特別研究報告

題目

Java ソフトウェア部品検索システムのための索引付け手法の提案と実装

指導教官
井上 克郎 教授

報告者
梶田 泰伸

平成 16 年 2 月 19 日

大阪大学 基礎工学部 情報科学科

平成 15 年度 特別研究報告

Java ソフトウェア部品検索システムのための索引付け手法の提案と実装

梶田 泰伸

内容梗概

近年のソフトウェア開発現場においては，再利用を用いた開発が頻繁に行われている．そのため，ソフトウェアの再利用に有益な情報を提供する検索システムを実現することにより，効率のよい再利用が可能になる．私の所属する研究グループでは，Java ソフトウェア部品を対象とした検索システム SPARS-J の開発に取り組んでいる．SPARS-J では，検索キーによる全文検索を効率的に行うため，あらかじめ各部品を索引付けして索引語をデータベースに登録する．しかし，索引語をそのままの形で登録しているので，検索語がある部品に出現していても，複合語として登録されていた場合や，語形が一致しない場合にその部品が検索できない．また，不要語もそのまま登録してしまうため，データベースの容量が無駄となる．

そこで本研究では，Java ソフトウェア部品検索システムのための索引付け手法を提案する．本手法では，Java ソースコードの特徴を生かして索引語の拡張を行う．その結果，複合語として登録されている語や，語形が変化している語を含む部品を検索できるようになり，検索洩れを減らすことができる．また，Java ソースコードにおける不要語を定義し，索引語から除外する．その結果，索引語を拡張したために起こるデータベースの容量の増加を抑えることができる．また提案した索引付け手法を SPARS-J に実装した．そして単純な索引付けとの比較実験を行った結果，検索洩れを大幅に減らすことができ，またデータベース容量の増加が抑えられていることを確認した．

主な用語

索引付け (Indexing)
索引語 (Indexing term)
不要語 (Stop word)

目 次

1　まえがき

ソフトウェアの大規模化と複雑化に伴い，高品質なソフトウェアを一定期間内に効率良く開発することが重要になってきている．これを実現するために近年のソフトウェア開発において，再利用を用いた開発が頻繁に行われている．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利用することを指し，開発期間の短縮や品質向上が期待できるといわれている [3, 4, 11, 13]．ソフトウェアの再利用による効果を最大限に引き出すためには，開発者が開発しようとするソフトウェアに必要な部品およびライブラリに関する知識を持つことが重要であるが，知識の共有が十分にされていないために，同種のプログラムが別々の場所で，独立して開発されている事も多い．

一方でインターネットの普及により，SourceForge [27] などのソフトウェアに関する情報を交換するコミュニティが誕生し，大量のプログラムソースコードが簡単に入手できるようになった．これらの公開されている大量の部品の中から，開発者の必要としている機能を持つ部品，その機能の使い方を示している部品のような，再利用に有益な情報を提供する検索システムを実現する事で，知識の共有が実現でき，再利用を促進する事ができる．

そこで私の所属する研究チームでは Java を対象としたソフトウェア部品検索システム SPARS-J (*Software Product Archive, analysis and Retrieval System for Java*) の開発を行っている．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ソースコードの分類を自動的に行う検索システムであり，クラス，またはインターフェイスの Java ソースコードをソフトウェア部品として検索を行いユーザに提供する．このシステムは検索キーによる全文検索を行うが，全文検索を検索時に行うと検索結果を得るために多大な時間を要してしまう．そこで検索時の高速化を目的として，あらかじめ各部品を索引付けして，索引語をデータベースに登録している．しかし索引語をそのままの形でデータベースに登録すると，複合語や語形が変化している語もそのまま登録される．検索キーは一般的に原形で入力されるので，これらの語は検索キーと一致せず，検索洩れを引き起こす原因となる．また，部品中には記号など部品の特徴を表すとはいえない語が存在し，これらの語を登録するとデータベースの容量に無駄が生じる．

