社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

メソッド間の依存関係を利用した再利用支援システムの実装

小堀　一雄† 　山本　哲男†† 　松下　誠† 　井上　克郎†

† 大阪大学 大学院情報科学研究科
†† 立命館大学 情報理工学部情報システム学科
E-mail: †{k-kobori,matusita,inoue}@ist.osaka-u.ac.jp, ††tetsuo@cs.ritsumei.ac.jp

あらまし　プログラム理解や再利用を目的として，開発者はソースコード検索システムなどを利用することが多い．これらのシステムから得られる情報は，あるクラスやメソッドなどといった，単一の部品に関する情報である．しかし，再利用対象となるソフトウェア部品の機能は，その部品のみで実装されていることは稀であり，複数の部品にわたって実装されていることが多い．このため，ある部品を理解するためには，別の部品を繰り返し取得し，その内容を理解する必要があり，これが開発者にとって大きな負担となる．そこで本研究では，メソッド間の依存関係を利用して，あるメソッドが依存する部品群を抽出する手法を定め，その情報を用いて開発者に対してソフトウェア部品の理解支援を行うシステムを提案する．また，本システムをソフトウェア検索システム SPARS-J 上の部品情報表示機能に対する拡張機能として実装する．拡張した SPARS-J を複数のオープンソースプログラムに適用して実験を行った結果，依存部品群の総規模や依存関係のグラフが機能理解の難易を判断する基準になることや，よく利用されていて，かつ複雑な部品はソフトウェア中の主要な機能を司る部品である可能性が高いことなどが分かった．
キーワード　ソフトウェア部品，ソフトウェア再利用，プログラム理解，Java

Implementation of Software Reuse Support System Using Dependency of Method Call

Kazuo KOBORI†, Tetsuo YAMAMOTO††, Makoto MATSUSITA†, and Katsuro INOUE†

† Graduate School of Information Science and Technology, Osaka University
†† Ritsumeikan University
E-mail: †{k-kobori,matusita,inoue}@ist.osaka-u.ac.jp, ††tetsuo@cs.ritsumei.ac.jp

Abstract　In order to understand and reuse program, developers often use research system of source code. Informations that these system output are only about a single compornent (a Class or a Method) of Program. However, a function of program is often implemented in not a single component but multiple components. So, developpers have been forced to search all depended components repeatedly. At first, we suggest a program understanding system using dependency of Java method. We also implement that system on SPARS-J. While, we apply this system to some open source program for validation. As a result, we understand the total scale of dependence part group and dependency graph can be used as a standard by which the difficulty of understanding is judged.
Key words　software component, software reuse, program understanding, Java

1. まえがき

生産効率や品質の向上を目的として，既存のソフトウェア部品を再利用する試みが広く行われている．再利用を行う際には，CASE ツールやソースコード検索システムが利用されることが多い．これらのシステムから得られる情報は，あるクラスやメソッドなどといった，単一の部品に関する情報である．しかし，再利用対象となるソフトウェア部品の機能がその部品に閉じていることは少なく，一般には複数の部品にわたって実装されている．このため，ある部品を理解するためには別の部品を繰り返し取得し，その内容を理解する必要があり，これが開発者にとって大きな負担である．

そこで本研究では，メソッド間の依存関係を利用して，あるメソッドが依存する部品群を抽出する手法を定め，その情報を用いて開発者に対してソフトウェア部品の理解支援を行う手法を提案する．本手法では，事前に蓄積された Java ソースコー

ド群を対象として，含まれるクラスやメソッド間の依存関係を解析し，依存関係グラフを構築する．次に，あるメソッドが与えられた際，そのメソッドが依存する全てのメソッドをグラフから探索して，得られたすべてのメソッドについて依存関係のグラフや，メトリクスの値を表示する．ここでメトリクスとして，LOC やサイクロマチック数などの一般的なメトリクスのほか，依存関係内での呼び出し頻度を示す Component Rank [8] 等を示すことによって，依存関係のあるメソッド内において，構造が複雑すぎるメソッドやよく利用されるメソッドなどの情報を開発者に提示することが可能となる．また，本手法をソフトウェア検索システム SPARS-J [7] 上の部品情報表示機能に対する拡張機能として実装した．拡張した SPARS-J を，複数のオープンソースプログラムに適用して実験を行った結果，依存部品群の総規模や依存関係のグラフが機能理解の難易を判断する基準になることや，よく利用されていて，かつ複雑な部品はソフトウェア中の主要な機能を司る部品である可能性が高いことなどが分かった．

