修士学位論文

題目

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の提案と実現

指導教官
井上 克郎 教授

報告者
小堀 一雄

平成 17 年 2 月 14 日

大阪大学 大学院情報科学研究科
コンピュータサイエンス専攻 ソフトウェア工学講座

平成 16 年度 修士学位論文

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の提案と実現

小堀 一雄

内容梗概

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．しかし，これら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，別の部品を繰り返し取得し，その内容を理解する手間がかかり，これが開発者にとって大きな負担であった．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利用関係を解析し，部品間利用関係グラフを構築する．次に，メソッドが与えられた際，依存関係のあるすべ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 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 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．

主な用語

ソフトウェア部品 (Software Component)
ソフトウェア再利用 (Software Reuse)
プログラム理解 (Program Understanding)
Java

目 次

1 まえがき **5**

2 背景 **6**

2.1 プログラム理解と再利用 6

2.2 SPARS-J 7

2.2.1 SPARS-J の概要 7

2.2.2 SPARS-J におけるソフトウェア部品 7

2.2.3 利用関係 8

2.2.4 類似部品の部品群化 9

2.2.5 順位付け手法 12

2.2.6 システムの実装 18

2.2.7 システムの機能 19

2.2.8 システムの構成 20

2.2.9 データベースの説明 23

2.2.10 部品登録部の説明 24

2.2.11 部品検索部 25

2.2.12 データ表示部 26

2.3 SPARS-J の効果と問題点 26

2.4 関連研究 30

3 メソッド間の依存関係を利用したプログラム理解手法 **31**

3.1 プログラム理解支援手法の概要 31

3.2 ソフトウェア部品 31

3.3 依存解析の粒度 31

3.4 依存関係 32

3.5 再利用部品例の解析 32

3.6 部品依存グラフ 33

3.7 依存部品グラフ探索の範囲 33

3.8 解析対象 35

3.9 解析精度 35

4 提案手法の実現 **36**

4.1 システムの概要 36

4.2 システムの構成 37
4.3 データベース 37
4.4 部品登録部 39
4.5 依存部品解析部 40
4.6 データ表示部 40
4.6.1 利用部品表示部 41
4.6.2 依存部品群の特徴メトリクス表示部 41
4.6.3 依存部品群ツリー表示部 41
4.6.4 ダウンロード部 42

5 評価 **44**
5.1 評価概要 44
5.2 ケーススタディー 44
5.2.1 適用例 1 44
5.2.2 適用例 2 45
5.3 適合率 46
5.4 結果に関する考察 46

6 まとめ **49**

謝辞 **50**

参考文献 **51**

1　まえがき

近年のソフトウェア開発現場では，ソフトウェアの大規模化，複雑化によって，生産効率や品質の向上を目的とした，既存のソフトウェア部品の再利用や保守が頻繁に行われている．その際，既存の CASE ツール，特にソースコード検索システムなどで得られる最終的な支援情報はある単一の部品（１つのクラス，メソッド）に関する情報であることが多い．ここで，開発者が再利用したい機能が，その部品単独で実装されている場合は問題無いが，現実には，機能は複数の部品に渡って実装されている場合が多々ある．そのような場合，開発者は内部で依存している部品に関する情報を改めて CASE ツール等で支援情報を再帰的に取得する必要がある．そして，最終的にどの程度の規模の依存部品群があるのかという情報は，依存部品が出尽くすまで繰り返し情報を取得して初めて開発者が把握できるという状況だった。そのため，開発者は全体の規模を把握することなく，再利用部品を選択することも多く，再利用を進める中で，予想外の規模に気づき，再利用を諦めることも多々あった。

このような問題を解決するには，部品単独だけでなく，その部品が内部で依存している全部品についての理解支援情報を，再利用部品を選択する前に開発者に提示することが必要となる．

そこで，本論文では，まずソースコードを登録し，そこに含まれるクラス，メソッド間の利用関係を解析して部品間利用関係グラフを構築する．その後，開発者が理解支援したいメソッドを指定すれば，そのメソッド，およびそのメソッドが内部で依存している全メソッドに関する情報（規模や範囲，依存構造，ソースコードファイル）を取得できるシステムを構築する．開発者に提示する情報としては，LOC, サイクロマチック数，その依存関係内での呼び出し頻度を表す Component Rank や，階層深くに存在している重要メソッドや複雑すぎるメソッドに対する注意情報などがある．

さらに，本システムの評価実験として多数のオープンプロジェクトのソースコードを実際に登録，解析し，任意のメソッドに関する依存情報を表示，分析を行う．そして，得られたメソッド群や，その中の依存構造がどのような性質をもっているかを考察する．

なお，本システムは，過去のソフトウェア資産を保守したり，一部変更して再利用する等の際にプログラム理解支援システムとして利用されることを想定している．

以降，2 節で本システムが必要とされる状況について述べ，関連研究の紹介を行なう．3 節で本手法について述べ，4 節でシステムの構成について説明する．5 節では本システムの性能評価を行なう．最後に 6 節で本論文のまとめと今後の課題について述べる．

2　背景

本節ではプログラム理解と再利用について説明した後，プログラム理解支援システムの概要と，既存の研究について述べる．

2.1　プログラム理解と再利用

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 [8]，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 [5][16]．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 [23] 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．

一方，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．

１つめ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，２つめ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．

このことを，ソフトウェア検索システムの１つである SPARS-J[32] を例にとって説明する．

2.2 SPARS-J

本節では，本研究で構築する SPARS-J の概要と，SPARS-J で用いている主な技術について述べる．

2.2.1 SPARS-J の概要

SPARS-J*(Software Product Archive, analysis and Retrieval System for Java)* は，Java 言語 [18] で記述されたプログラムを対象としたソフトウェア部品検索システムである．Java の利用者が指定したキーワードと関連する部品を検索する．検索結果はキーワードの出現頻度と，利用関係に基づくソフトウェア部品重要度評価手法 [15] によって順位付けされる．さらに，検索結果に併せて部品間の利用関係や類似部品に関する情報を提供することで，再利用やプログラム理解・保守を行なうことが可能となる．

2.2.2 SPARS-J におけるソフトウェア部品

SPARS-J では Java の一つのクラス・インタフェースのソースコードを部品として扱う．Java はオブジェクト指向に基づいた言語であり，クラス・インタフェースはソフトウェア部品の概念を満たす．SPARS-J はプログラム理解，保守といったより広範な利用目的を想定している．そのため，部品の詳細を利用者が把握可能なソースコードという形態が適切だと考えた．その他ソースコードの配布により，利用者の目的に応じたカスタマイズが可能であるという利点が存在する．同時に，単に部品を再利用したいという場合にはクラスや Java に関

する知識が必要となるが，後述する様々に提供する部品の詳細情報を提供することでその欠点も補うことができると考えている．部品の索引は索引キーの集合から成り立つ．索引キーは Java ソースコード中に現れる予約語以外のキーワードとトークン種類の組である．また，索引キーに用いられるキーワードのことを特に索引語と呼ぶ．同様に，検索時に用いられるキーワードを検索語と呼び，検索語と検索対象のトークン種類の組を検索キーと呼ぶ．

2.2.3 利用関係

ソフトウェアは複数の部品の集合体として構成される．これらの部品は互いに独立であって，お互いに影響を及ぼさず部品単位で代替可能である．ソフトウェアはそれら独立な部品間でメッセージをやり取りし協調作業を行うことで一つの大きな機能を提供する．Java のクラスを部品として捕らえると，他クラスのフィールド参照やメソッド呼出しなどがメッセージに相当する．このメッセージのやり取りはそれぞれの部品の属性や振る舞いを利用するために行われるが，そのようにある部品がある部品を利用する際，部品間に**利用関係** *(Use Relation)* が存在すると呼ぶ．利用関係の定義は部品の種類によって異なる．UML であれば依存，汎化などの関係が利用関係にあたるであろうし，バイナリ実行ファイルであれば上述のメッセージを利用関係と見なせるかもしれないが，ここではそのような議論はしない．

本論文では Java のクラス・インタフェースのソースコードを部品として扱うため，Java におけるクラス・インタフェース間の関係を利用関係として抽出する．Java プログラムのソースコード集合について，静的依存解析し依存関係をもとめる．SPARS-J では，以下の 7 種類の依存関係を利用関係と定めている．

1. 継承 : 子クラスが親クラスを利用.
2. 抽象クラス実装 : 実装クラスが抽象クラスを利用.
3. インタフェース実装 : 実装クラスがインタフェースを利用.
4. 変数宣言の型 : 変数宣言したクラスが変数型を定義するクラスを利用.
5. インスタンス生成 : インスタンス化したクラスが生成したクラスを利用.
6. フィールド参照 : 参照元クラスがフィールドを定義するクラスを利用.
7. メソッド呼出し : 呼出し元クラスがメソッドを定義するクラスを利用.

上記の内，実行時に動的に決定される利用関係については静的解析のみで一意に特定する事は不可能である．しかし，全てのソフトウェア部品について実行を行う動的依存解析の

は多くのコストがかかり，実行する際に入力が必要な場合もあるので自動化することが難しい．そのため，静的依存解析を行い宣言されている部品に対して利用関係を抽出する．

そして，抽出した利用関係をもとに，**部品グラフ** *(Comopnent Graph)* を構築する．部品グラフは部品間の利用関係をグラフ上に表現したもので，各部品を頂点とし，部品間の利用関係を利用する側からされる側への有向辺で表す．以下では，V を部品（頂点）の集合，E を有向辺の集合として，$G = (V, E)$ と表現する．図 1 は二つのソフトウェアシステム X と Y を部品グラフ化したものである．X は a から e の 5 つ，Y は f から i の 4 つの部品で構成されており，部品 c は 部品 a および b を利用し，部品 d および e は 部品 c を利用している．同様に部品 h と i は 部品 g を利用し, 部品 g と f はお互いに利用しあっている．