そこで本研究では，Java ソフトウェア部品検索システムのための索引付け手法の提案と実装を行う．Java のソースコード中には，実際に実行する命令を記述するプログラム部と，その実行されるプログラムの注釈を記述するコメント部が存在し，それぞれは異なる形式で記述されている．よって提案する手法では，それぞれに対して異なる手法で索引語の拡張を行う．これにより，複合語や語形が変化している語を含む部品を検索できるようになり，検索洩れを大幅に減らすことができる．また，Java ソースコードにおける不要語を定義し，索引語から除外する．その結果，索引語を拡張したために起こるデータベース容量の増加を抑

えることができる．

以降，2 節で SPARS-J について説明し，3 節で関連技術として既存の索引付けの手法を述べ，次に関連システムとして既存のソフトウェア部品検索システムについて述べる．そして 4 節で提案する手法の内容を説明し，5 節で提案手法の評価をする．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 SPARS-J

2.1 SPARS-J の概要

現在私の所属するチームで研究している SPARS-J*(Software Product Archive, analysis and Retrieval System for Java)*[18, 26] とは，Java 言語 [7] 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．このシステムではソフトウェア部品の単位を Java の一つのクラス・インタフェースのソースコードとしており，ユーザが検索キーを指定すると検索キーと一致する索引キーが含まれるソフトウェア部品を検索する．検索結果はキーワードの出現頻度による順位付け手法 KR 法と，利用関係に基づくソフトウェア部品重要度評価手法 [10, 29]CR 法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．

以下で，SPARS-J の順位付けで使われている CR 法と KR 法をを紹介する．

Component Rank 法 私が所属する研究チームはこれまでに，利用関係からソフトウェア部品の利用実績を測定し，順位付けする手法 (*Component Rank* **法，***CR* **法**) を提案している [10, 29]．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．また、CR 法による順位付けを CR と呼ぶ．

Keyword Rank 法 索引語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．抽出された索引語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語の重み付けである．索引語の重みを利用することによって，同じ索引語を含む部品でもその索引語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，その索引キー中の索引語の重みの総和を検索キーと部品の適合度とする．SPARS-J における索引語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 [14] を用いる．TF-IDF 法は任意の部品中における特定の索引語の出現頻度 *TF*(*Term Frequency*)，および特定の索引語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語と高い索引語を差別化し，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語の分布について考慮するためのもので，ある索引語が，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に順位付けすることを，CR 法に対して *KR* **法**(*Keyword Rank* **法**) と呼ぶ．また，測定した適合度の値を KR 値と呼び，KR 法による順位付けを KR と呼ぶ．

