ソースコードの特徴語を用いた Javaソフトウェア部品の自動分類手法の提案

仁井谷 竜介† 松下 誠† 井上 克郎†

† 大阪大学 大学院情報科学研究科
〒 560-8531 大阪府豊中市待兼山町 1-3

ソフトウェア部品検索システムは，再利用および理解支援に有効なシステムで，様々な手法で実現されている．本研究ではこの中でもカテゴリ検索に着目する．カテゴリの作成や維持は，手作業で行うには多大な時間と手間を要するため，自動化は不可欠といえる．そこで，ソースコードの特徴語を用いてソフトウェア部品を自動分類する手法を提案する．ソースコードの特徴語とは「そのソースコードを特徴づける重要な語」のことを指し，1 つのカテゴリを形成するものとする．提案する分類手法をもとにソースコードを分類するシステムおよび，その分類結果を利用する検索システムを構築した．またその評価を行い，提案手法がソフトウェア部品の分類に有効であることを確認した．

Automatic Categorization Method for Java Components Using Feature Words of Source Code

Ryusuke Niitani† Makoto Matsusita† Katsuro Inoue†

† Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama-cho, Toyonaka, Osaka 560-8531, Japan

A software component retrieval system is one of efficient support systems for the software reuse. I focus category search systems in the retrieval systems implemented by various method. Categories are maintained by superintendent; so the cost for maintenance is very high. In this paper, I propose a method for automatic software categorization using feature words of source code. A ”feature word of source code” means the word that characterizes its source code and forms one category. Based on the proposed method, I implemented an auto-categorization system and a category search system. I also evaluated and confirmed that proposed method can categorize software adequately.

1　まえがき

近年のソフトウェアの大規模化と複雑化に伴い，ソフトウェア部品の再利用性の向上が叫ばれている．そのための技術の 1 つとしてソフトウェア検索システムがある．ソフトウェア検索システムは大きく分けて，検索者にクエリを入力させるものと，あらかじめ分類されたカテゴリから探すものの 2 つである．前者の例として SPARS-J[10, 2]，後者の例として SourceForge[6] が挙げられる．以降，後者のカテゴリ検索について考える．

カテゴリ検索の利点は，ツリー上のカテゴリに従って段階的に絞り込めることや，検索者が存在を知らない有意義なカテゴリの発見ができることにある．しかし，このカテゴリは手作業で維持されることが多く多大な時間と手間がかかる．特に，ソフトウェア部品を対象とする場合，追加あるいは更新しようとするソフトウェア部品がどのカテゴリに属するのかの判断が非常に困難である．また，対象数が多いため，手作業でそれを判断するには多くの専門知識を持った人と膨大な時間を要する．さらに，カテゴリの再構成・維持作業も大きな手間がかかる．従って，ソフトウェア部品を対象としたカテゴリ検索を効率的に行うには，カテゴリへの分類とカテゴリの維持の自動化が不可欠である．

本研究では Java ソフトウェア部品の特徴を表す特徴語を定義し，この特徴語を 1 つのカテゴリとみなしてソフトウェア部品を自動的に分類する手法を提案する．本手法では分類の入力として，ソースコード中に出現する識別子，コメント中の単語を用いる．部品ごとに，これらの全ての語に対して重みを求め，重みの大きい上位の語をその部品の特徴語と決定する．次に，得られた特徴語を 1 つのカテゴリとみなして各部品を分類する．そして，ツリー状のカテゴリ間の関係および，類似したカテゴリをまとめる関係を生成する．

さらに，提案する分類手法を実現するシステムおよび，検索システムを構築し分類結果の評価を行った．その結果，提案手法を用いたソースコードのみによる分類の有効性を確認した．

以下，2 節で既存のソフトウェア部品検索システムについて述べる．3 節でソースコードの特徴語を用いてソフトウェア部品を分類する手法について述べる．4 節で提案手法を実現したシステムについて述べる．5 節で，そのシステムの評価実験について述べる．最後に 6 節でまとめと今後の課題について述べる．