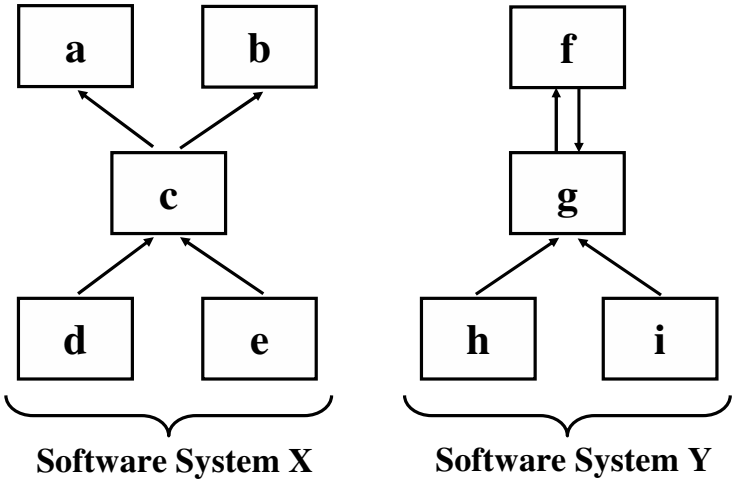

図 1: 部品グラフの例

2.2.4 類似部品の部品群化

似たような機能を提供する部品を類似部品という．例えば，コピーした部品や，コピーして一部を変更しただけの部品は類似している．ライブラリが様々なソフトウェアから構成される場合，その中にはコピーされたりコピーされた上で大小様々な変更が加えられた部品が少なからず存在し，ライブラリの規模が大きくなるほど類似部品は多くなる．

異なるソフトウェアをまたいで類似部品が現れる場合を考える．もし図 1 で部品 c と g が類似部品であったとき，ソフトウェア X と Y は似たような機能を利用していると推測できる．そのため，利用関係は部品 c と g をまとめたクラスタに対して定義するのが妥当である

と考えられる．そこで，提案手法では似た部品をクラスタ化する．このクラスタを部品群，クラスタ化することを部品群化と呼ぶ．そして，ある部品群に属する部品が他の部品群に属する部品を利用している場合には，その 2 つの部品群間に利用関係が存在するとみなしている．図 2 に部品群化によって得られた部品群グラフを示す．

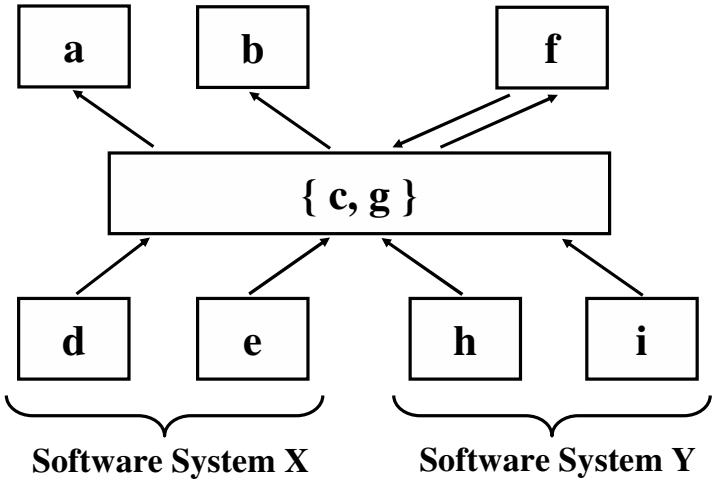

図 2: 部品群グラフの例

SPARS-J の類似度判定は [22] で提案された手法を用いている．[22] では，Java ソースコードから静的解析によって各クラスの静的特性メトリクスを抽出する．そして，それらのメトリクス値からクラスの類似度を判定する．

- 構成トークン類似度メトリクス
  プログラムの表層的特徴の一つとして，トークンの種類別出現頻度が考えられる．Java における全トークン，すなわち予約語，演算子，記号，識別子，合計して全 96 種類のトークンについて，それぞれ種類別出現頻度をメトリクスとして構成トークン類似度を分析する．プログラムを構成しているトークンの数と種類が良く似ているプログラム同士は類似度が高いとみなす．

- 複雑性類似度メトリクス
  プログラムの構造的複雑さの特徴を表すために，表 1 に示すメトリクスを，ソースコードの複雑さの指標として記録する．さらにそれぞれのメトリクスに対して，類似判定に関する閾値を設定する．

メトリクス名	説明	閾値
cyclomatic	サイクロマチック数	0
declamethodnbr	メソッド宣言の数	1
callmethodnbr	メソッド呼出しの数	2
nestingdepth	ネストの深さ	0
NOclass	クラスの宣言数	0
NOinterface	インタフェースの宣言数	0

表 1: 複雑性類似度メトリクス

部品 P, Q の類似判定方法について説明する．まず，各複雑性類似度メトリクスの差分が表 1 で設定した閾値以内に収まっていれば部品 P, G は複雑性類似度において類似であると判定する．

続いて，構成トークン類似度の判定を行なう．部品 P をそれを構成する 96 種類のトークンの出現頻度によって $P = \{p_1, \cdots, p_m\}$ と表現する．各 p_i は P における各トークンの出現数である．$P = \{p_1, \cdots, p_{96}\}, Q = \{q_1, \cdots, q_{96}\}$ としたとき，P，Q の類似判定は次のように行なう．P の全トークン数を $Ttotal(P)$，P と Q の各トークン数の差分の和を $\mathit{diff}\,(P, Q)$ とすると次式のようになる．

$$Ttotal(P) \equiv \sum_{k=1}^{96} p_k$$

$$\mathit{diff}\,(P, Q) \equiv \sum_{k=1}^{96} |p_k - q_k|$$

$\mathit{diff}\,(P, Q)$ が 0 なら，全く同じ種類のトークンを，全く同じ回数用いてプログラムを構成していることになるので，コピー部品であると推測できる．また，$\mathit{diff}\,(P, Q)$ が同じ 30 トークンだとしても，全トークンが 40 トークン中の差分が 30 トークンであるのと，10000 トークン中に差分が 30 トークンあるのとでは，その性質が全く違うため，非類似度を求める際には全トークン数で正規化した値を求める．部品 P と 部品 Q の非類似度 $D(P, Q)$ を次のように定義する．

$$D(P, Q) \equiv \frac{\mathit{diff}\,(P, Q)}{min(Ttotal(P), Ttotal(Q))}$$

ここで，$D(P, Q) < 0.03$，つまり部品 P, Q の各要素の差分の合計が全体の 3% 以下であるものを構成トークン類似度において類似とみなす．この値は経験的な値であり，ライブラリの規模やライブラリ管理者の意図によって随時変更されるべきものである．

以上，複雑性類似度メトリクス，構成トークンメトリクスの類似判定のどちらにおいても類似であれば部品 P, Q は類似部品であると判定する．

2.2.5 順位付け手法

部品検索をキーワードによって行なうと，検索キーと合致する索引キーを持つ部品のが多数存在する場合がある．そのため，検索結果を提示する際に適当な順位で表示することが必要となる．自然言語文書検索システムで一般的に用いられている手法では，検索対象となる文書集合から各文書の特徴を表すキーワードである索引語を抽出し，索引語の集合によってその文書の内容を近似する．登録するそれぞれの文書の特徴を的確に表すように付与された，索引語の集合などからなる情報のことを索引キーと呼ぶ．検索キーは検索者の要求の内容を近似しているため，検索キーと索引キーを用いて検索キーと文書の適合度を測り，順位付けするのが一般的である．

しかし，ソフトウェア部品を対象としてを検索する場合は，検索者の要求の内容と適合する部品であるかどうかの他，その部品がよく利用されている部品かどうかに関しても考慮する必要がある．検索キーと適合度が高い部品よりも，よく利用されている部品を上位に提示した方が，利用例も多く，部品の再利用をスムーズに行ない易いと考えられる．そのため，利用しやすい部品かどうか定量的に評価するための指標を導入し，検索キーと部品の適合度も併せて両面を考慮した部品の順位付けを行なう必要がある．

Keyword Rank 法

索引キーの中には部品の内容と密接に関係したものもあれば，関係の薄いものも存在する．抽出された索引キーが部品の内容を表すうえでどれだけの重要度を持っているか測ることができれば，より望ましく順位付けされた検索結果が提供できる．このために用いられるのが索引キーの重み付けである．索引キーの重みを利用することによって，同じ索引キーを含む部品でもその索引キーの各部品中での重要度を考慮して，検索キーに対する部品の適合度を計算し部品を順序付けすることが可能になる．検索者が与えた検索キーがある部品の索引キーにヒットしたとき，その索引キーの重みの総和を検索キーと部品の適合度とする．SPARS-J における索引キーの重み付け手法として，情報検索の分野で一般的に用いられる TF-IDF 法 [21] を用いる．TF-IDF 法は任意の部品中における特定の索引キーの出現頻度 TF(*Term Frequency*)，および特定の索引キーを含む部品数の逆数 IDF(*Inverse Document Frequency*) の値を正規化して重みを算出する．TF は部品内で出現頻度の低い索引キーと高い索引キーを差別化し，部品をより特徴付ける語を選別するためのものである．IDF は部品集合内の他の部品の索引キーの分布について考慮するためのもので，ある索引キーが，どの程度その部品に特徴的に現れるのかという特定性を示す．検索キーと部品の適合度の高い順

トークン種類	重み	トークン種類	重み
クラス定義名	200	メソッド定義名	200
インタフェース名	50	パッケージ名	50
インポートパッケージ名	30	呼出しメソッド名	10
参照フィールド変数名	10	生成したクラス名	10
変数の型名	10	参照する変数名	1
コメント (/*...*/)	30	文書コメント (/**...*/)	50
行末コメント (//...)	10	文字列リテラル	1

表 2: トークン種類とその重み

に順位付けすることを，後述する CR 法に対して ***KR* 法 *(Keyword Rank* 法*)*** と呼ぶことにする．また，測定した適合度の値を KR 値と呼ぶ.