2.2 SPARS-J の構成

以下，SPARS-J の構成を図 1 に示し，それぞれについて説明を行う．

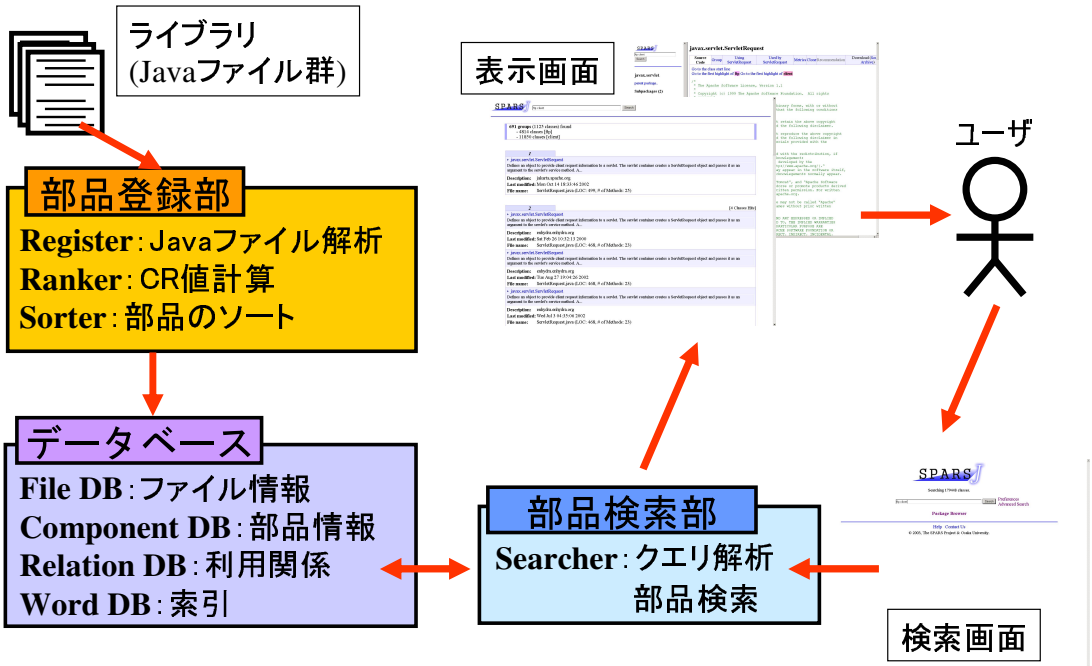

図 1: SPARS-J の概観図

2.2.1 ライブラリ

ライブラリには，検索対象とする Java ソースファイルを蓄積する．部品登録部では，このライブラリ内の Java ソースファイルを解析する．また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，その詳細を表示する．

2.2.2 部品登録部

部品登録部は以下の 3 つのモジュールから構成される．

- Register ： ライブラリのファイルを解析して，部品の切り出しの他，索引キー，メトリクス，利用関係の抽出，さらに類似度の測定と部品群化を行う．また，ファイル，部品，部品群，索引キーへ ID を割り振り，正引き，逆引き索引を作成する．
- 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 ： 検索時の高速化のために，CR 値によって部品群を降順に並びかえ，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．

2.2.3 データベース

部品登録部で解析したあらゆるデータを保存するデータベース．実装には BerkeleyDB[22] を用いた．データベースは以下の情報を管理している．

- File DB ： ライブラリ中のファイル情報を管理する DB.
- Component DB ： 登録された部品に関する情報を管理する DB.
- Relation DB ： 部品間の利用関係を管理する DB.
- Word DB ： 部品中に出現した全索引キーとその ID を格納する DB．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．

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．

2.2.4 部品検索部

クエリの解析，部品の検索，データ表示を行う Searcher で構成される．ユーザが指定した検索キーをもとにデータベースの検索を行い，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ユーザに対して提供する．

2.3 本研究との接点

2.2.2 節でも述べたように現在私の所属する研究チームで開発している SPARS-J では, 検索キーの指定によるキーワード検索を実現するために，事前に対象の Java ソースコードか

ら索引キーの抽出を行いデータベースに保存している．索引キーとは索引語とその語の種類の組のことであり，索引語の詳細については 3.1 節で説明する．検索キーと一致する索引キーを探し出すことにより，キーワード検索を実現しているため，索引語の抽出方法により検索される部品が大きく異なってくる．よって索引付けの方法によっては，検索時の検索洩れを減らすことができる．

3　関連技術と関連システム

これまでに自然言語を対象とした多くの検索システムが開発されている．そこで本研究の関連技術として，これらの検索システムの索引付けで用いられる技術を 3.1 節で紹介する．

次に関連研究として，3.2 節で SPARS-J に関連するソフトウェア部品検索システムを紹介する．その後，それらの検索システムの中で最も関連の深いと思われる，オブジェクト指向プログラムを対象とした検索システムについて，概要と本研究との接点について述べる．