2　ソフトウェア部品検索

本節ではソフトウェア部品検索システムについて説明した後，クエリ入力検索とカテゴリ検索による部品検索の概要と，既存の研究について述べる．

2.1　ソフトウェア部品検索システム

部品の検索とは，分類された部品集合から必要な部品を探し出す作業である．自然言語文書の検索システムでは，検索しようとするトピックのキーワードをクエリに用いて検索するのが一般的であるが，ソフトウェア部品検索システムでは，キーワードの他に部品の特徴をあらわすメトリクス，記述されている言語，求める機能の抽象表現などをクエリに用いる場合もある．また，クエリを必要としないものもある．

2.1.1　クエリ入力検索システム

検索者に検索語や検索文といったクエリを入力させ，そのクエリとの適合度の高いものを検索するシステム．適切なクエリを入力すれば検索者の意図通りの結果が得られるのが利点である．逆に，適切なクエリを入力しないと，意図したものが検索されなかったり，意図しないものが検索される問題点がある．

2.1.2　カテゴリ検索システム

クエリを入力せずあらかじめ用意されたカテゴリから対象を探す検索システム．段階的に絞り込めることが利点である．また，検索者はその存在を知らないカテゴリなど意図しない発見ができる．カテゴリの維持は手作業でされることが多いため，多大な時間と手間がかかる．

2.2　関連研究

我々の研究チームでは，Java のソースコードを対象としたソフトウェア部品検索システム SPARS-J[10, 2] の構築を行っている．SPARS-J は，依存や類似といったソフトウェア部品特有の特性を考慮しながら，大規模なライブラリの構築を行う．検索語とその出現箇所を検索クエリとした全文検索

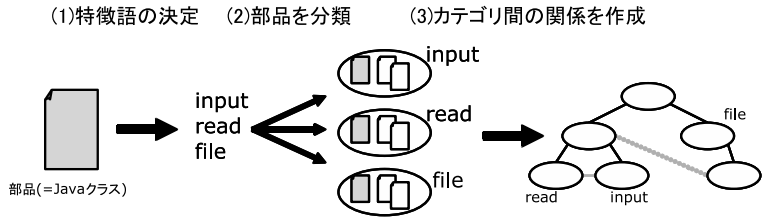

図 1: 提案手法の概要

を行い，部品に対する検索語の重みを出現箇所と TF-IDF 法 [8] から求める KR 法と，部品の重みをその利用実績から求める CR 法の 2 つの手法によって検索結果を順位付けている．

ソフトウェアを分類し，ライブラリとしての再利用を支援する研究も行われている．Börstler は，コンポーネントの性質や属性からソフトウェアの特徴を抽出し，それに基づいて分類分けを行う手法 FOCS[1] を提案した．コンポーネントはいくつかの特徴から構成される．

鷲崎らの JComponentSearch[9] は分類システムではないが，依存関係に基づく分類システムという見方もできる．これはブラックボックス的な再利用を支援する．

ソフトウェア部品ではなくソフトウェアを対象とした分類手法として，川口の LSA に基づく手法 [4] が挙げられる．これはソフトウェアのソースコードに出現する識別子に対して LSA[5] を適用し，その結果を用いて識別子によるソフトウェアの非排他的クラスタリングを行っている．これにより，前提知識を用いずにソフトウェアの集合を機能やライブラリやアーキテクチャといった，複数の視点による自動分類を実現している．

ソフトウェアシステム間の関連を対象にした研究として，山本らの SMMT[7] が挙げられる．SMMT は 2 つのソフトウェアシステムの類似度を計測する．SMMT ではコードクローン検出技法 [3] に基づいてソースコード行単位での類似度を測定する．しかし，SMMT での類似度の定義は，同じソースから派生したようなソフトウェア間の類似度は適正な値が出力されるが，全く出自が異なるソフトウェア間では類似度の値が極端に低くなる．

3 提案手法

本研究では，ソフトウェア部品をカテゴリに自動的に分類し，得られたカテゴリ間に関係を作成する．

提案する手法の概要を図 1 に示す．本手法は，(1) ソースコードを入力として部品ごとに特徴語を決定し，(2) 得られた特徴語を 1 つのカテゴリとみなして各部品を分類する．(3) ツリー状のカテゴリ間の関係および，類似したものをまとめる関係を生成するという 3 つの手順で分類を行う．