SPARS-J では，索引キーの重みを算出する際に，そのトークンの種類によって異なる重みを与えている．表 2 に区別する索引語のトークン種類と重み付けの例を示す．どの種類の索引語にどの程度の重みを与えると良い結果が得られるかは知られていないが，経験的にクラス定義名やメソッド定義名など，その部品の概念を象徴した名前が付けられる傾向の索引キーに対して大きな重みを設定している．

データベース中の総部品数を N，部品を c，与える m 個のキーワードを $t_1, \ldots, t_m$，検索対象としたトークン種類の集合を $subject$，c 中の種類 s の索引語 t の出現回数を $tf_s(t, c)$，集合 $subject$ 中の任意のトークン種類の索引語 t を含む部品の数を $df_subject(t)$ と表す．さらに，$kw(s)$ は索引語の種類 s の重みを指し，その値は表 2 に従う．例えば，kw(クラス定義名) は 200 である．そのとき部品 c の KR 値は以下の式で求める．

$$KR = \sum_{i=1}^{m} (log_e(\sum_{s \in subject} kw(s) \cdot tf_s(t_i, c)) \cdot \frac{N}{df_subject(t_i)})$$

Component Rank 法

これまでに，利用関係からソフトウェア部品の利用関係を測定し，順位付けする手法 ***(Component Rank* 法，*CR* 法*)***[15] が提案されている．CR 法では，十分な時間が経過し利用関係が収束した部品の集合に対して，各部品間に存在する利用関係に基づいてグラフおよび行列を構築し，構築された行列に対して繰り返し計算を行う事で各部品を評価する．求められる値は，開発者が利用関係に沿って参照を行うと仮定した場合の各部品の参照されやすさを表しており，よく利用される部品や，重要な部品から利用される部品の順位は高くなる．CR 法によって測定した適合度の値を CR 値と呼ぶ．以下，CR 法について説明する．

重みの定義

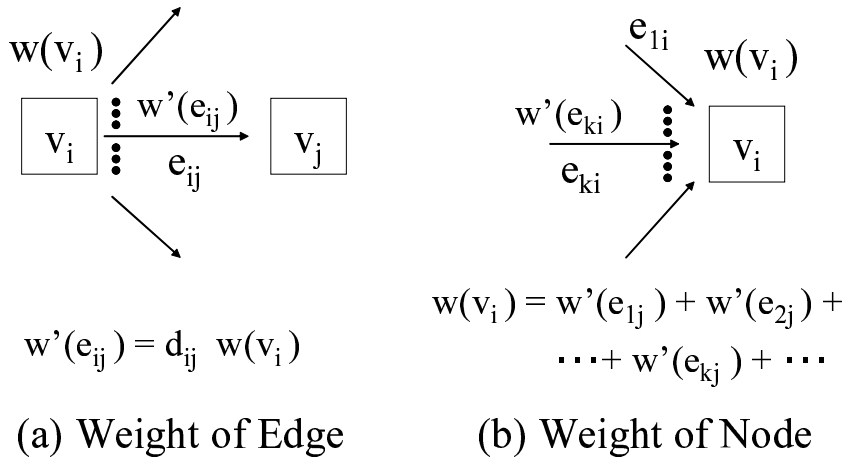

図 3: 重みの定義

CR 法では，図 1 部品グラフ $G = (V, E)$ 上の個々の辺および頂点に対して重みを計算し，対応する頂点の重みをもとに各部品の重要度を評価する．頂点の重みと，辺の重みを次のように定義する．

定義 1 (頂点の重みの和) 部品グラフ G 上の各頂点 v は $0 \leq w(v) \leq 1$ の重みを持ち，G の頂点の重みの総和は *1* とする．すなわち，

$$\sum_{v \in G} w(v) \quad = 1$$

定義 2 (辺の重み) 頂点 v_i から v_j への辺 e_{ij} に関する辺の重み $w'(e_{ij})$ は，

$$w'(e_{ij}) = d_{ij} \times w(v_i)$$

図 3 (a) はこの定義を図示したものである．d_{ij} は配分率とよび，$0 \leq d_{ij} \leq 1$ かつ $\sum_i d_{ij} = 1$ を満たす値とする．頂点 v_i から v_j へ利用関係が存在しない場合，ここでは $d_{ij} = 0$ とする．この配分率 d_{ij} は，次に示す頂点の重みの定義において，有向辺の終点となる頂点の重みの決定に利用される．図 3 (b) は次の定義を図示したものである．

定義 3 (頂点の重み) *IN(v_i)* を v_i を終点とする有向辺の集合とする．この時，頂点 v_i の重みは v_i が終点となる有向辺 e_{ki} の重みの総和とする．つまり，

$$w(v_i) = \sum_{e_{ki} \in \text{IN}(v_i)} w'(e_{ki})$$

重みの計算

定義に基づいて，頂点 $w(v_i)$ に対して次の方程式が生成できる．

$$w(v_i) = \sum_{e_{ki} \in \text{IN}(v_i)} d_{ki} \times w(v_k)$$

各頂点に関してこの方程式を立てる事で，$n(= |V|)$ 個の連立方程式が生成できる．また，各頂点の重みをベクトル W として次のように表現し，

$$W = \begin{pmatrix} w(v_1) \\ w(v_2) \\ \vdots \\ w(v_n) \end{pmatrix}$$

配分率を次の行列 D として表現する事で，

$$D = \begin{pmatrix} d_{11} & d_{12} & ... & d_{1n} \\ d_{21} & d_{22} & ... & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n1} & d_{n2} & ... & d_{nn} \end{pmatrix}$$

$n(= |V|)$ 個の連立方程式を次のように表す事ができる．ここで 行列 D^t は D の転置行列を表す．

$$W = D^t W \tag{1}$$

定義 1 から，行列 D^t は推移確率行列の性質を満たしているため，D^t を用いて開発者が部品をどのように参照するかをマルコフ連鎖 [6] でモデル化し表現する．すなわち，開発者はある部品を参照した後に，辺に沿って利用関係のある部品の一つを参照するとみなす．ここで，式 (1) を満たす解 W は D^t に関する定常分布となり，対象とする部品の集合の中での参照を十分長い間時間繰り返した場合の，開発者によって各部品が参照されている確率を示すことになる．つまり，重みが高い部品ほど開発者によって頻繁に参照されることになる．この値を重要度とすることで，ただ単に利用数が多い部品だけでなく，利用数が多い部品が利用している部品も重要であると評価することが出来る．定常分布が一意にもとめられる事を示すためには，D^t が既約であることが必要となるが，CR 法では，既約である事を保証するために後述する補正を加えている．

この場合，D^t における絶対値最大の固有ベクトルを求める事で，定常分布を求めることができるが，行列 D^t の絶対値最大の固有値は 1 であるため，累乗法 (power method) を用いて

適当な初期ベクトルに対して繰り返し D^t を掛ける事で，近似解を容易に求める事ができる．図 4 は，与えられたグラフにおいて各頂点の重み（重要度）を計算した結果である．v_1 は 2 つの有向辺の始点で，v_1 の重み 0.4 は二つの辺に 0.2 ずつ等分されている (つまり，$d_{12} = d_{13}$ = 0.5)．また，v_3 は 2 つの有向辺の終点で，それぞれの辺が 0.2 の重みを持つため，v_3 の重みは 0.4 であることがわかる．

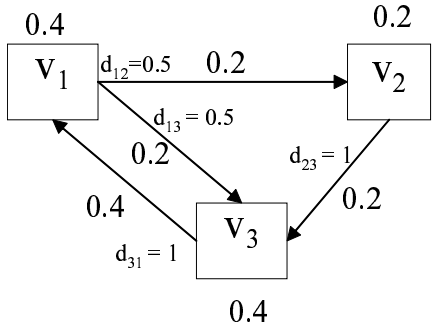

図 4: 重みの計算例

部品重要度計算に関する補正

部品の重要度の計算に関する説明を行ったが，実際に運用する場合不具合が起こる場合がある．例えば，部品 i がどの部品も利用していない場合，頂点 v_i から全ての頂点への配分率 d_{ix} が全て 0 になり，配分率に関する定義（頂点からの辺への配分率の総和が 1）を満たさなくなる．また，グラフが図 5(a) のように強連結でない場合，頂点 v_1 の重みが 0 になってしまい，v_1 から v_2 への利用関係を正しく評価する事ができない．

このような場合，与えられた行列が既約であることを保証できず，定常分布が一意に定まらない場合がある．そこで，全ての頂点を図 5 (b) のように低い配分率の擬似辺で結ぶ事で，与えられたグラフを強連結なグラフに変換し，行列が既約である事を保証する．この擬似辺は，各部品における自分を含めた全ての部品への明示的でない参照を意図している．ここで p は実際の辺と擬似辺の重みの配分比率を指す．

定義 4 (修正配分率) 頂点 v_i を始点とする辺への配分率 $d'(e_{ij})$ を次のように定義する．

$$d'_{ij} = \begin{cases} p \times d_{ij} + (1-p)/n & \textit{if } v_i \text{ からの辺が存在} \\ 1/n & \textit{if } v_i \text{ からの辺がない} \end{cases}$$

この補正は，前節で説明したマルコフ連鎖を用いたモデル上での参照行為を,「ある部品を参照した後に，辺に沿って利用関係のある部品の一つを参照するが，ある確率（$1-p$）で，ランダムに全部品の中の一つを参照する」のように修正する．この修正により，参照行為をより自然な形でモデル化することができる．

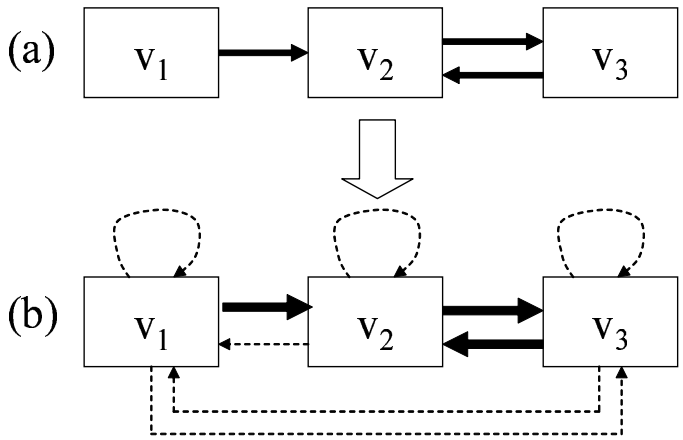

図 5: 部品重要度計算に関する補正（擬似辺の追加）

類似部品化の効果

それぞれの類似部品へ指されていた辺が，部品群化を行う事で一つの類似部品群への辺とみなされる．そのため，コピーして再利用される回数が多いほど，その部品群は多くの部品から利用されることになる．このため，コピーされたという利用関係を重要度に反映させる事ができる．この時，G の部品群グラフ（*clustered component graph*）$G' = (V', E')$ を次のように定義する．

定義 5 (部品群グラフ $G' = (V', E')$) V の商集合からなる集合 V' を G' の頂点集合とする．また v_i，v_j が属する V' 中の集合をそれぞれ v'_i，v'_j としたときに，$E' = \{(v'_i, v'_j) | (v_i, v_j) \in E\}$ を G' の辺集合とする．