3.1　索引付けに関する技術

検索システムにおいては，対象の文書の内容を文書中に含まれる単語の集合で近似するということがよく行われる．しかし，文書に含まれている単語全てが文書の内容と関係しているわけではない．たとえば，日本語の場合には助詞や助動詞，英語の場合には冠詞や前置詞などは，文書の内容と一般に無関係である．したがって，文書を単語の集合で近似する場合には，文書の内容を特徴付ける単語のみを抽出することが重要となる．このような文書の内容を特徴付けるうえで，重要な単語のことを**索引語** (indexing term) と呼ぶ．また，文書中から索引語を抽出する処理を**索引付け** (indexing) あるいは**インデキシング**と呼ぶ．[14]

以下，多くの検索システムで用いられる索引付けに関する技術について述べる．

- **単語の同定**

  索引語として用いられるものは単語である．したがって，文書から索引語を抽出するためには，まず文書を構成する文字列のどの部分が単語であるかを同定する必要がある．英語やフランス語などの多くのヨーロッパ系の言語では，単語が空白により区切られているために単語の同定は容易だが，日本語などの単語を空白で区切らない言語では単語の同定をすること自体が難しい．ここで，文書から単語を正確に抽出するためには，**形態素解析** (morphological analysis) と呼ばれる技術を用いる必要がある．形態素解析とは，文書を構成する文字列を単語に分割し，各単語に品詞や語形変化などの情報を与える処理のことである．

- **単語以外のインデキシング単位**

  単語には，**複合語** (compound word) が存在する．複合語とは，単語のうち，意味・語形の上から二つ以上の単語の結合によってできたと認められる語のことである．複合語を単語に分割するための手法としては，文字種を考慮する手法がある．日本語では，漢字，ひらがな，カタカナといった文字種を考慮して単語を分割するといったものである．例えば「検索システム」という複合語の場合,「検索」と「システム」として索引付けされる．

- **不要語の除去**

  高い頻度で出現する単語は，文書を特定する能力が低く，索引語として適当でない．このような単語のことを**不要語** (stop word) と呼ぶ．索引語の候補から不要語を除去することにより，索引語の総数を減らすことができるため，検索システムの処理の効率化や高速化，あるいは計算機資源の削減を行うことができるといった利点がある．

  どのような単語を不要語と認定するかについては，単語の種別に基づくものがある．自然言語における単語は，大きく内容語 (content word) と機能語 (function word) の２つに分けることができる．内容語は，それ自体で意味を持った，ある特定の概念を表している単語であり，機能語は，単語と単語の間の関係を表している単語である．一般に，機能語は文書を特徴付ける上で訳に立たないため，不要語とすべきであると考えられている．機能語かどうかの判別は品詞情報から行うことができる．

3.2 関連システム

Java ソフトウェア部品検索システムとして SPARS-J を紹介したが，その他にもソフトウェア部品の検索のために様々なシステムが提案されている．

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う形式の検索システムに [21][2][12] がある．Agora[21] 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うシステムであり，CORBA オブジェクトである ORB や JavaBeans が検索可能である．jCentral[2] 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り，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．また、亀井ら [12] 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．

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[16] や Web ページ検索システム Google[6] などが存在する．

これら以外には，鷲崎ら [28] が提案している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の検索システムがある．このシステムは Java ソースコードを対象としており，SPARS-J にもっとも関連が深い．

そこで，以降このシステムについて概要を説明する．

3.2.1 コンポーネント抽出型検索システムの概要

鷲崎らの検索システム [28] は，オブジェクト指向プログラムのソースコード集合について，プログラム中のオブジェクト指向クラス間の依存関係を解析し，独立して再利用可能な部分を特定してコンポーネント化し，得られたコンポーネントをデータベースに格納して検索および試行可能とするものである．このシステムは，解析対象として Java 言語で記述されたプログラムを対象としており，抽出するコンポーネントの形式として，Java 言語上のコンポーネントアーキテクチャである JavaBeans[8] を対象としている．