3.1 用語

本提案手法における用語の定義を与える．

部品・ソフトウェア部品 Java のクラス．

特徴語自然言語を対象とした分類，検索，データマイニング関連で使われる言葉.「文書の特徴を表す語」という共通の意味で使われる．特徴語は，単に文書中に出現した重要な語を指す場合と，その文書中に出現したかに関わらずその文書を特徴づける重要な語の 2 つに大きく分けられる．前者は検索対象の語の数を削減するためや，要約の一種として検索者に見せるために用いられる．後者は前者と同様の目的以外に「文書中に出現しない重要な語の付加」のために用いられる．

本研究では，後者の意味をソースコードに導入した形で用いる．つまり，その語がソースコードに出現したかに関わらず部品を特徴づける語を特徴語と定義する．特徴語の集合は，全部品のソースコードのいずれかに出現する語全ての集合に含まれるものとする．

複合語1 つ以上の語が繋がってできている語．例えば，XMLParser は「XML」と「Parser」の 2 つからなる複合語.

単一語複合語の最小単位．例えば，XMLParser では「XML」と「Parser」.

部分語複合語を単一語の切れ目で区切ったときの一部，あるいは全部．例えば，XMLParser の部分語は,「XML」と「Parser」と「XMLParser」.

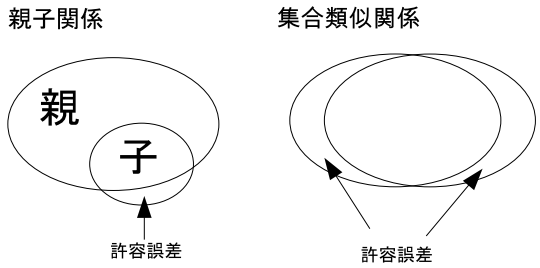

図 2: 親子関係・集合類似関係

単語重み特徴語を決定するための重み．全ての部品に対して単語ごとに決定される．

語の出現重み語の種類（表 1）を考慮した重み．

表 1: 語の種類

クラス定義名	メソッド定義名
インタフェース名	パッケージ名
インポートパッケージ名	呼出しメソッド名
参照フィールド変数名	生成したクラス名
変数の型名	参照する変数名
コメント	文書コメント
行末コメント	文字列リテラル

カテゴリ共通の特徴語を持つ部品の集合．部品に対して複数個の特徴語が決定されるので，ソフトウェア部品はカテゴリによって非排他的に分類されることになる．

カテゴリ間の関係本研究ではカテゴリ間の関係を親子関係，集合類似関係，類似関係の 3 つの関係と定義する．

- **親子関係**カテゴリを部品の集合としてみたときに，ある程度の誤差を許容して集合の包含関係が成り立つとき，その包含する方を親，包含される方を子と定義する（図 2 左）．誤差については後述する．
- **集合類似関係**カテゴリを部品の集合としてみたときに，ある程度の誤差を許容して集合の同値関係が成り立つとき，その間に集合類似関係があると定義する（図 2 右）．
- **類似関係**カテゴリの特徴語同士に高い類似度が見られるものにこの関係があると定義する．

カテゴリ木カテゴリ間の親子関係に基づく，カテゴリをノードとするツリー．ただし，子の重複を認める．

3.2 特徴語決定

特徴語を決定するために，まず部品ごとに語に対する重み（単語重み）を求める（図 3）．得られた単語重みの高い語を部品の特徴語とする．ここでは上位 10 語を特徴語とする．