図 6 は部品群化を行った際に，類似部品の重要度がどのように変わるかを例で示したものである．3 つのシステムのうち，2 つのシステムには $X.A$ および $Y.A$ という同一の部品が存在する．部品がコピーされた場合は，部品が属する名前空間（パッケージ名）が異なる，または部品名が異なるなどの理由から $X.A$，$Y.A$ は別々の部品として扱われてしまう．部品群化を行わないと，コピーされた部品は別々の部品として評価されるため，図 6 の場合すべての部品の重要度が等しくなる．そこで，類似部品群化によって部品 $X.A$ および $Y.A$ へのそれぞれの有向辺が一つの部品群 A' への有向辺に統合され，部品 A の重要度が他の部品よりも高くなる．以降の実際の重要度計算では，部品グラフではなく部品群グラフを対象としており，部品群グラフに対して修正配分率に関する補正を行った上で部品群における重要度を計算している．

順位の統合

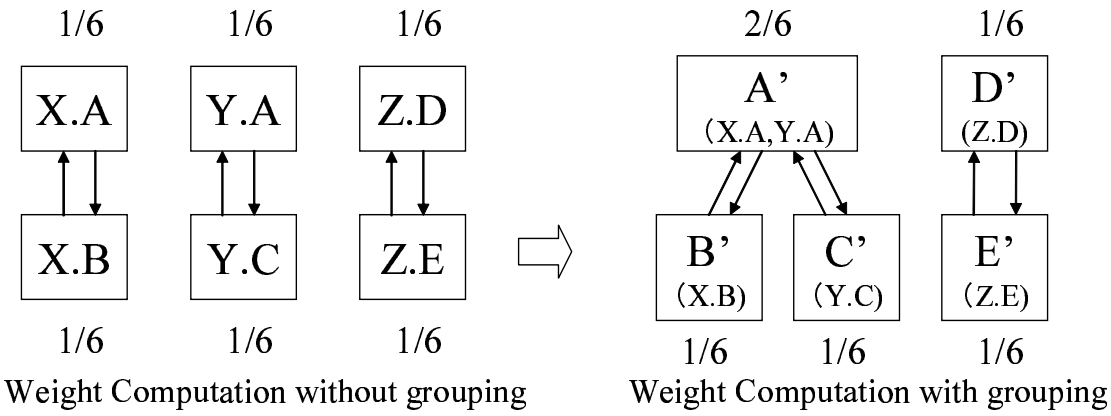

図 6: 部品群化の効果

KR 法によって得られた順位と CR 法によって得られた順位を統合する．順位の統合はメタ検索システムで利用される手法であり，経験的な (heuristic) なものとして知られている．メタ検索システムとは，複数の検索システムで同一の検索対象集合の検索を行ない，検索結果を統合するシステムである．検索システムそれぞれが持つ検索アルゴリズムの特徴を考慮した検索が可能であり，検索結果の再現率と適合率を改善することができる．順位の統合は，広告目的で作成者が検索結果で意図的に高い順位を得ようと作成した Web ページなど，不相応に高い順位を得ているもの (spam と呼ばれる) の順位を下げ，情報の健全性を守ることができるとされている [10]．最も単純な順位の統合手法に Borda の手法 [7] がある．計算量が少なく，線形時間で解けることが知られている．Borda の手法では，各順位に対して評価点を割り当て，その合計値をもとに最終的に統合された順位を求める．

SPARS-J では，検索結果を KR 法，CR 法で評価し，それぞれの順位を部品の表価点とする．そして評価点を足した合計点によって検索結果を昇順に並べ直し，検索結果として提供する．例を用いて Borda の手法を説明する．表 3 は KR 法，CR 法，および Borda の手法で統合された CR と KR の統合結果 (以降 CR+KR と表記する) を，表 4 はそれに対応した各部品の評価点を表す．表中の A，･･･，J はそれぞれ部品である．例えば部品 A は KR で 1 位，CR で 4 位であり，評価点は 5 点となる．よって部品 A は順位を統合した結果 2 位となる．

2.2.6 システムの実装

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[28] を，字句解析に [12]，構文解析に [11] を用いてシステムの実装を行なった．さらに日本語検索対応のために日本語わかち書きソフトウェア KAKASI[19]，および日本語表示のために gettext[13] を用

	KR	CR	CR+KR
1 位	A	C	C
2 位	E	F	A
3 位	C	G	E
4 位	I	A	F (3 位)
5 位	J	B	B
6 位	B	D	I (5 位)
7 位	H	I	G
8 位	F	E	J
9 位	G	H	D
10 位	D	J	H (9 位)

表 3: Borda の手法

	KR	CR	CR+KR
A	1	4	5
B	6	5	11
C	3	1	4
D	10	6	16
E	2	8	10
F	8	2	10
G	9	3	12
H	7	9	16
I	4	7	11
J	5	10	15

表 4: 評価点

いた．ソースコードの規模は，データベース約 9100 行，部品解析部約 13100 行，部品検索部・データ表示部約 11000 行，合計約 33200 行であった．

2.2.7 システムの機能

本システムが持つ機能は以下の通りである．

- キーワード検索
  ユーザが要求したキーワードを検索語として部品の検索を行なう．日本語を検索語とすることも可能である．’@’および’.’，’$’には特殊な意味がある．’@’ではじまる検索語はパッケージ指定であり，指定したパッケージのみを対象に検索を行なう．他に検索語を入力していなければ，指定パッケージをパッケージブラウザで表示する．’.’または’$’を含む検索語は，部品の完全限定名として解釈され，その部品を検索する．’$’は内部クラスとして解釈される．この時，他に入力された検索語は無視される．
  - 検索対象とするトークン種類の指定
    キーワード検索の際に，検索対象とするトークンの種類の指定を行なうことができる．例えば，クラス定義名のみを対象に検索を行なうことが可能である．
  - 検索結果の順位付け手法の指定
    検索結果の順位付けを，CR，KR，および CR+KR のいずれを用いるか利用者が選択可能である．利用者の目的に合わせた検索を行なうことができる．
- 類似部品群の提示
  類似していると判定された部品の一覧が得られる．

- 利用関係の提示
  部品間の利用関係を提示することで，部品の使用方法に関する情報を得られる．

- パッケージブラウザ
  パッケージ階層構造をハイパーリンクで表現しており，パッケージ内に含まれる部品の一覧を得られる．

- メソッド一覧表示
  部品内で定義されているメソッドの一覧が表示される．メソッド名はメソッド定義行へのリンクになっており，辿ると定義行へジャンプできる．

- キーワードハイライト
  ヒットしたキーワード部分がハイライトされて表示される．ハイライトをクリックすることで，次のハイライト部分へジャンプできる．

- アーカイブダウンロード
  部品が含まれる jar ファイルなどのアーカイブをダウンロードして使用することができる．

2.2.8 システムの構成

本システムは以下のような部分から構成される．構成を図 18 に示す．

- ライブラリ
  収集した Java ソースファイルが蓄積されているライブラリ．SPARS-J 上では部品の完全限定名で部品の分類を行なうので，ディレクトリ階層をパッケージに合わせる必要は特にない．検索結果の部品詳細表示では，ライブラリ中の該当部品が存在するファイルを参照し，表示する．

- データベース (File DB, Component DB, Relation DB, Word DB)
  ライブラリ中のファイル情報を管理する File DB，登録された部品に関する情報を管理する Component DB，部品間の利用関係を管理する Relation DB，索引の管理をする Word DB がある．各 DB は，ID からファイル名や部品名，ファイル名や部品名から ID を対応付けるために正引き，逆引き索引を持つ．

  – File DB
    ライブラリ中の Java ファイルのファイルパス，ファイル ID，各ファイルに存在

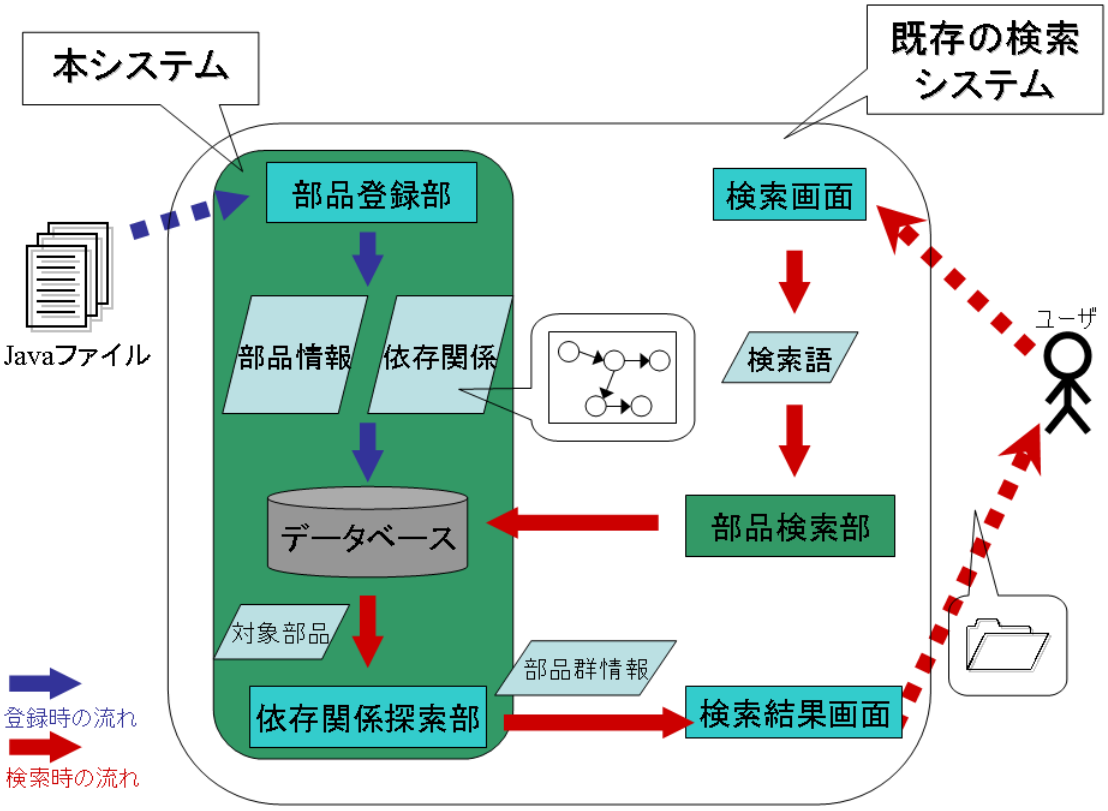

図 7: SPARS-J の構成

する部品の ID，更新時間などの情報が格納されている．さらに，将来の拡張 (他言語対応など) に備えてファイルの記述言語情報も格納している．

- Component DB
  部品名，部品 ID，部品が存在するファイル ID，ファイル中の部品の場所など部品とライブラリ中のファイルを File DB を介して対応付けるための情報の他に，各部品中で定義されたメソッドやフィールド名，類似判定のためのメトリクス，各部品の類似度，部品が属する部品群 ID，CR 値，部品中に現れた索引キー ID や出現頻度，場所など部品に関する全情報が格納されている．
- Relation DB
  部品間の利用関係と，その利用関係の種類が格納されている．その他，登録過程でまだ登録されていない部品への利用関係があった場合，利用する部品名を未解決名として格納し，部品が登録され次第利用関係を追加するための未解決名情報が格納されている．
- Word DB
  登録された部品中で出現した全索引キーとその ID を格納する DB である．検索時の大文字小文字の区別のために区別あり索引情報と，区別なし索引情報を格納している．