以下，2. では，現状の問題点について考察した後，3. で支援手法を提案する．次に 4. で，システムの実装について述べ，5. でシステムの評価を行った結果を考察し，最後に，6. でまとめと今後の課題について説明する．

2. 背　　景

一般にソフトウェア開発現場では，既存のソフトウェアを保守する際に，プログラム理解や再利用が重要視されている．プログラム理解とは，ソフトウェア資産のソースコードを読み，そのソフトウェア資産が持つ機能がソースコードのどの部分にどのように実装されているかを解読する作業のことである．プログラム理解は，他人が書いたソースコードに対して行う場合，また，ソースコードの作者本人が行う場合であっても，記憶が薄れるにつれて困難なものになる．一方，ソフトウェア資産は，バグが発見されて修正する必要がでたり，また，機能追加を施す必要が現れることがある．そのため，プログラム理解を支援することはソフトウェア開発において重要な要素である．

また，ソフトウェア資産の再利用も近年注目されている．再利用とは，既存のソフトウェア部品を同一システム内や他のシステムで用いることをいい [1]，ソフトウェア生産性と品質を改善し，結果としてコスト削減するという報告が多く出されている [2] [3]．ソフトウェア再利用には，部品の形式や再利用の目的に応じて，**ホワイトボックス再利用** *(white-box reuse)* と**ブラックボックス再利用** *(black-box reuse)* が存在する．

[**ホワイトボックス再利用**]

仕様や文書，プログラムソースコードの再利用を指す．部品の内部構造に基礎を置くため，部品詳細を把握することが可能であり，用途に応じて修正可能でプラットフォームに非依存である．ホワイトボックス再利用における再利用者は主にソフトウェア開発者である．特にソースコード再利用に関しては，ライブラリ内の部品を利用者の再利用環境に応じて洗練する手法に関する研究 [12] などが存在する．

[**ブラックボックス再利用**]

主にバイナリ実行ファイルの再利用を指す．具体的には JavaBeans や ActiveX/DCOM，CORBA などがあり，部品の詳細を知らずにその機能だけを再利用したい時に有効である．プログラムに詳しくないエンドユーザのソフトウェア開発で行なわれるもので，開発形態としてはビジュアルプログラミングなどがある．コンパイル不要で迅速に再利用可能な反面，プラットフォームに依存しており詳細な解析は困難である．

ここで，ブラックボックス再利用とは，上記にもあるように，部品の詳細を知らずに行うことが可能であるように既に設計されているため，再利用を行う段階においては支援の必要性は低い．そこで，本研究が支援する再利用として，ホワイトボックス再利用を対象とする．

ホワイトボックス再利用はソースコードの内部構造や詳細を知る必要があるため結果としてプログラム理解を支援することがホワイトボックス再利用を支援することにつながる．

開発者の立場から考えると，開発者はプログラムが持つなんらかの ”機能 ”を実現したいという目的のためにプログラム理解や再利用を行う．

ここで，開発者を支援する内容は大きく二つにわけることができる．それは,「どのソフトウェア部品（Java では一般的にメソッド）が目的とする“ 機能 ”を持っているのか」を調べることと,「そのソフトウェア部品を理解，再利用するために必要な情報を集める」ことである．

前者の支援内容は既存のソフトウェア検索システムなどで効果的に行うことができる．しかし，後者の支援内容は既存のシステムでは十分とは言いがたい面がある．

というのも，既存のソフトウェア検索システムの多くはデータ表示として，ユーザが指定した単一の部品の情報しか表示しないことが多いため,「目的のソフトウェア部品を理解，再利用するために必要な情報を集める」という問題を解決するには不十分と考えれられるからである．検索者がプログラム理解や再利用を目的としている場合に，目的とする機能が検索結果の部品単独で処理が完結しているような場合は稀であり，現実的には処理が複数の部品に渡って実現されているようなことが多々ある．そのような場合には単一の部品の情報だけでは不十分である．具体的には，あるメソッドが内部で他のメソッドを呼んでいる場合，最初のメソッドが記述されているソースコードを見ただけでは，呼び出し先の処理が全く分からない．