3.2.1 語の出現重み

$Kind$ を語の種類，KW_k を種類 k に対応する重み，$Count_{cwk}$ を部品 c における語 w の種類が k として出現した回数と表すとき，部品 c に対する語 w の出現重み (WW_{cw}) は次の式で表される．

$$WW_{cw} = \sum_{k \in Kind} KW_k \times \log(1 + Count_{cwk})$$

回数の対数をとっているのは，同じ語が何度も使われたときに，重みが高くなりすぎることを防ぐためである．

3.2.2 語の記法統一と複合語による重み再計算

コーディングスタイルなどの違いによる語の表記の違いの統一と，部分語に対する重みの配分計算を同時に扱う．再計算後の部品 c に対する部分語 w' の重み ($WW'_{cw'}$) は以下の式で与えられる．

$$WW'_{cw'} = \sum_{w \in W} DW_{ww'} \times WW_{c'w}$$

W は処理前の全ての語の集合．$DW_{ww'}$ は語 w に対する部分語 w' の関係の重みで以下の式で与えられる．

$$DW_{ww'} = \left(\frac{count(w')}{count(w)} \right)^2$$

$count(w)$ は語あるいは部分語 w を構成する単一語の数．例えば RemoveElementAt という語の重みは，ElementAt に $\frac{4}{9}$，Element に $\frac{1}{9}$ 倍して加算することになる．

また，elementAt や element_at といった表記の違う語を Element と At の複合語として同じものとみなし，表記の違いを統一する（大文字小文字の違いは無視する）．

3.2.3 対象語の削減

計算量の軽減のために，分類に影響のない語を削除する．1 つの部品にしか出現しない語は，以降の手法にとって有益な情報ではないので削除対象としている．

3.2.4 利用関係による重み加算

部品間の利用関係を語の重みに反映するために，利用関係による重み加算後の部品 c に対する語 w

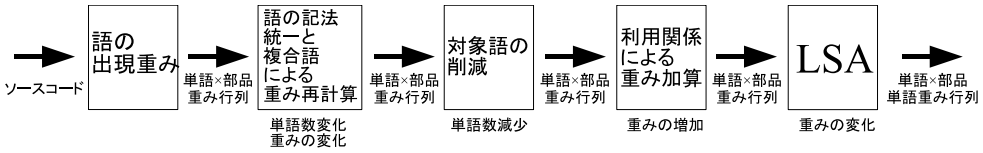

図 3: 単語重み計算の流れ

表 2: 利用関係の種類

種類	概要
継承	子クラスが親クラスを利用
抽象クラス実装	実装クラスが抽象クラスを利用
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
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
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
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
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

の重み（$WW'cw$）を求める．

$$WW'_{cw} = WW_{cw} + \sum_{c' \in C, c' \neq c} RW_{cc'} \times WW_{c'w}$$

ここで C は全部品の集合．$RW_{cc'}$ は c と c' の関係の重みである．

Java のようなオブジェクト指向言語では，親クラスに実装のほとんどが書いてあり，子クラスにはその差分だけが書いてある場合が多い．そのため，対象のクラスの語を見るだけに比べ，親クラスの語にもある程度の重みを与えて加えた方が，有益な特徴語の決定に結びつくと考えられる．同様の理由から，メンバ変数として持つクラスなどといった関係のあるクラスの語にも重みを与えている．

なお，部品間の利用関係を表 2 に示す．これは SPARS-J で用いられているものと同じである．

3.2.5 LSA

LSA は類似度を求めるための統計的手法である．LSA により，出現の傾向が類似する語同士に似たような重みがつくような補正をかける．

表 3: LSA で得られたベクトル

語＼文書	A	B	C	D
foo	1.23	3.56	5.67	7.89
bar	-4.12	15.2	3.89	0.23

例として 4 つの文書 A，B，C，D と 2 つの語 foo，bar について考える．表 3 は文書 A，B，C，D に対する foo と bar の LSA 適用後の重みを表す．通常 LSA は「(foo) = (1.23, 3.56, 5.67, 7.89)」の様に foo と bar をベクトルとして用いるが，本手法では,「(foo に対する A の重み) = 1.23」の様に，ベクトルの要素をその語に対する重みとして利用する．

3.3 カテゴリ間の関係生成

この節では，カテゴリに親子関係，集合類似関係，類似関係を与える手法について述べる．

3.3.1 包含関係によるカテゴリ木・関係生成

カテゴリは部品の集合なので，それぞれに包含関係が存在する．従って，包含するものを親，包含されるものを子とすると，ツリー状の関係を生成することができる．また，同じ集合をなすカテゴリを同値の関係とすることができる．

