにもかかわらず弱結合の依存関係を切断していると見なすので，C_{body} に含まれる文だけを集めても実行可能な部品にはならないことがある．これは，弱結合の矢印を単純に切断すると，C_{whole} 内部に存在しなかった新しいデータ依存関係が生じることや，本来含まれるはずのデータ依存関係が消滅するからである．たとえば，**図13**(a)のように，節点1と節点3にデータ定義順序依存関係（def-order dependence）が存在する場合，節点3の代入文を単純に削除すると，節点1と節点4に新しいデータ依存関係が生じる．また，図13(b)のように，節点1と節点2，3にそれぞれ真と偽の制御依存関係，さらに節点2，3と節点4にデータ依存関係が存在する場合，節点1の条件文を単純に削除すると，節点2と節点4のデータ依存関係が消滅する．

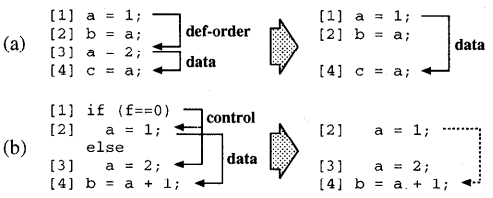


図13 洗練後部品における依存関係の変化
Fig. 13 Change of dependencies in a refined component.

このような問題に対して，洗練後の C_{body} に含まれない節点を無条件に部品から削除するのではなく，削除する節点のラベル（節点に対応する文）を開発者に問合せを行う文に置換することを考えている．問合せ文とは，削除対象節点での代入文において定義される変数の値，あるいは削除対象節点での条件分岐において制御の移る方向を，実行時に開発者が入力する文を指す．

6.2 部品の拡大

本洗練手法では，あらかじめ用意した部品 C_{whole} から C_{body} に含まれる文を選択することで洗練を達成している．よって，C_{whole} に含まれない機能を持つ C_{body} が洗練により生成されることはない．そこで，C_{body} だけでなく，C_{whole} も変更するような仕組みを導入する．我々は，開発者が変更した部品 C_{reuse} のPDGが C_{whole} のPDGを完全に包含するとき，つまり，C_{whole} のPDGの節点および矢印がすべて C_{reuse} のPDGに存在するとき，C_{whole} を C_{reuse} に置換することを考えている．この場合，PDGの包含関係より，置換前の C_{whole} に含まれるすべての機能

が C_{reuse} から抽出可能なので，置換前の C_{whole} における C_{body} はそのままでよく，洗練後の C_{body} の種類を増加できる．

6.3 部品の分裂

本洗練手法は，1つの部品 C_{whole} 内部に含まれる2つ以上の機能を独立に扱うことはできない．このため，機能の関連は少ないが共通な文を含む部品変更を繰り返した場合，過去に再利用したすべての文を含むソースコードが洗練後の C_{body} となり，適切な洗練が行われない可能性がある．この問題に対して，C_{whole} を頻繁に再利用する2つの部分に分裂させ，それぞれの分裂部品に対して独立に重み付けを行うことを考えている．これにより，1つの C_{whole} から2つ以上の洗練後 C_{body} を生成することが可能である．このためには，部品変更のパターンを分類する必要がある．

7. 関連研究

本洗練手法は，もとの部品 C_{whole} のソースコードから，開発者の再利用ドメインに適した機能を持つ一部のソースコードを部品本体 C_{body} として特定することで部品洗練を達成している．よって，プログラム全体を C_{whole} とすると，既存プログラムから再利用ドメインに適した部品を自動抽出することが1回の部品洗練とみなせる．このような手法として，プログラム・スライシング[10]を用い，データの入出力に着目して部品を抽出する手法[4]や，プログラム制御の条件分岐に着目して部品を抽出する手法[11),12]などが提案されている．また，文献3)では，プログラム内部のデータの利用関係や制御の呼び出し関係の事実と推論により強結合の関数を特定し，それらの関数をまとめることで部品を作成する．さらに，文献13)では，もとのプログラムから複数のスライスを抽出し，抽出したスライスに含まれる文どうしの結合の強度を依存関係の存在する割合から算出し，部品に含まれる文を選択する．

このように，プログラムの構造や依存関係に基づき部品を洗練する手法は従来から提唱されている．また，ソフトウェア・リストラクチャリング[14]におけるソースコード変換も洗練手法の1つであると捉えることができる．しかしながら，従来提唱されている部品抽出および再構成手法は，プログラムの静的な解析だけによって達成されている．このため，部品のソースコードが与えられれば，洗練後の部品の形が一意に決まるが，開発者の再利用ドメインの特性に応じて洗練を行っているとはいえない．これに対して，本洗練手法は，過去の変更履歴を反映した依存関係の重みに基づき部品を洗練するため，同じプログラム構造や依存関係を持つ部品を開発者の再利用ドメインにおける利用頻度に応じた形に洗練可能である．さらに，洗練の単位が関数やスライスではなく個々の文であるので，従来手法より粒度の細かい機能変更が達成でき，開発者の部品変更の負担を軽減できる可能性が高い．