このシステムは次の機能を持つ．

1. ソースコード解析
2. コンポーネント抽出
3. コンポーネント蓄積
4. コンポーネント検索
5. コンポーネント試行
6. コンポーネント取得

これらの機能によって，検索者は Java プログラム群から，要求に関連するプログラムを検索し，その場で Web ブラウザを介して試行することで要求への適合性を判断し，単独で再利用可能なコンポーネントの形式で取得することができる．

3.2.2 コンポーネント抽出型検索システムと本研究の接点

このシステムと本研究との接点は，3.2.1 節の「3. コンポーネントの蓄積」にある．

このシステムでは，Java ソースコードを解析して抽出したコンポーネントの情報をデータベースに蓄積しており，さらに検索のためのインデキシング処理として，抽出コンポーネントのソースコードについて以下の単語情報を，各種類毎にその出現回数とともに取得している．

- 内部で利用するクラス・インターフェイス名
- 構成する定義クラス・インターフェイス名
- 構成クラス・インターフェイス中の定義メソッド名
- 構成クラス・インタフェース中の定義フィールド名

- 構成クラス・インタフェース中の定義変数名
- コメントを空白で分割した単語の集合

3.3 現状の問題点

3.2 節の関連研究で，オブジェクト指向プログラムを対象としたコンポーネント抽出型検索システムを紹介した．この検索システムでは 3.2.2 節で説明したように，ソースコード内に含まれる単語を取り出して，その単語をそのままの形で索引語としている．しかしこの方法で抽出した索引語の中には，不要語や複合語等が含まれている可能性がある．それらの不要語をデータベースに登録することで計算機資源の無駄となったり，また検索システムの処理の効率の低下となるおそれがある．また複合語等をそのまま索引語とすることで，検索語と索引語が一致する確率が低下することも考えられる．

そこで，3.1 節の関連技術で紹介した，検索システムにおける索引付けに関する手法を導入することを考える．これによって記憶容量の無駄を省くことができ，また検索洩れを減らすことができる．しかしこれらは自然言語を対象とした索引付けの手法であるため，不要語や複合語の定義が Java ソースコードを対象とした検索システムにおける定義と一致しないため，効果が半減すると考えられる．

4　提案する手法

3 節で関連技術と関連研究，そして現状の問題点を説明した．このような背景から，本研究では Java ソフトウェア部品検索システムのための，Java ソースコードの特徴を生かした索引付けの手法を提案する．

以下で，その詳細について説明する．

4.1　方針

検索システムにおいて検索洩れは大きな欠点となる．しかし，検索キーが意味する語が索引キーとして登録されているにも関わらず，語形が違うという理由で検索されないという事態がよく起こる．このような検索洩れの事態を回避するために，本手法では索引語の拡張を行う．また，索引語を拡張すれば拡張した索引語を保存しなければならないため，データベースの容量が増加してしまう．そこで Java ソースコードにおける不要語を定義し，索引語から除去することにより，データベースの容量の増加を抑えることを試みた．

まとめると，提案手法では以下の 2 つのことを行う．

- **索引語の拡張** ：検索洩れを減らすことが目的
- **不要語の除去** ：索引語拡張の際のデータベース容量の増加を抑えることが目的

適合部品とは検索されるべき部品であり，ここでは

1. 検索キーが出現する部品
2. 検索キーが，語形変化して出現する部品

と定義する．検索洩れとは適合部品が検索されないことを指す．

以降で提案手法の説明を行う．まず Java ソースコードから語を抽出する方法を説明し，次に不要語の除去について説明し，最後に索引語の拡張について説明する．

4.2　語の抽出

ある文書から索引語を抽出するためには，まず文書を構成する文字列のどの部分が語であるかを同定しなければならない．これは，対象文書が Java ソースコードであっても同様である．よって Java ソースコードの構文に従った字句解析を行い語を抽出する必要がある．字句解析とは，論理的にひとまとまりになっており，かつ最小単位であるような字句に切り分ける処理のことである [25, 15].