- 部品登録部 (Register, Ranker, Sorter)
  部品登録部はライブラリのファイルを解析して，各種情報をデータベースに格納する Register，利用実績に基づいて部品群の CR 値を計測する Ranker，検索時の高速化のために CR 値によって部品群を降順にソートし，各部品の索引キーとその出現頻度から索引キーの重みを計測する Sorter，以上 3 つのモジュールから構成される．

  - Register
    利用者が指定した Java ソースファイルを解析して，部品の切り出しの他，索引，メトリクス，利用関係などの抽出，さらに類似度の測定と部品群化を行なう．ファイル，部品，部品群，索引キーへの ID を割り振り，正引き，逆引き索引も作成する．サブモジュールとして，ソースファイル解析部 Parser と部品群化部 Cluster を持つ．
  - Ranker
    Register が抽出した利用関係から，各部品の CR 値を計測する．
  - Sorter
    SPARS-J は部品検索時に CR と KR の順位の統合を行なうため，データベース構

築時にそのオーバーヘッドを軽減しておく必要がある．そのため，CR 値によって部品群を降順にソートしておき部品検索時には既に CR でソート済みのリストを取得できるようにする．また，KR 値は部品検索時に計測するが，Sorter 索引キーと出現頻度から索引キーの重みを計測しておくことで，検索時に KR 値の計測を高速に行なうことができる．

- 部品検索部 (Searcher)
  Searcher はクエリの解析，部品の検索を行なうモジュールである．利用者が指定したクエリを解析し，得られたキーワードとトークン種類の組を検索キーとしてデータベースの検索を行なう．得られた結果を順位付けし，各部品の KR 値の計測，CR と KR の順位の統合を行なう．検索結果と併せて，部品の詳細情報として，ソースコードや利用関係，メソッド一覧などを利用者に対して提供する．

- データ表示部 (Web インタフェース)
  データ表示は CGI であり，Web インタフェースを利用している．ユーザから受け付けたクエリを部品検索部に引き渡す．また，Searcher から受け取った検索結果をユーザに対して提供する．検索結果と併せて，部品の詳細情報として，ソースコードや利用関係，メソッド一覧などをユーザに対して提供する．

2.2.9 データベースの説明

File DB

File DB はライブラリ中の Java ファイルと Component DB 中の部品の対応をとるための情報を格納する．解析対象の Java ファイルは一意なファイル ID を与えられ，ファイル中に存在する部品に割り振られた部品 ID と対応付けられる．同名のファイルが存在しなければ新規ファイルとして格納し，一意なファイル ID を与える．存在していればファイルの更新とみなし，File DB のファイル ID や含まれる部品 ID の情報を更新する．

Component DB

Component DB には Java ファイルを解析することによって得られた，各部品に関する情報が格納される．主に以下のような情報がある．

- 部品の完全限定名と部品 ID
- 部品のファイル中の出現位置
- 部品がインポートしたパッケージ
- 部品中で定義されたコンストラクタ，メソッドと行番号

- 部品中で定義されたフィールド変数と行番号
- 部品の特徴メトリクス値
- 部品が所属する部品群 ID
- 部品の CR 値

Relation DB

各部品間の利用関係を格納する DB である．利用関係は，利用関係の種類と関係を結ぶ部品 ID の組から構成される．さらに，未解決部品のリストも持つ．現在登録している部品がまだ登録されていない部品を利用した場合に，一時的に未解決部品として保存しておき，その部品が登録され次第そのリストから未解決部品を削除して，利用関係として登録する．

Word DB

Word DB には索引キーの情報が格納される．索引キーに対して一意な ID が割り振られており，同じ索引語でも異なったトークン種類であればその ID も異なる．また，検索時の大文字小文字区別のために索引語を全て小文字に直したものと，そのままのものをそれぞれ格納している．さらに，全ての索引キー ID はどの部品の何トークン目に出現したのか，といった情報を持っている．また，部品検索部における KR 法の高速化のために，あらかじめ索引キーの重みを Word DB に格納している．

2.2.10 部品登録部の説明

部品登録部は，Java ファイルを解析することで，検索時に必要な情報を抽出し，それらの情報を各種 DB に格納する．

Register 部品登録部に入力された Java ファイル群は Register で解析される．Register は Java ソースコードの解析を行ない部品や索引キー，利用関係，部品の特徴メトリクスを抽出する Parser と，特徴メトリクスから部品の類似度を判定し部品群化を行なう Cluster という 2 つのサブモジュールを持つ．

Parser はファイル中からクラスまたはインタフェースを見つけ出し部品として切り出し一意な部品 ID を与える．部品は記述されているファイル ID やファイル中の行番号をファイル位置情報として Component DB に格納する．

さらに，各部品の索引付けを行なう．検索時に利用するために各部品中に含まれる識別子やコメント中の語句を索引語として抽出し，トークン種類毎に出現頻度をカウントする．それらを索引キーとして Component DB 格納する．

また，部品間の利用関係の抽出を行う．抽出した利用関係は Relation DB に格納する．

その他に特徴メトリクスの計測を行い，後述する Cluster によって部品群化を行なう．計測したメトリクス値は Component DB に格納する．部品の特徴メトリクスを計測し比較することで，類似部品をまとめた部品群を作成することが可能になる．すべてのメトリクスは構文解析時に計測可能であるため，Parser で特徴メトリクスの計測だけを行う．

Ranker

Relation DB に格納された部品群間の利用関係から CR 値の計算を行う．CR 値は部品群 ID と対にして Component DB に格納する．

Sorter

Component DB から CR 値を参照して部品群の順位付けを行なう．また，後のキーワード出現頻度に基づく順位づけのために各索引キーの重みを算出しておく．

CR 法の配分比率について CR 法では，パラメータとして辺と擬似辺の重みの配分比率 p が存在する．配分比率 p に関して p の値を変えて順位の変動を検証したところ，$p = 0$（補正のみしかないため全ての部品が同順位になる）と $p = 1.0$（補正無し）の時以外は，値を変更しても結果として得られる順位に違いはほとんど見られなかった．また，$p = 1.0$（補正なし）の場合は補正を行った場合と比べて 10 倍以上の計算時間を必要とした．このことから，p は順位の決定に影響を与えるパラメーターではなく，計算の収束にのみ必要なパラメータで，p はどの値でも問題ないことがわかった．そこで，現在のところ最初に採用していた値である $p = 0.85$ を利用している．

2.2.11 部品検索部

部品検索部は，検索者によって与えられたクエリを解析して検索キーを取り出し一致する部品を検索する．そして，検索結果の部品を評価値が高い順に順位付けしてデータ表示部に渡す．クエリには検索キー以外にも以下の項目を検索条件として指定することができる．

- 検索対象とするパッケージ名
- 検索対象とするトークン種類
- 順位付けの方法
- 検索方法 (AND-OR 検索) や，検索キーの大文字小文字の区別の有無

部品検索部は，検索条件に応じて Component DB の適切な部分のみを検索し，ヒットした部品の部品 ID のリストを作成する．ヒットしたリスト中の部品 ID を持つ各部品について，索引キー中の索引語の重みの総和を検索キーと部品の適合度とし，KR 法を用いて順位付けする．

その後，求めた KR 法による評価値と，Ranker で求めた CR 法による評価値をもとに検索条件に指定した方法でリストを再度順位付けし，結果をデータ表示部に渡す．指定できる順位付けの方法として，KR のみ，CR のみ，および Borda の手法で統合した CR+CR を選択できる．デフォルトは CR+KR である．

2.2.12 データ表示部

データ表示部は部品検索部と検索者を結ぶ Web インタフェースである．SPARS-J は Microsoft の Internet Explorer などの Web インタフェースを通して検索機能を提供する．検索者が指定した検索条件を部品検索部に渡し，部品検索部が返してくる検索結果の部品リストを受け取って表示する．

検索結果表示では，ヒットした部品を部品リストの順位で表示する．検索結果表示の様子を図 10 に示す．各部品の行数や部品中で定義しているメソッド数，部品の利用実績を併せて表示することで，再利用に有益な情報を検索者に提供する．詳細を知りたい部品を選択することで，部品の詳細データ表示が可能である．

部品の詳細データ表示では，部品のソースコードや，同一部品群に属する部品，パッケージブラウザによるクラス階層構造，部品間の利用関係，などといった各部品の詳細情報を閲覧することができる．

2.3 SPARS-J の効果と問題点

SPARS-J が検索語に対して，高い精度で有効なソフトウェア部品を検索結果にランキング表示することができることがわかっている [32]．そこで，SPARS-J を用いることで，プログラム理解，再利用に対する支援目標の１つである「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」とう問題は解決することができる．

一方，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１つの部品の情報しか表示しないことが多いため，もう一つの支援目標である「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．なぜなら，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は，目的とする機能を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良いが，処理が複数の部品に渡って実現されているような場合には上記のような 1 つの部品の情報しか表示しないデータ表示能力では不十分だからである．というのも，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のが原因である．図 12 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