図 1 を例に説明すると，クラス A のメソッド aa は自身の中で処理が完結しているため，メソッド aa 内に記述されている処理のみを理解すれば良い．それに対して，クラス A のメソッド a は内部でクラス B のメソッド b を呼び出しているため，メソッド b を宣言しているソースコードも読む必要がある．そこで，メソッド b を宣言しているソースコードを見ると，今度はメソッド b の中でクラス C のメソッド c を呼び出しているため，さらにメソッド c を宣言しているソースコードを読まないと全体が理解できないことがわかる．そこでさらにメソッド c を宣言しているソースコードを読むと，メソッド c が他のメソッドを呼び出してはおらずに単独で処理を完結しているので，これ以上追跡する必要が無いことがわかり，依存範囲が特定される．

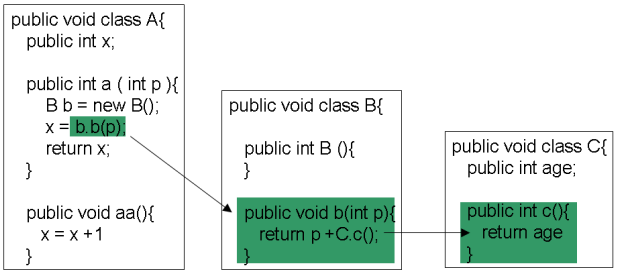

図 1　複数の部品に渡って実装されている処理

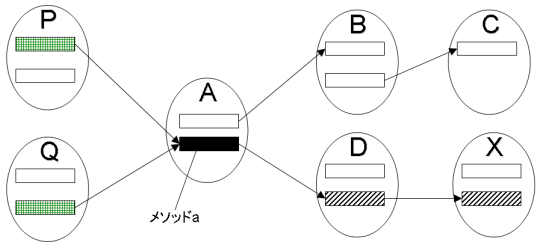

図 2　部品の利用例

メソッド a，メソッド b，メソッド c の 3 つの処理を理解して初めて，メソッド a の処理を完全に理解することができる．

既存のソフトウェア検索システムでは，上記のように，起点となるメソッドから始まる一連の処理が，新しい部品を呼び出さなくなるまで繰り返し検索しないと，その処理がどこまで繋がっているのかということや，処理を理解するためにはどの部品を見ればいいのかというようなことがわからない．このことは，開発者にとって大きな作業コストを生んできた．

3.　メソッド間の依存関係を利用した再利用支援手法

3.1　研究の目的

本研究では，ある部品から始まる一連の機能，処理を理解するのに必要な全ての依存部品を収集，分析して開発者に提示することである．

具体的には，依存部品の総規模を調べて，処理がどこまで繋がっているかの情報を提示したり，依存部品内で特に重要な部品を自動的に解析し，理解支援を行う．

3.2　ソフトウェア部品

Java おいて，１つ１つの具体的な機能を実装しているのは一般的にメソッドである．そこで，本研究では今後特に断り無くソフトウェア部品とはメソッドのことを指し，依存関係とはメソッド呼び出し関係のこととする．

3.3　部品依存グラフ

本システムは多数のソースコードを解析して，メソッドを点とし，依存関係を辺とする図 3 のような部品依存グラフを作成する．

そして検索システムなどで得たれた目的の部品を起点として辺を探索していき，それ以上辺がたどれなくなるまで部品依存グラフを探索して，最終的に得られる部品群をこの部品の依存部品群と判断する．

3.4　依存部品グラフ探索の範囲

依存解析は最悪でもシステムに登録した全部品を依存部品群に入れた時点で止まる．しかし，あるプロジェクトに属するクラスの部品から，JDK の String クラス等のような一般的に