本研究では，包含関係が完全に成り立っていなくても，ある程度の誤差を許容して上記の親子関係と同値関係が成り立ったものとみなして，それぞれ親子関係，集合類似関係を生成する．

3.3.2 特徴語間の類似度による関係生成

特徴語（カテゴリ）同士の類似度は，特徴語決定に用いた行列から求めることができる．その類似度を用いて，類似度が一定以上のカテゴリ同士に類似関係を生成する．

類似度はコサイン尺度で求める．コサイン尺度は以下の式で表される．

$$Sim_{ab} = \cos\theta = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \sqrt{\sum_{i=1}^{n} b_i^2}}$$

Sim_{ab} は特徴語 a，b 間の類似度．$\vec{a}$ は特徴語決定に用いた行列における a の行（あるいは列）．$\vec{b}$ は特徴語決定に用いた行列における b の行（あるいは列）．

3.3.3 誤差の許容

親子関係A の要素の 8 割以上が B に含まれたら，A が子で B が親とする．

集合類似関係A ∩ B の要素が A，B にそれぞれ 8 割以上含まれていたら成り立つとする．

3.3.4 カテゴリ間の関係の優先順位

同時に複数の関係が成り立つ場合があるので，以下の優先順位によって 1 つに決める．

(1) 集合として同一なら集合類似関係
(2) 包含関係なら親子関係
(3) 成り立っていれば集合類似関係
(4) 成り立っていれば親子関係
(5) 成り立っていれば類似関係

4 自動分類システム

本研究で開発したシステムは大きく分けてソースコードの解析部，単語重み計算部，特徴語決定部，カテゴリ間の関係決定部，検索部の 5 つに分けられる（図 4）．

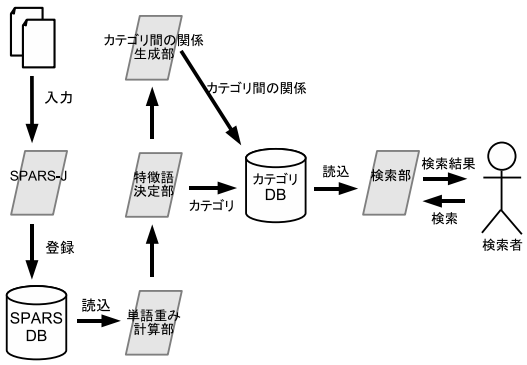

図 4: システム構成図

4.1 ソースコード解析部 (SPARS-J)

本研究では Java ソースコードの解析結果を SPARS-J のデータベースから取得する．クラス名やメソッド名といった語の抽出や，各部品間の利用関係を利用する．

4.2 単語重み計算部

3 節で説明した，語の重み計算・利用関係による重み加算・語の記法統一と複合語による重み再計算・対象語の削減・LSA の適用を行う．

4.3 特徴語決定部

部品ごとに重み順で語をソートし，上位 10 語をその部品の特徴語とする．

4.4 カテゴリ間の関係生成部

包含される要素数の割合，$WW_{cw}^{T} \times WW_{cw}$ で求められる特徴語間の類似度を用いてカテゴリ間に関係を生成する．

4.5 検索部

分類結果を評価するための検索システム．カテゴリ，部品の検索を行う．

5 実験

4 節で述べたシステムを用いて実際に部品の分類を行い，分類結果の評価を行った．

5.1 実験対象

ロボットシステム（Robocode のロボット）の部品 245 クラス（35 システム）を分類対象とした．

5.2 実験方法

実際に分類を行い，どれだけ適切にカテゴリが作られているかを評価する．また，ソースコード中にはない特徴語がどれだけ特徴を表しているかを評価する．

評価は (1) カテゴリに属する部品の適合率，(2) 部品が属するカテゴリの適合率，(3) ソースコード中にはない特徴語の適合率の 3 つの値によって行う．

適合率 (precision) とは，検索結果の中でどれだけ適合するものがあるかを表し，1 に近いほど良い結果であることを示す．

検索結果を評価するには再現率 (recall) も併せて用いることが多い．再現率を求めるには，部品ごとに「特徴語であるべき語」の一覧を作らなければいけないが，何が特徴語であるかを事前に決定することが困難であるため，今回の実験では用いない．