図 8: 検索画面

図 9: Advanced Search

図 10: 検索結果表示

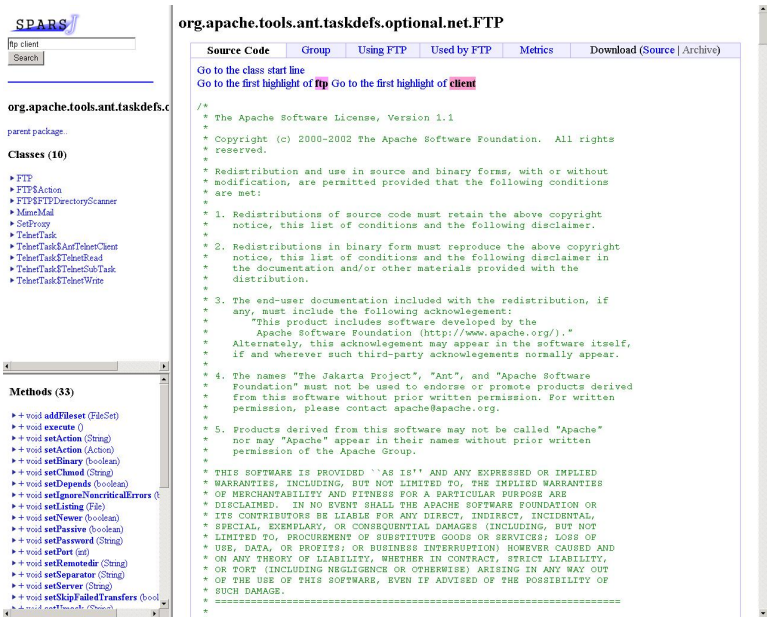

図 11: 部品詳細表示

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．ここまで追跡して初めて，メソッド a の処理を理解するためには，メソッド a，メソッド b，メソッド c の 3 つの処理を理解しないとならないことがわかる．

既存のソフトウェア検索システムでは，上記のように，処理内容を追跡して計 3 回検索しないとこのことが分からない．つまり，内部の依存関係が複雑になればなるほど検索者の労力は増えていく．

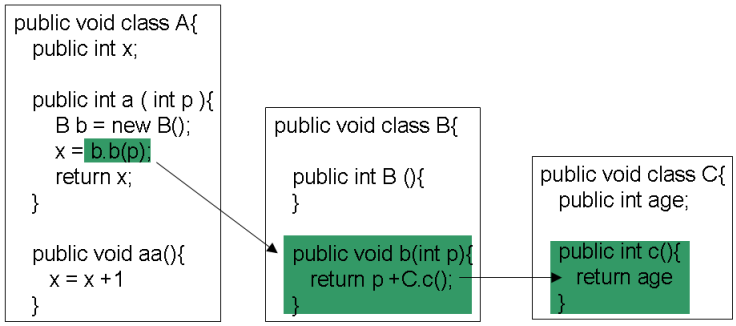

図 12: 機能の実装範囲

このような事実から，ソフトウェア検索システムの検索結果表示においては，検索にヒットした部品を起源とする依存関係部品群の情報を利用するような手法が必要であることがわかる．

2.4 関連研究

ソフトウェア部品を単体ではなく，部品群という単位で情報を提供する研究として，鷲崎ら [30] は独立して再利用可能なコンポーネントを，検索および試行可能な検索システムの提案を行なっている．提案システムでは，Java ソースコードからクラス間の依存関係を解析することで，要求された機能を実現する複数のソースコードを JavaBeans コンポーネントとして抽出し，ユーザに提供する．本論文で提案する手法が，検索結果の部品に対して，依存部品群を関連づけるのに対して，鷲崎らは検索システムへ登録する際に，試行可能な依存範囲であらかじめ部品群を構築しておく点が異なる．

また，これまでにソフトウェア部品検索を行なうために様々な手法が提案されている．

WWW のソフトウェア部品を自動収集し，それらを解析することでデータベースに分類・蓄積し，作成したデータベースを対象に検索を行なう形式の検索システムに [27][4][20] がある．Agora[27] はサーチエンジン AltaVista の SDK を利用してソフトウェア部品の自動収集を行なう．CORBA オブジェクトである ORB や JavaBeans を検索可能である．jCentral[4] は IBM が開発した Java のソフトウェア部品検索システムである．jCentral では Java ソースコードの他，アプレット，JavaBeans，ニュース，FAQ，チュートリアルなどの情報を検索することが可能である．亀井ら [20] の提案したシステムでは，クエリにソフトウェアメトリクスなどの特有の値を含めてソフトウェアを検索することができる．

庭山ら [25] はセマンティック Web の手法を利用してソフトウェアのソースコードにメタデータを付与し，検索ワードをオントロジ変換したものとのマッチングを行なうことで，ソフトウェア部品を意味に基づいて検索する手法を提案している．

その他，ソフトウェア部品検索として用いることが可能な自然言語文書検索システムとして，日本語全文検索システム namazu[24] や Web ページ検索システム Google[14] などが存在する．

3　メソッド間の依存関係を利用したプログラム理解手法

本節では，本研究で提案するプログラム理解支援手法の概要と，実現のために用いている主な技術について述べる．

3.1　プログラム理解支援手法の概要

本手法は，Java 言語 [18] で記述されたメソッドを対象とした依存部品群の解析，表示を行うシステムである．利用者は，ソフトウェア検索システム等で再利用したい機能をもつメソッドを検索する．本手法は，ソフトウェア検索システムの部品表示部に実装することで，そのメソッドを起点とする依存部品群を解析する．さらに，ソフトウェア部品重要度評価手法 [15] によって依存部品郡内における各部品の呼び出し頻度を計算する．また，依存部品群内の利用関係をツリー形式で表示したり，各部品の特性メトリクス情報を提供することで，再利用やプログラム理解・保守を行なうことが可能となる．

3.2　ソフトウェア部品

本研究の目的は，機能単位のプログラム理解，再利用の支援することであり，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呼ぶときはメソッドのことを指すこととする．

3.3　依存解析の粒度

2.2.10 節で述べたように，SPARS-J 等ではクラス間の依存関係を解析しており，それは図 13 のように表現することができる．

ここで，点はクラスであり，各点の上部の文字はクラス名を表している．また，各辺はクラス間の利用関係とする．

Java においては，1 つの機能はメソッドに集約されて記述されていることが多く，機能単位のプログラム理解，再利用を考慮した依存関係解析を考えると，ある機能に関する依存関係は点がクラスのままでは冗長性を持つ．

なぜなら，クラス間の依存関係をさらに分析すると，そのほとんどはメソッド呼び出し関係であり，メソッドを考慮した場合の依存関係は図 14 のように表現することができる．ここで，点の中にある長方形はクラスで宣言されているメソッドとする．

ここで，クラス A のメソッド a が内部の処理で実際に依存している範囲というのは，メソッド a から依存辺を辿って到達可能な部品群である．それらは図 14 で説明すると，斜線を引いたメソッド群であって，斜線以外のメソッドは，メソッド a の機能を理解する上では

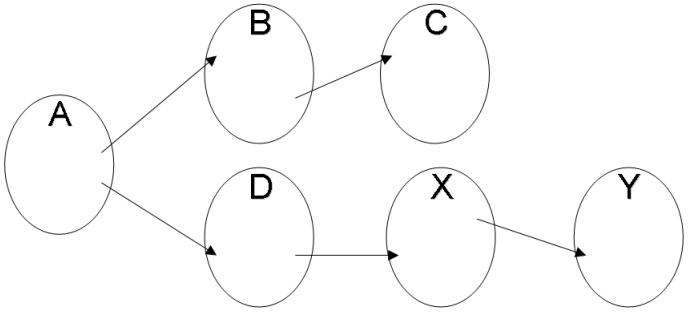

図 13: クラス間の依存関係

全く関係がない冗長な範囲となる．特に，クラス X からクラス Y への利用関係などは，メソッド a が直接利用しないクラス X のメソッド x を実現するための利用関係でありメソッド a の機能を理解する上では冗長な情報である．

この冗長性はクラスやメソッドの規模が大きくなればなるほど顕著にあわられることは自明であり，このような冗長な依存関係は，検索者に目的のメソッドの処理の本当の依存範囲を判断させる労力を強いることになる．

そこで，本研究では機能単位の依存関係を解析する際の粒度としてメソッド間の依存関係を採用する．こうすることによって上で述べた冗長性は排除され，起点となるメソッドの実装を読み取るうえで本当に必要な範囲のみを知ることができる．

3.4 依存関係

本研究でいう部品間の依存関係とは，部品間のメソッド呼び出し関係のことをいう．

3.5 再利用部品例の解析

図 14 で，目的クラスからの依存辺を辿ることで依存部品を特定することができることを示した．

これとは逆に，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 15 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

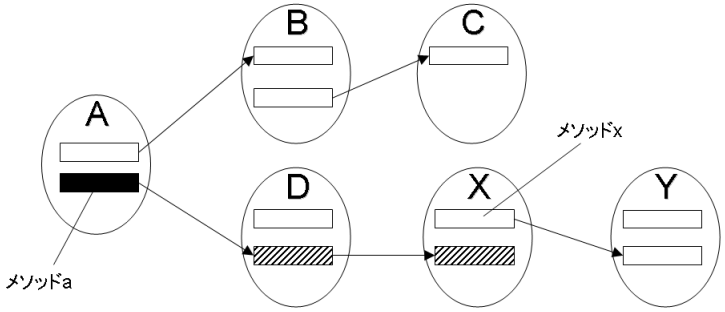

図 14: メソッド間の依存関係

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．

3.6 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 17 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．

3.7 依存部品グラフ探索の範囲

依存部品グラフから，依存部品群を解析できることを示した．この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 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 16 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

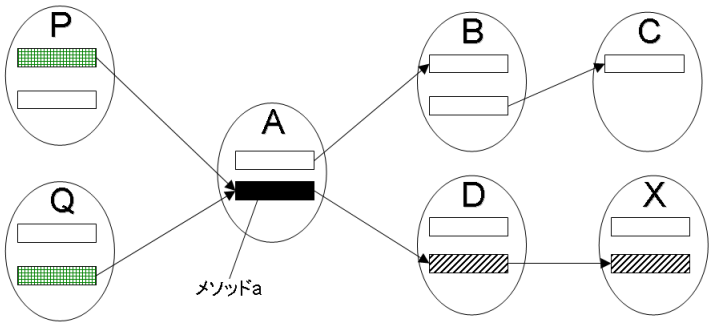

図 15: 部品の利用例

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.

一方，javax.xml.parsers.DocumentBuilder.main() が起点の場合には 1，2，3，5 番についてはそれ以降の解析を止める.