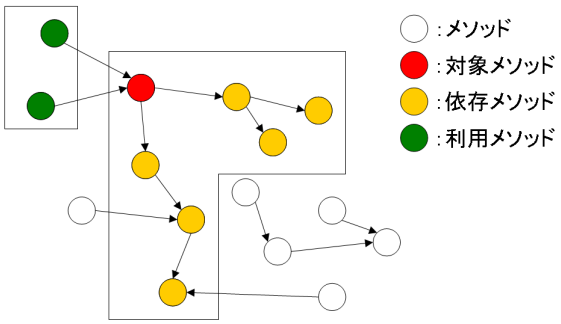

図 3　部品依存グラフ

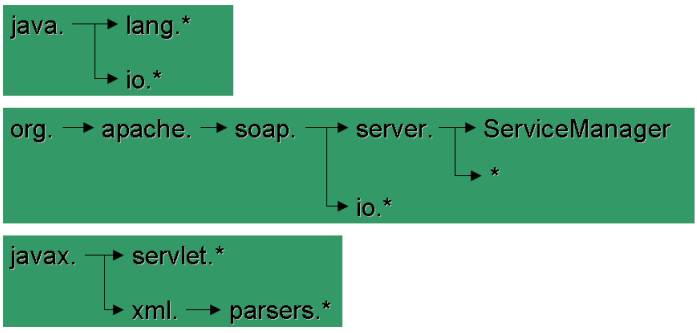

図 4　グラフの探索範囲

utility 的に利用されるメソッドを呼び出していた場合に，その utility メソッドがそれ以後どのような依存関係をもっているかという情報は検索者にとって必要ない情報と推測できる．

そこで，本研究では図 4 のようにパッケージ名を区切り文字（ドット）で区切った先頭の文字列が異なるクラスに属するメソッドを呼んだ場合に依存解析の探索はそのメソッドまでで止めることにする．

例えば，探索の起点なるメソッドが org.apache.soap.server.Servicemanager.main() だった場合に以下の 1，2，3 番のような名前のメソッドはそのメソッド内の依存関係も解析するが，4，5 番はライブラリ的に利用されたと考えて，その内部の解析を

止める．ユーザには表示時に解析を止めた事を知らせる．

- 1 org.apache.soap.client.Client.send()
- 2 org.apache.soap.io.File.write()
- 3 org.apache.struts.digester.Rule.begin()
- 4 javax.xml.parsers.SAXParser.parse()
- 5 java.lang.String.toString()

3.5 再利用部品例の解析

また，目的部品に向かった辺を持つ部品も存在する可能性がある．図 2 で説明すると，メソッド a に向けて辺がひかれている格子状のメソッドはその内部でメソッド a を実際に呼び出して利用している．そのような部品はメソッド a を再利用するような際には有効な利用例としてユーザに提示できる情報である．

4. システムの実装

4.1 システムの概要

記述言語として C 言語および C++言語を，データベースに BerkeleyDB [6] を用いて，システムの実装を行なった．

本システムが持つ機能は以下の通りである．

- 全依存部品の総規模の表示

ユーザが理解支援を要求した部品が依存している全部品の総行数，総サイクロマチック数，総メソッド数を表示し，それが要求部品に対して何パーセント程度の規模なのかを表示する

- 依存部品ツリーの表示

依存部品群が内部でどのような依存関係をもっているかを表示するために，子ノードが，親ノードにメソッド呼び出しをされているという関係を持つ部品ツリーを表示する．ツリーで表示するため，既に依存関係が描かれている部品が再度出現したときには「***」という印をつけて，それ以降の解析情報を表示しない．また，要求部品と先頭パッケージ名が異なるような部品が出てきたときも,「→」という印をつけて，それ以上の解析情報を表示しない．

- CR が非常に高い依存部品の表示

依存部品群の中で，CR(Component Rank) を計算し，その値が上位 30 %に入る部品を表示する．これは，要求部品の処理の中で頻繁に呼ばれる部品を表しており，プログラム理解の上で重要な役割を持つ部品と推測できる．30 %という閾値はこれまでの経験則で設定してある．

- 複雑度が非常に高い依存部品の表示

IBM OS の検証 [9] によると，1 メソッド当たりのサイクロマチック数が 32 を越えるとエラーが潜在している可能性が急に高くなることがわかっている [11]．また，C.Jones [10] は，サイクロマチック数が高くなるにつれてどの程度誤修正をする確率が高くなるかを調査した．そこで，サイクロマチック数が 32 を超える部品にマーク表示をすることで，要求部品の処理の理解や再利用をする上で困難であると予想される部品がどの階層にどの程度存在しているかの理解を支援することができる．また，リファクタリングを目的として本手法を用いた場合にも，バグ潜在などの問題のある可能性が高い部品を探すのに有効な支援情報と考えることができる．

- 依存部品群が実際に記述されているソースコードのダウンロード

依存部品群に属する部品が実際に記述されているソースコードを指定してダウンロードすることができる．この機能は，登録されたソースコードを探す手間を省いたり，依存部品群に属するソースコードセットを他のソースコード解析ツールで解析する際に利用することができる．