5.2.1 評価に用いる適合率

$$\text{(カテゴリ C に属する部品の適合率)} = \frac{|in(C) \cap match(C)|}{|in(C)|} \quad (1)$$

$$\text{(部品 c が属するカテゴリの適合率)} = \frac{|Belong(c) \cap Match(c)|}{|Belong(c)|} \quad (2)$$

$$\text{(ソースコード中にはない特徴語 C の適合率)} = \frac{|nothave(C) \cap in(C) \cap match(C)|}{|nothave(C) \cap in(C)|} \quad (3)$$

C 特徴語、または特徴語 C が形成するカテゴリ
c 部品
in(C)カテゴリ C に含まれる部品
match(C)カテゴリ C に適合する部品
nothave(C)ソースコード中に特徴語 C がない部品
Belong(c)部品 c が属するカテゴリ
Match(c)部品 c に適合するカテゴリ

5.2.2 カテゴリ・部品間の適合の条件

カテゴリに部品が適合する条件とは，カテゴリに対応する特徴語が部品のソースコードもしくは，親クラスのソースコードの特徴を表すことである．

ただし，以下のようなものは適合としない．

- そのカテゴリに含まれる部品の集合が同様のものであるが，そのうちのどれかを表すだけの特徴語がそのカテゴリに対応している
  例：Read というカテゴリ中に，FileReader と FileWriter が入っている
- ソースコード中に現れるがあまり特徴を表さない
  例：ループ制御変数 i
- 自身の特徴を表さないような子クラスの特徴語
  例：FileWriter の子クラスが FileURLWriter で FileWriter の特徴語に URL が入っている

5.3 結果

図 5 は式 1 の各カテゴリに属している部品の適合率で，縦軸が適合率を表し，各縦棒が 1 つのカテゴリに対応する．横軸はカテゴリを適合率でソートして並べている．横軸の数はカテゴリの数を表し，横軸に平行な点線は適合率の平均を表している．適合率の平均は 0.8460 であった．

図 6 は式 2 の各部品が属しているカテゴリの適合率で，各カテゴリに属している部品の適合率の図 5 とは，部品とカテゴリの関係が逆転している．従って，各縦棒が 1 つの部品に対応し，横軸は部品を適合率でソートして並べている．横軸の数は部品の数を表す．適合率の平均は 0.8633 であった．

図 7 は式 3 のソースコード中にはない特徴語の適合率で，縦軸，横軸，横軸に平行な点線は図 5 と同様である．適合率の平均は 0.4933 であった．

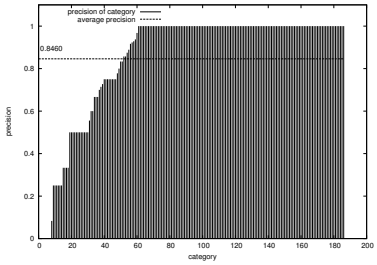

図 5: 各カテゴリに属している部品の適合率

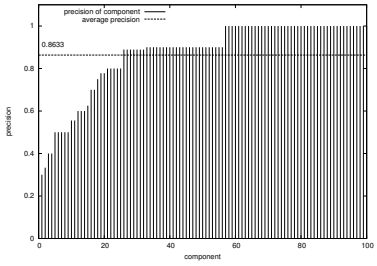

図 6: 各部品が属しているカテゴリの適合率

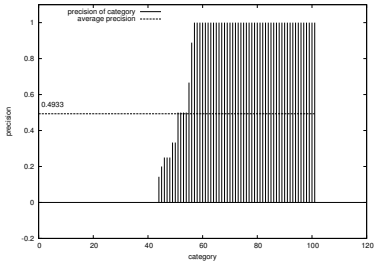

図 7: ソースコード中にはない特徴語の適合率

5.4 考察

各カテゴリに属している部品，各部品が属しているカテゴリの適合率は，0.85 前後の高い値を示している．従って，本手法による分類結果は適切であると言える．

しかし，カテゴリに属している部品の中で，適合率が 0 のカテゴリがいくつか発見された．具体的には，Javadoc タグ (@param,@author など)，HTML タグ (br,li など)，前置詞，助詞，代名詞といったもので，ドキュメントコメント中に繰り返