また，洗練の指標という観点に着目すると，部品が開発者の再利用ドメインにどの程度適しているのかを評価する手法（メトリクス）も従来から提案されている．たとえば，文献15)では，再利用ドメインにおける標準的な部品から呼び出される頻度を指標として，注目する部品の有効性を見積る．このような手法に対して，本洗練手法では，開発ドメインとの適合度を，開発者が実際に再利用した部品の変更前後のソースコードから計算するため，ドメイン固有の標準的部品をあらかじめ設定しておく必要がない点で有利である．また，本洗練手法に，部品の有効性に関するメトリクスを組み込むことで，本手法の有効性をさらに向上させることが考えられる．

8. おわりに

ソースコード再利用において，過去の部品変更の履歴に基づく重みをPDGの依存関係矢印に割り付け，重みを指標としてライブラリ内の部品を将来再利用される可能性が高い形に自動洗練する手法を提案した．さらに，本洗練手法を用いることで，開発者が部品を変更する負担が減少することを示した．

現在，さらに多くの変更パターンを用意して本手法の適用実験を行い，重み付けパラメータが部品洗練に与える影響の分析を行っている．さらに，6章で述べた課題に対する考察を進めている．

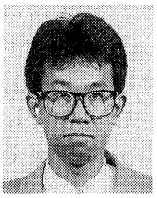
謝辞 日頃ご指導ご討論いただく後藤厚宏リーダーはじめ，グループの皆様に深く感謝します．

参考文献

1) Biggerstaff, T. and Richter, C.: Reusability Framework, Assessment, and Directions, *IEEE Software*, Vol.4, No.2, pp.41–49 (1987).
2) Abd-El-Hafiz, S.K., Basili, V.R. and Caldiera, G.: Towards Automated Support for Extraction of Reusable Components, *Proc. Conf. Softw. Maintenance*, pp.212–219 (1991).
3) Dunn, M.F. and Knight, J.C.: Automating the Detection of Reusable Parts in Existing Software, *IEEE 15th Int. Conf. Softw. Eng.*, pp.381–390 (1993).
4) Ning, J.Q., Engberts, A. and Kozaczynski, W.: Recovering Reusable Components from

Legacy Systems by Program Segmentation, *Proc. Working Conf. Reverse Engineering*, pp.64–82 (1993).
5) Cutillo, F., Fiore, P. and Visaggio, G.: Identification and Extraction of "Domain Independent" Components in Large Programs, *Proc. Working Conf. Reverse Engineering*, pp.83–92 (1993).
6) 丸山勝久，高橋直久：区間設定可能なプログラムスライシングを用いたソフトウェア部品の作成，情報処理学会論文誌，Vol.37, No.4, pp.520–535 (1996).
7) Ferrante, J., Ottenstein, K.J. and Warren, J.D.: The Program Dependence Graph and Its Use in Optimization, *ACM Trans. Prog. Lang. Syst.*, Vol.9, No.3, pp.319–349 (1987).
8) 麻生英樹：ニューラルネットワーク情報処理，産業図書 (1988).
9) 篠崎信雄：統計解析入門，サイエンス社，chapter 11, pp.253–268 (1994).
10) Weiser, M.: Program Slicing, *IEEE Trans. Softw. Eng.*, Vol.10, No.4, pp.352–357 (1984).
11) Lanubile, F. and Visaggio, G.: Function Recovery based on Program Slicing, *Proc. Conf. Softw. Maintenance*, pp.396–404 (1993).
12) Canfora, G., Cimitile, A., Lucia, A.D. and Lucca, A.D.: Software Salvaging Based on Conditions, *Proc. Conf. Softw. Maintenance*, pp.424–433 (1994).
13) Kim, H.S., Chung, I.S. and Kwon, Y.R.: An Approach to Restructuring Programs Through Program Slicing, *Proc. Joint Conf. Softw. Eng.*, pp.307–314 (1993).
14) Arnold, B.S.: Software Restructuring, *Proc. IEEE*, Vol.77, No.4, pp.607–617 (1989).
15) Caldiera, G. and Basili, V.R.: Identifying and Qualifying Reusable Components, *IEEE Computer*, Vol.24, No.2, pp.61–70 (1991).

(平成 9 年 4 月 30 日受付)
(平成 10 年 1 月 16 日採録)



丸山 勝久（正会員）

1967 年生．1991 年早稲田大学理工学部電気工学科卒業．1993 年同大学院理工学研究科修士課程修了．同年日本電信電話株式会社入社．現在，NTT ソフトウェア研究所 広域コンピューティング研究部に所属．ソフトウェア再利用，プログラム変更支援，プログラム自動合成，プログラム解析技術の研究に従事．本学会 1997 年度山下記念研究賞授賞．電子情報通信学会，日本ソフトウェア科学会，IEEE-CS，ACM 各会員．

島 健一（正会員）

1976 年北海道大学工学部電気工学科卒業．1978 年同大学院情報工学専攻修士課程修了．同年 NTT 武蔵野電気通信研究所入所．現在，NTT ソフトウェア研究所主任研究員．主に，知識ベース構築用システムの基礎研究，ソフトウェア設計での知識獲得，学習システムなどの研究開発に従事．また，WWW でのユーザモデル研究に興味を持つ．電子情報通信学会，人工知能学会各会員．