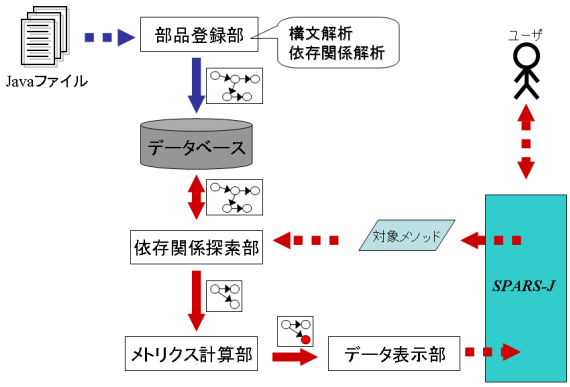

図 5　システムの概要図

4.2 システムの構成

本システムの構成を図 5 に示す．

システムの処理の流れを説明する．まず，Java ソースコードを部品登録部に入力として与える．すると，部品登録部がソースコードを構文解析し，クラスやメソッドなどの部品情報や，メソッド呼び出し等の利用関係情報を取得する．これらから，部品依存グラフを作成し，データベースに登録する．このようなデータベースが用意できた状態で，ユーザが SPARS-J を利用して，理解したい，または再利用したいメソッドを特定し，依存関係探索部に入力として与える．依存関係探索部は，データベースから登録部品に関する部品依存グラフを取得して，入力されたメソッドを起点とする依存部品群を探索し，それらで構成された部分グラフを作成する．作成された部分グラフは，メトリクス計算部に渡され，メトリクス評価を受けて部品群内の重要部品が解析される．重要部品に関する情報を付加された部分グラフは，データ表示部に渡され，データ表示部が部分グラフを人間にわかりやすい形に加工してユーザに提示する．

4.3 部品登録部

部品登録部は，字句解析に Flex [5]，構文解析に Bison [4] を用いて Java ファイルを解析することで，支援情報表示時に必要な情報を抽出し，それらを対応する各 DB に格納する．具体的には，ファイル，クラス，メソッドに関する名前やメトリクス情報，クラス間の利用関係，メソッド間の利用関係などを抽出する．これらの情報から，3.3 節で説明した部品依存グラフを作成する．

4.4 依存部品解析部

依存部品解析部は，部品解析部で作成した部品依存グラフを用いて，理解支援要求部品が自身の処理中に依存している部品を探索し，依存部品群として部品群化するコンポーネントで

ある．

依存部品解析部は，要求部品から始まる利用関係の探索を行い，利用先の部品を依存部品群に追加していく．最終的に依存部品群候補の部品が見つからなくなるまで探索して解析を終了する．なお，この探索は以下の 2 つの条件の下で行われている．

- 一度部品群に追加した部品が他の部品から利用されていたとしても重複して登録しない．
- 先頭パッケージ名が要求部品と異なる部品は，その部品は部品群に追加するが，その部品から先に引かれる利用関係にある部品は探索しない．

前者は無限ループを発生させない為であり，最悪でもシステムに登録されている全部品を依存部品群に登録すれば探索が止まることは保証される．

後者は，先頭パッケージ名が異なる部品を呼び出している場合には，その部品はライブラリ的に利用されていると判断し，本手法ではそれ以降の依存関係は検索者にとって価値が無いと判断する仕様としているためである．

依存部品解析部は，探索を終了した時点での依存部品群と起点となる部品で構成された部分グラフを作成する．

4.5 メトリクス計算部

メトリクス計算部は，各依存部品のメトリクスを評価して，依存部品群内での重要部品を自動的に解析する．

採用するメトリクスは以下の 3 種類である．

- LOC(行数)

部品の規模を表すメトリクス

- CYC(サイクロマチック数)

部品の複雑さを表すメトリクス

- CR(Component Rank) 部品の利用頻度に関する重要度を表すメトリクス

CR は多くの部品から利用されている部品は重要であり，重要な部品から利用されている部品もまた重要であるという基本理論を持つ．

以上の 3 つのメトリクスを評価して，重要部品を解析する．

部品群内で繰り返し呼ばれ，かつ複雑な部品は，その部品群が持つ処理の主要な機能に強い関連をもつ可能性が高いと予想されるので，そのような部品を重要部品として抽出する．