し出現していたのが原因と考えられる．this は代名詞以外にも，Java の予約語として使われているが，他の識別子と区別をしなかったため，いくつかの部品では特徴語となっていた．

特徴語のカテゴリに含まれる部品が，全てソースコード中にその特徴語を持っていた場合，分母が 0 になり式 3 の適合率が計算できない．このようなカテゴリの，全カテゴリに対する割合は 0.5677 であり，多くを占める．これより，特徴語の多くはソースコード中に含まれていることがわかる．

また，図 7 のソースコード中にはない特徴語の適合率の平均は 0.5 近くであり，部品対カテゴリの適合率に対し，低い値となった．この原因は，特徴語が 10 個固定である点にあると考えられる．複雑な部品では特徴語の数は 10 個では足りず，高い適合率を得られたものの，特徴語のほとんど，あるいは全てがソースコード中に出現していた．逆に，単純な部品では特徴語の数は 10 個では多すぎ，その中にいくつもの無関係な特徴語が含まれたために，適合率は低い値となっていると考えられる．

6 まとめ

本研究では，ソフトウェア部品に対して特徴語を決定し，それを用いてカテゴリ木を構成する分類手法の提案と実装を行った．また実験を行い，本手法を用いることにより対象部品群に対する詳細な知識がなくとも自動的に分類が行えることを示した．

今後の課題としては，まず，部品ごとの特徴語の数が 10 個に固定であることを解決するために，適切な数を調査することが上げられる．次に，その数の取得を自動化すること，不適切な特徴語の排除方法の考案が挙げられる．また，SPARS-J との連携，利用関係の重みといったパラメータのよりよい決定方法の模索や，より大規模な実験，そして実行速度およびスケーラビリティの向上が挙げられる．

参考文献

[1] Börstler, J.: Feature Oriented Classification for Software Reuse, *Proceedings of the 7th International Conference on Software Engineering and Knowledge Engineering*, Skokie, IL, pp. 204–211 (1995).

[2] Inoue, K., Yokomori, R., Yamamoto, T., Matsushita, M. and Kusumoto, S.: Ranking Significance of Software Components Based on Use Relations, *IEEE Transactions on Software Engineering*, Vol. 31, No. 3, pp. 213–225 (2005).

[3] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: A Multi-Linguistic Token-based Code Clone Detection System for Large Scale Source Code, *IEEE Transactions. Software Engineering*, Vol. 28, No. 7, pp. 654–670 (2002).

[4] Kawaguchi, S., Garg, P. K., Matsushita, M. and Inoue, K.: MUDABlue: An Automatic Categorization System for Open Source Repositories, *The Proceedings of the 11th Asia-Pacific Software Engineering Conference(APSEC2004)*, Busan, pp. 184–193 (2003).

[5] Landauer, T. K., Foltz, P. W. and Laham, D.: An Introduction to latent Semantic Analysis, *Discourse Processes*, Vol. 25, pp. 259–284 (1998).

[6] SourceForge: . `http://sourceforge.net/`.

[7] 山本哲男，松下　誠，神谷年洋，井上克郎：ソフトウェアシステムの類似度とその計測ツール SMMT，電子情報通信学会論文誌， Vol. J85-D-I, No. 6, pp. 503–511 (2002).

[8] 北　研二，津田和彦，獅々堀正幹：情報検索アルゴリズム，共立出版 (2002).

[9] 鷲崎弘宜，深澤良彰：オブジェクト指向プログラムのためのコンポーネント抽出型検索システム，情報処理学会オブジェクト指向シンポジウム 2003 論文集，pp. 77–84 (2003).

[10] 横森励士，梅森文彰，西　秀雄，山本哲男，松下　誠，楠本真二，井上克郎：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， Vol. J87-D-I, No. 12, pp. 1060–1068 (2004).