- 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()

上のような場合には，org.apache.XXX. の XXX がプロジェクト名を示しているので，org.apache.XXX まで一致しない部品はライブラリ的に使用したと設定したいが，自動化するとなるとパッケージ名のどこまでが，プロジェクト名を示していてどこからがそのプロジェクトの内部的

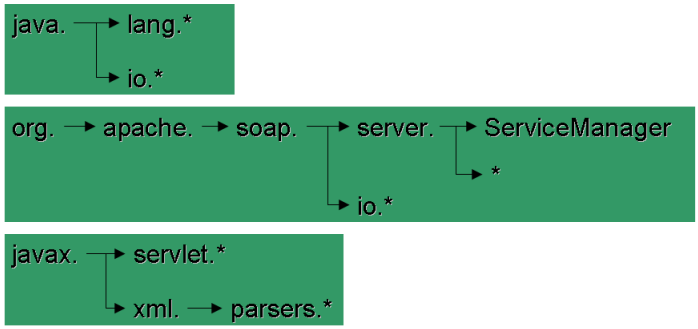

図 16: グラフの探索範囲

なパッケージ名なのかということが判別できないため，本手法では暫定的な手段として先頭のパッケージ名が変わった場合に解析を止めるという処置を施している．

将来的な課題としては，ユーザごとに設定ファイルなどに既知のプロジェクト名やパッケージ名を記述してユーザや本システムの利用状況ごとに解析中止判定のレベルを設定できるようにすることなどが考えられる．

3.8 解析対象

本手法が仕様として想定する解析対象は，全ソースコードのそろったソフトウェアである．これは，他の部品から呼び出されているメソッドを記述しているソースコードが欠落していると，その内部以降の依存関係が解決できずに正確な依存範囲が特定できないためである．

3.9 解析精度

本手法では，ソースコードから得られる静的な情報を解析して分かる範囲の精度を持つ．動的にクラス名が決まるような記述があると，その利用関係は捕捉できない．

また，全く同じパッケージ名とクラス名をもつ複数の部品が登録されていて，ある部品からその部品に利用関係があった場合は，それらの部品を置いてあるファイルパスが近い方を採用する．

4　提案手法の実現

本節では構築したプログラム理解支援システムについて説明する．

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[28] を，字句解析に Flex[12]，構文解析に Bison[11] 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．ここで，表示形式をグラフではなくツリーにした理由は，理解支援対象とする部品に依存関係の規模が大きくなったり，複雑になるとグラフでは理解が急激に困難になるのに対し，ツリー形式では，そのようなことがないためである．

  ツリーにしたために，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 [33] 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 [35]．また，C.Jones[34] 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．

4.2 システムの構成

本システムは以下のような部分から構成される．構成を図 18 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ます．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します．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します．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ます．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します．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ます．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します．

以降，システムの各部に関して詳細に説明する．

4.3 データベース

データベースは 4 つあり，それぞれ以下のような詳細を持つ．

- Method DB
  Method DB には Java ファイルを解析することによって得られた，各部品に関する情報が格納される．主に以下のような 6 つの情報がある．

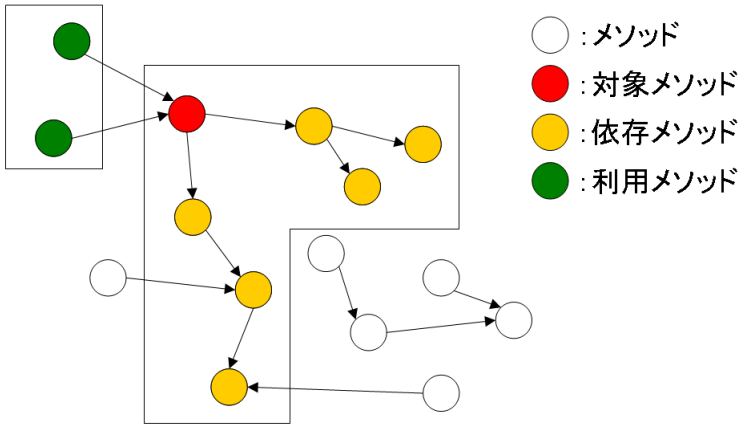

図 17: 部品依存グラフ

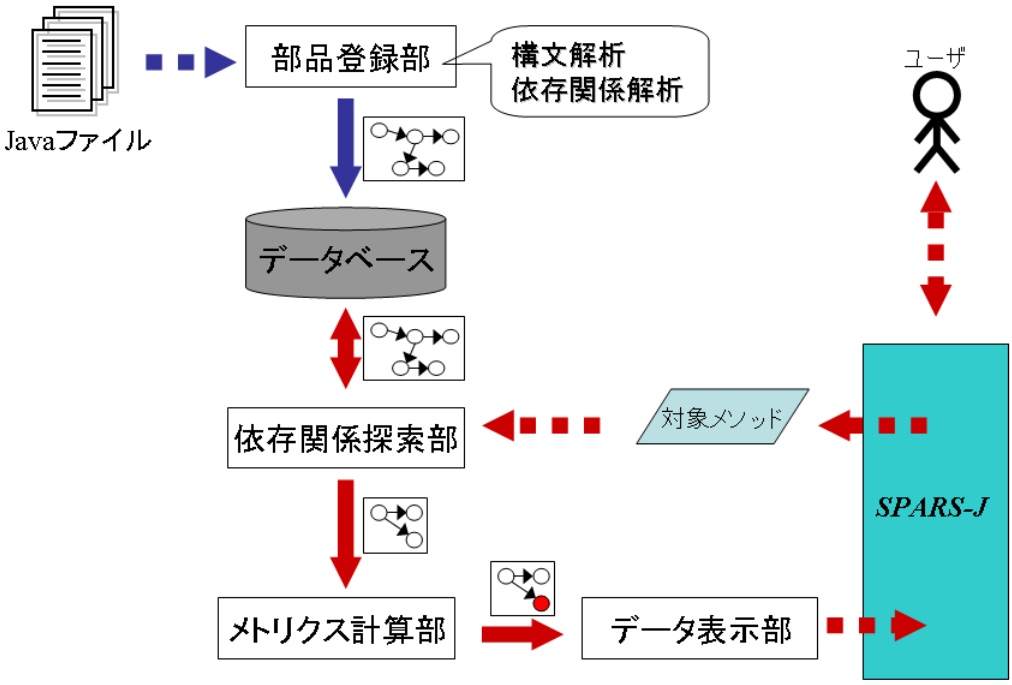

図 18: 本システムの構成

  - メソッドの ID
  - メソッドの完全限定名
  - メソッドのソースコードファイル中の記述位置
  - メソッドの返り値，引数の型名
  - メソッドの特徴メトリクス値
  - メソッドを宣言しているクラスの ID

- Class DB
  クラスに関する情報を格納する DB である．SPARS-J の ComponentDB を利用した．MethodDB の「部品を宣言しているクラスの ID」がこの DB の ID に対応する．

- File DB
  ソースコードファイルに関する情報を格納する DB である．SPARS-J の FileDB を利用した．

- Method Relation DB
  各部品間の利用関係を格納する DB である．利用関係は，利用関係の種類と関係を結ぶ部品 ID の組から構成される．さらに，未解決部品のリストも持つ．現在登録している部品がまだ登録されていない部品を利用した場合に，一時的に未解決部品として保存しておき，その部品が登録され次第そのリストから未解決部品を削除して，利用関係として登録する．

4.4 部品登録部

部品登録部は，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．

まず，Java ファイルに一意な File ID を与え，ファイルパスと共に File DB に登録する．次に Java ファイルからクラスまたはインタフェースを抽出し，一意な Class ID を与えると同時に，名前と，それが記述されているファイルの File ID と共に Class DB に登録する．クラス，インタフェースを登録した後は，それらの中で宣言されているメソッドを抽出し，一意な Method ID を与えると同時に，名前と返り値，そのメソッドが宣言されているクラスの Class ID を一対として Method DB に登録する．また，各クラス，メソッドがファイルの何行目から何行目までに記述されているかを解析し，それぞれ開始行番号，終了行番号として登録する．さらに，ソースコード中の if-else 文，for 文，while 文，switch 文，try-catch 文の処理分岐数を解析して，対応するクラス，メソッド当たりのサイクロマチック数を計測して DB に登録する．

一方で，メソッド間の利用関係も登録する．各メソッド間の呼び出し関係を解析し，呼び出し先メソッドと名前 (完全限定名) が一致するメソッドが既に登録されてあれば呼び出し元，呼び出し先のメソッド ID の対と，利用関係の種類を Method Relation DB に登録する．名前が一致するメソッドが無ければ，未解決 DB に登録しておき，将来一致する名前を持つメソッドが登録されたときに初めて，Method Relation DB にその利用関係を登録して，未解決 DB から情報を削除する．

こうすることによって，メソッドの呼び出し元，呼び出し先のどちらが先にシステムに登録されても整合性がとれるようにしている．以上のようにしてメソッドとメソッド利用関係を登録していき，3.6 節で説明した部品依存グラフを作成する．

4.5 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．

システムは，依存部品解析部が探索を終了した時点で形成されている依存部品群を，その要求部品の依存部品群と認識する．

4.6 データ表示部

データ表示部は，依存部品解析部で作成した依存部品群に関する情報を表示する．表示部は大きく分けて 4 つに分かれており，各部を図 19 の A から D と呼ぶことにする．

4.6.1　利用部品表示部

図 19 中の A の部分がこれに該当する．

ここでは要求部品を実際に利用しているクラスが表示される．これらの利用部品は再利用例としてユーザに提示される．

4.6.2　依存部品群の特徴メトリクス表示部

図 19 中の B の部分がこれに該当する．

各メトリクスの部品群の総計と，要求部品に対する割合が表示される．表示する特徴メトリクスは以下の 3 種である．

- LOC
- サイクロマチック数
- メソッド数

上記のメソッド数は部品群の要素数でもある．

4.6.3　依存部品群ツリー表示部

図 19 中の C の部分がこれに該当する．

ここでは依存部品群内の利用関係を反映させて親が子を呼び出しているツリー形式で表示される．このツリーでは，各ノードが依存部品群の各部品を表しており，また，各エッジは親ノードが子ノードを呼び出しているという利用関係を表している．

各ノードには，部品に関する以下の情報が表示されている

- メソッドの特徴メトリクス
  - LOC
  - サイクロマチック数
  - Component Rank の順位
- メソッドの完全限定名
- 外部パッケージマーク
- 重複表示マーク

4.6.4　ダウンロード部

図 19 中の D の部分がこれに該当する．

依存部品群が実際に記述されているソースコード一覧表示とそれらのダウンロードを行う．ソースコードのファイルパス名と，それに対応するダウンロード用リンクを設けている．