具体的には，CR が高い部品は部品群内で繰り返し呼ばれていることがわかり，LOC/CYC が高い部品は複雑であるため，両方が高い部品を重要部品として抽出する．

このようにして，メトリクス計算部は部分グラフに重要部品に関する情報を付加し，これを最終的な依存部品情報として出力する．

4.6 データ表示部

データ表示部では，メトリクス計算部が出力した情報を人間にわかりやすい形に加工して表示する．

具体的には，図 6 のような CGI を表示する．CGI は上中下の 3 部に分かれており，それぞれ以下のような情報を表示する．

- 上部：利用部品の一覧

部品を利用している利用部品名の一覧を表示する．各部品の名前は，ソースコード中で，その部品が宣言されている箇所へのリンクとなっている．

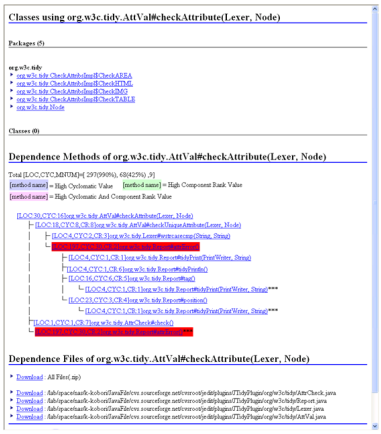

図 6　データ表示部

- 中部：依存部品群に関する情報

依存部品群の総規模に関するメトリクス値と依存ツリーを表示する．総規模に関するメトリクスには，LOC，CYC，依存部品群数の 3 つである．

また，依存ツリーは，根が入力された部品で，節や葉が依存部品であるようなツリーである．各依存部品については，その部品がソースコード中に宣言されている箇所へのリンクやメトリクス情報を表示する．さらに，メトリクス計算部で重要部品と判定された部品に対して赤色でハイライトすることでユーザに提示する．

- 下部：依存部品が宣言されているソースコード

依存部品が実際に宣言されているソースコードのパス情報を表示し，それらをダウンロードすることが可能である．

5. システム評価

5.1 評価概要

まず提案システムのケーススタディーを行う，次にメトリクス計算により自動解析する重要部品の適合率を評価する．

5.2 ケーススタディー

ソートのプログラムを書く必要があり，再利用を目的として SPARS-J に対して“ quicksort ”というキーワードで検索を行い，以下の 2 つの再利用候補部品を得たとする．このとき，どちらが再利用部品として理解し易いかを判断する支援を行う例を以下に示す．

部品 1 :org.apache.turbine.util.QuickSort.quickSort(Object, int, int, Comparable)

部品 2 ：cz.dhl.io.CoSort.QuickSort(CoFile, int, int)

それぞれ，単独の規模は表 1 のようになっており，サイクロマチック数は同程度で，行数が部品 1 のほうが大きいことから，部品 1 のほうが大規模であることがわかる．

しかし，依存部品群の総規模は，表 2 のようになっており，全体としては，逆に部品 2 のほうが大規模であり，また，実際にこれら 2 つの部品を理解するのにかかった時間も部品 2 のほ

候補部品	LOC	CYC
部品 1	74	11
部品 2	47	12

表 1　単体の規模 (LOC：行数，CYC：サイクロマチック数)

候補部品	総 LOC	総 CYC	理解に要した時間 (sec)
部品 1	76	12	44
部品 2	94	31	169

表 2　依存部品群全体の規模

ソフトウェア名	依存部品数	重要部品数 (システム)	重要部品数 (人間)	適合率
JEdit	237	16	10	0.625
JEditLogger	7	1	1	1.000
JGraph	49	1	1	1.000
Astyle	92	5	4	0.800
Tidy	35	2	2	1.000
VncViewer	102	11	7	0.636
GanttProject	163	12	11	0.917
FontChooser	3	1	0	0.000
Parser	94	14	11	0.786
ClassFinder	8	1	1	1.000
	790	64	48	0.750

表 3　適　合　率

うが大きいという結果が得られた．

これらの情報から，部品 1 のほうが再利用候補として処理を理解し易いと判断する等の利用の仕方が考えられる．

このように，同種の機能を持つ複数の再利用候補部品があった場合に，それらの総依存部品の規模を基準として再利用候補を取捨選択するための理解支援が可能であると考える．

5.3　適　合　率

無作為に選んだ 10 個のソフトウェアの main メソッドに対して本手法を適用し，高 CR，高 CYC のマークが表示されているメソッドがそのソフトウェアの機能の一部を担っているかどうかの適合率を求めた．