ここで考えなければいけないことは，Java ソースコードに限らず通常のプログラムのソースコードは，実際に実行する命令を記述する**プログラム部**と，その実行されるプログラムの

注釈を記述する**コメント部**のどちらかに区別できるということである．プログラム部はそのプログラムの記述言語の構文に従った字句解析を行えばよいが，コメント部の文法はそのプログラムの記述言語の構文とは無関係なので，語を抽出するための別の処理が必要となる．ここでコメント部は自然言語で書かれていることに着目すると，その自然言語の構文に従った形態素解析を行えば，語を抽出することができる．本研究は，Java ソースコードのコメント部が国際語でもある英語で書かれているものを対象とした．

以降で，プログラム部とコメント部の語の抽出について説明する．

4.2.1 プログラム部

Java ソースコードを対象としているので Java の文法に従って [7]，字句解析を行い語を抽出した．また抽出した語の位置から，語の種類も判別し保存した．以下の表 1 に字句解析の構文を載せる．処理には flex[15] を使った．

4.2.2 コメント部

本研究で対象とした Java ソースコードは，コメント部が英語で書かれているものとしている．英語は単語が空白により区切られているため，空白を利用して区切りごとに語を抽出した．

4.3 不要語の除去

Java ソースコードにおける不要語は，自然言語における不要語とは必ずしも一致しない．そこで Java ソースコードの性質を考慮した上で不要語を定義する必要がある．

以下で，プログラム部とコメント部に分けて，不要語とした語について説明する．

4.3.1 プログラム部

- **記号**

  ソースコードにおいて，記号というのは頻繁に使われる語である．またそれ自体に意味はなく，自然言語における機能語のような役割を果たす．よって記号を検索キーとして検索することはほとんどなく，仮にこれらの語を検索キーとして検索を行っても，ほとんど全てのソースコードが検索されてしまい必要な文書を絞りこむことができない．よって Java ソースコードのプログラム部から抽出した語の中で，記号であるものは不要語とした．

  ここでいう記号とは表 1 における「Symbol」のことであり，名前に使える “$”,“_” は記号に含まないとする．

JavaProgram	=	{Token | W-Space}
W-Space	=	Blank | Tab | NewLine | NewPage | Comment
Blank	=	空白
Tab	=	タブ
NewLine	=	改行
NewPage	=	改ページ
Comment	=	NormalComment | DocComment | LineComment
NormalComment	=	“/*” { Token | W-Space } “*/”
DocComment	=	“/**” { Token | W-Space } “*/”
LineComment	=	“//” { Token | Blank | Tab } (NewLine | NewPage)
Token	=	Identifier | Keyword | Number | String | Character | Symbol
Identifier	=	Letter{Letter | Digit}
Number	=	整数 | 小数 | 浮動小数点数 | 8 進数 | 16 進数
String	=	文字列
Character	=	文字
Letter	=	[A-Za-z_$]
Digit	=	[0-9]
Keyword	=	“abstract” | “assert” | “boolean” | “break” | “byte” | “case”
		| “catch” | “char” | “class” | “continue” | “default” | “do”
		| “double” | “else” | “extends” | “false” | “final” | “finally”
		| “float” | “for” | “if” | “implements” | “import” | “instanceof”
		| “int” | “interface” | “long” | “native” | “new” | “null”
		| “package” | “private” | “protected” | “public” | “return”
		| “short” | “static” | “super” | “switch” | “synchronized” | “this”
		| “throw” | “throws” | “transient” | “true” | “try” | “void”
		| “volatile” | “while” | “strictfp”
Symbol	=	“.” | “(” | “)” | “{” | “}” | “[” | “]” | “;” | “,” | “=” | “==”
		| “+” | “+=” | “>” | “<=” | “−” | “−=” | “<” | “>=” | “*”
		| “*=” | “!” | “!=” | “/” | “/=” | “∼” | “&&” | “&” | “&=” | “?”
		| “||” | “|” | “|=” | “:” | “++” | “^” | “^=” | “−−” | “%”
		| “%=” | “<<” | “<<=” | “>” | “>>=” | “>>>” | “>>>=”