A
B
C
D
Classes using org.w3c.tidy.AttVal#checkAttribute(Lexer, Node)
Packages (5)
org.w3c.tidy
org.w3c.tidy.CheckAttribsImpl$CheckAREA
org.w3c.tidy.CheckAttribsImpl$CheckHTML
org.w3c.tidy.CheckAttribsImpl$CheckIMG
org.w3c.tidy.CheckAttribsImpl$CheckTABLE
org.w3c.tidy.Node
Classes (0)
Dependence Methods of org.w3c.tidy.AttVal#checkAttribute(Lexer, Node)
Total [LOC,CYC,MNUM]=[297(990%), 68(425%) ,9]
[method name] = High Cyclomatic Value
[method name] = High Component Rank Value
[method name] = High Cyclomatic And Component Rank Value
[LOC:30,CYC:16]org.w3c.tidy.AttVal#checkAttribute(Lexer, Node)
[LOC:18,CYC:8,CR:8]org.w3c.tidy.AttVal#checkUniqueAttribute(Lexer, Node)
[LOC:4,CYC:2,CR:3]org.w3c.tidy.Lexer#wstrcasecmp(String, String)
[LOC:197,CYC:30,CR:2]org.w3c.tidy.Report#attrError()
[LOC:4,CYC:1,CR:1]org.w3c.tidy.Report#tidyPrint(PrintWriter, String)
[LOC:4,CYC:1,CR:6]org.w3c.tidy.Report#tidyPrintln()
[LOC:16,CYC:6,CR:5]org.w3c.tidy.Report#tag()
[LOC:4,CYC:1,CR:1]org.w3c.tidy.Report#tidyPrint(PrintWriter, String)***
[LOC:23,CYC:3,CR:4]org.w3c.tidy.Report#position()
[LOC:4,CYC:1,CR:1]org.w3c.tidy.Report#tidyPrint(PrintWriter, String)***
[LOC:1,CYC:1,CR:7]org.w3c.tidy.AttrCheck#check()
[LOC:197,CYC:30,CR:2]org.w3c.tidy.Report#attrError()***
Dependence Files of org.w3c.tidy.AttVal#checkAttribute(Lexer, Node)
Download : All Files(.zip)
Download : /lab/space/nasfk-kobori/JavaFile/cvs.sourceforge.net/cvsroot/jedit/plugins/JTidyPlugin/org/w3c/tidy/AttrCheck.java
Download : /lab/space/nasfk-kobori/JavaFile/cvs.sourceforge.net/cvsroot/jedit/plugins/JTidyPlugin/org/w3c/tidy/Report.java
Download : /lab/space/nasfk-kobori/JavaFile/cvs.sourceforge.net/cvsroot/jedit/plugins/JTidyPlugin/org/w3c/tidy/Lexer.java
Download : /lab/space/nasfk-kobori/JavaFile/cvs.sourceforge.net/cvsroot/jedit/plugins/JTidyPlugin/org/w3c/tidy/AttVal.java

図 19: データ表示部

5　評価

本節では，提案手法を実現したシステムを用いて適用実験を行い，ケーススタディーと提案手法の有効性の評価を行う．

5.1　評価概要

オープンソースプロジェクトの任意のソフトウェアの main メソッドもしくはそれに該当するメソッド（run，execute など）に対して本手法を適用する．これは，ソフトウェアの main メソッドがそのソフトウェアの主要な機能を持っていることが予想されるため，理解すべき内容が比較的明確であると考えたためである．

5.2　ケーススタディー

5.2.1　適用例 1

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．

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 5 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 のほうが若干大規模であることがわかる．

候補部品	LOC	CYC
部品 1	74	11
部品 2	47	12

表 5: 単体の規模 (LOC：行数，CYC：サイクロマチック数)

しかし，依存部品群の総規模は，表 6 のようになっており，全体としては，逆に部品２のほうが遥かに大規模であり，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 2 のほうが大きいという結果が得られた．

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 6: 依存部品群全体の規模

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．

5.2.2 適用例 2

C や Java のソースコードを整形したり再構成するソフトウェア“ AStyle ”の main メソッドに対して本手法を適用した．適用結果のデータ表示部を図 20 に示す.

AStyle の main メソッドは，92 種のメソッドに依存していることがわかり，また総行数も 2608 行で main メソッド本体の約 15 倍の依存部品群があり，大規模であり，部品依存ツリーも非常に大きなものになった.

そこで，まずシステムがメトリクス計算をすることで重要部品と判断したものについて見てみる．ここで，ツリー表示部において高 CR ＆高 CYC なマークが付いている 5 種のメソッドを表 7 に示す.

メソッド名	機能	LOC	CYC	CR
ASBeautifier.beautify	ソースコードを整形する	918	292	20
Astyle.parseOption	ソースコードの言語を設定する	163	41	14
ASFormatter.isOneLineBlockReached	ブロックが 1 行に納まるか調べる	52	14	16
ASBeautifier.getNextProgramCharDistance	次の文字までの空白数を調べる	32	8	27
ASFormatter.peekNextChar	次の文字を読み込む	18	5	26

表 7: 高 CR，高 CYC な部品（Astyle）

これらの部品のソースコードを見ると，peekNextChar 以外の 4 つのメソッドは，それぞれが“ AStyle ”の機能（ソースの整形，再構成）を実現する上で重要な役割を担っており，その中でも beautify メソッドが主要機能であるコード整形を実際に行っていることがわかった．このメソッドは Astyle.main メソッドからは直接見えておらず，ASFormatter.nextLine というメ

ソッドを間に挟んで呼び出されている．このことから，Astyle.main → ASFormatter.nextLine → ASBeautifier.beautify という一連の処理の流れが読み取れ，行単位 (nextLine) で beautify メソッドが呼ばれて，整形が行われているということがわかる．また，beautify をみることで，どのような整形が行われているのかが分かる．

このように，重要部品周りのツリー構造を読みとることで重要処理の流れを理解することができることがわかる．

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 8 に示す．

	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
1	JEdit	237	16	10	0.625
2	JEditLogger	7	1	1	1.000
3	JGraph	49	1	1	1.000
4	Astyle	92	5	4	0.800
5	Tidy	35	2	2	1.000
6	VncViewer	102	11	7	0.636
7	GanttProject	163	12	11	0.917
8	FontChooser	3	1	0	0.000
9	Parser	94	14	11	0.786
10	BruteForceClassFinder	8	1	1	1.000
計		790	64	48	0.750

表 8: 適合率

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 8 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．

図 20: AStype.main() に対するデータ表示部

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，いくつかのケーススタディーや適用実験によって，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．

謝辞

本研究において, 常に適切な御指導および御助言を賜りました 大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上克郎教授に深く感謝いたします.

本研究において, 常に適切な御指導および御助言を頂きました 大阪大学大学院情報科学研究科コンピュータサイエンス専攻 楠本真二助教授に深く感謝いたします.

本研究において, 常に適切な御指導および御助言を頂きました 大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下 誠助手に深く感謝いたします.

本研究において, 常に適切な御指導を頂きました 立命館大学情報理工学部情報システム学科 山本 哲男講師に深く感謝いたします.

本研究において, 適切な御指導を頂きました 大阪大学大学院情報科学研究科コンピュータサイエンス専攻 特任研究員 横森励士 氏に深く感謝いたします.

最後に, その他様々な御助言を頂いた大阪大学大学院情報科学研究科コンピュータサイエンス専攻井上研究室の皆様に感謝いたします.

参考文献

[1] About, Inc., “Focus on Java”, http://java.about.com/.

[2] The Apache Software Foundation, “The Apache XML Project”, http://xml.apache.org/.

[3] The Apache Software Foundation, “The Apache Jakarta Project”, http://jakarta.apache.org/.

[4] M. H. Aviram: “Code-centric search tool strives to reduce Java development time”, *Java World*, June (1998).

[5] 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).

[6] G. Blom, L. Holst and D. Sandell: “Problems and Snapshots from the World of Probability”, *Springer Verlag*, (1994).

[7] J. C. Borda: “M’emoire sur les ’elections au scrutin”, *Histoire de l’Acad’emie Royale des Sciences*, (1781).

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

[9] C. Braun: “NATO Standard for the Development of Reusable Software Components”, *NATO Communications and Information Systems Agency*, (1992).

[10] C. Dwork, R. Kumar, M. Naor, D. Sivakumar: “Rank Aggregation Methods for the Web”, *Proc. of WWW10*, pp. 613-622 (2001).

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

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

[13] Free Software Foundation, Inc., “gettext”, http://www.gnu.org/software/gettext/.

[14] Google, Inc., “Google”, http://www.google.com/.

[15] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, S. Kusumoto: “Component Rank: Relative Significance Rank for Software Component Search”, *Proc. of ICSE25*, pp. 14-24 (2003).

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

[17] I. Jacobson, M. Griss and P. Jonsson: “Software Reuse”, *Addison Wesley*, (1997).

[18] J. Gosling, B. Joy, G. Steele, and G. Bracha: “The Java Language Specification”, *Addison-Wesley* (2000).

[19] KAKASI Project, “KAKASI”, http://kakasi.namazu.org/.

[20] 亀井, 門田, 松本: “WWW を対象としたソフトウェア検索エンジンの構築”, 電子情報通信学会技術研究報告, SS2002-47, Vol. 102, No. 617, pp. 59-64, (2003).

[21] 北, 津田, 獅々堀: “情報検索アルゴリズム”, 共立出版, (2002).

[22] 小堀, 山本, 松下, 井上: “類似度メトリクスを用いた Java ソースコード間類似度測定ツールの試作”, 電子情報通信学会技術研究報告, SS2003-2, Vol. 103, No. 102, pp. 7-12, (2003).

[23] 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).

[24] Namazu Project, “Namazu”, http://www.namazu.org/.

[25] 庭山, 松尾, 萩原, 金田: “セマンティック Web を利用したソフトウェア検索システムの提案”, 電子情報通信学会技術研究報告, SS2002-57, Vol. 102, No. 704, pp. 27-31, (2003).

[26] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, J. Riedl: “GroupLens: an open architecture for collaborative filtering of netnews”, *Proc. of the 1994 CSCW*, pp. 175-186 (1994).

[27] R. C. Seacord, S. A. Hissam, K. C. Wallnau: “Agora: A Search Engine for Software Components”, *IEEE Internet Computing*, Vol. 2, No. 6, pp. 62-70 (1998).

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

[29] Sun Microsystems, Inc., “Java2 Software Development Kit, Standard Edition 1.3.0”, http://java.sun.com/.

[30] H. Washizaki, Y. Fukazawa: “A Component Extraction-based Retrieval System for OO Program”, IPSJ/SIGSE OO 2003 Symposium (2003).

[31] VA Linux Systems, Inc., “SourceForge”, http://sourceforge.net/.

[32] 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)

[33] 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)

[34] 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)

[35] 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)