機能の一部を担っているかどうかは，私がソースコードやドキュメント・コメントを読んで判断した．

採用したソフトウェアと適合率を表 3 に示す．

5.4　結果に関する考察

依存部品群の総規模に関しては，同種の機能をもつ複数の再利用候補部品があった際に再利用にかかる時間の指標として有効であることがわかった．今後の課題として，依存部品群の総規模でソートしたランキングを表示をするなどの理解支援も有効と考えている．

また，高 CR，高 CYC な部品が主要な部品であるかどうかの適合率に関しては，表 3 を見る限りでは 0.750 という高い値が出たため，本指標は主要メソッドを探すのに有効なものであると考える．ただし，8 番目のソフトウェア“ FontChooser ”などの，単純な構造を持つ依存部品郡に対しては，適合率が低い結果が出た．これは，高 CR の閾値を上位 30 %にしているのと，単純な構造では CR が有効に機能しないことなどが理由として考えられる．そこで，単純な構造をもつものに関しては，別の判断基準が必要になると考えるが，その一方で，このような単純な構造を持つメソッドに対しては，ソフトウェア理解支援の必要性もまた低いと考えられるので，特に大きな問題ではないと考える．

6. ま　と　め

本研究では，Java メソッド間の依存関係を利用したプログラム理解支援手法を提案し，SPARS-J のデータ表示部として実現した．本手法を用いることで，開発者はソフトウェア検索システムの検索結果を得たと同時に，再利用候補部品の選択条件として，また選択後の理解支援情報として，全依存部品群に関する規模や性質などの情報を得ることが可能となる．更に，適用実験によって，各支援情報の利用の仕方やその有効性を示した．今後の課題として以下が挙げられる．

- 主要メソッド判定の閾値の調整

より多くのソフトウェアに対して本手法を適用し，最も高い適合率が得られる閾値を調べる．

- 適合判断の考察

現状では，私が主要メソッドの適合性を判定をしているが，対象ソフトウェアのプログラマが行うことで，より精度の高い評価が可能となる．

文　　献

[1] C. Braun: “Reuse, in John J. Marciniak, editor”, *Encyclopedia of Software Engineering*, Vol. 2, John Wiley & Sons, pp. 1055-1069 (1994).

[2] V. R. Basili, G. Caldiera, F. McGarry, R. Pajerski, G. Page and S. Waligora: “The software engineering laboratory - an operational software experience”, *Proc. of ICSE14*, pp. 370-381 (1992).

[3] S. Isoda: “Experience report on a software reuse project: Its structure, activities, and statistical results”, *Proc. of ICSE14*, pp. 320-326 (1992).

[4] Free Software Foundation, Inc.,“Bison”,http://www.gnu.org/software/bison/.

[5] Free Software Foundation, Inc., “Flex”, http://www.gnu.org/software/flex/.

[6] Sleepycat Software, Inc., “BerkeleyDB”, http://www.sleepycat.com/.

[7] 横森, 梅森, 西, 山本, 松下, 楠本, 井上: “Java ソフトウェア部品検索システム SPARS-J”, 電子情報通信学会論文誌 D-I, Vol. J87-D-I, No. 12, pp. 1060-1068, (2004)

[8] 横森 励士, 藤原 晃, 山本 哲男, 松下 誠, 楠本 真二, 井上 克郎: ”利用実績に基づくソフトウェア部品重要度評価システム”, 電子情報通信学会論文誌 D-I, Vol.J86-D-I, No.9，pp.671–681, 2003.

[9] 大場充: “ソフトウェア・プロジェクトの実績データ収集・分析技法”, ソフトリサーチセンター (1993)

[10] C.Jones: “Applied Software Measurement Second Edition”, McGraw-Hill ソフトウェア開発の定量化法共立出版 (1991)

[11] W.S: “Humphrey Managing the Software Process”, Addison-Wesley ソフトウェアプロセス成熟度の改善日科技連 (1989)

[12] K. Maruyama, K. Shima: “A Mechanism for Automatically and Dynamically Changing Software Components”, Symposium on Software Reusability, ACM Software Engineering Notes, Vol. 22, No. 3, pp. 169-180, (1997).