表 1: 字句解析の構文

- **数字のみの語**

  自然言語を対象とした索引付けにおいて，数字は索引語としての重要度が低く，不要語とした方がいい場合がある．それをふまえソースコードにおける数字を考えてみると，ループ回数，int 型変数の初期値，等まさに様々な場面で使われており，その数字が出現した部品を特徴付ける能力が低いと考えられる．よって Java ソースコードのプログラム部から抽出した語で，数字のみで構成されているものは不要語とした．

ここでいう数字とは表 1 における「Number」のことである．

- **予約語**
  ソースコードにおいて，予約語はその単語本来の意味を表すものではなく，プログラムにおける機能の制御のために使われているにすぎない．

  また予約語というのはあらゆるソースコードに存在し，検索したい部品の絞り込みにも使えない．よって Java ソースコードのプログラム部から抽出した語の中で，予約語として使用されている語は不要語とした．

  ここでいう予約語とは表 1 における「Keyword」のことである．

- **１文字の語**
  １文字の語というのは意味を持つことがほとんどないので，検索キーとして入力される可能性が極端に低い．また変数等で使用する際も，一時的な変数として使うことが多くその文書の特徴を表すことがほとんどない．よって，Java ソースコードのプログラム部から抽出した語の中で，語の長さが１文字のものは不要語とした．

- **１文字の語＋数字**
  １文字の語は意味を持たないが，同じく１文字の語の末尾に数字が足された語も，意味を持たない．例えば変数として “a10” という語があったとしても “a” という１文字の語は意味を持たないので “a10” も意味を持たないと判断できる．よって，Java ソースコードのプログラム部から抽出した語の中で，１文字の語の末尾に数字が足された語は不要語とした．

4.3.2 コメント部

- **記号**
  プログラム部と同様の理由で不要語とした．

- **数字のみの語**
  プログラム部と同様の理由で不要語とした．

- **１文字の語**
  プログラム部と同様の理由で不要語とした．ここで，コメント部での大文字である１文字の語について考える．コメント部は自然言語で書かれており，大文字の語というのは文の最初の語であるか，固有名詞である語と考えられる．よって固有名詞というのはその文書を特徴付けるのに役立つので，大文字である１文字の語は不要語としなかった．

- **索引品詞リストに載っている品詞の語**

  コメント部は自然言語で書かれているので，コメント部から抽出した語には品詞が存在する．本研究では，Java ソースコードのコメント部は英語で書かれているものを対象としているので，各語の品詞情報によって不要語かどうかを決定した．

  実際には索引語として登録する品詞の一覧を載せたリスト，**索引品詞リスト**というものを作成し，そのリストに載っている品詞以外の語は不要語とした．

  索引品詞リストには以下の品詞を載せた．

  - 名詞 (代名詞は除く)
  - 動詞 (be 動詞は除く)
  - 形容詞
  - 副詞

  語の品詞を得るためには形態素解析が必要である．本研究では形態素解析のために TreeTagger というツールを利用した．このツールついては 4.5.1 節で紹介する．

4.4 索引語の拡張

検索システムでは，検索キーと一致する索引キーを探し出すことによって検索を行う．したがって，本来同一であるべき索引語が表現の多様性によって不一致を起こすと，検索すべき文書が検索できなくなってしまう [24].

この問題を解決するためのアプローチとして**検索質問拡張** (query expansion) がある．検索質問拡張とは，検索質問に新しい索引語を付け加える処理のことである [14]．例えば，検索キーとして「ダイアモンド」が入力されたとすると，索引語「ダイアモンド」に加え,「ダイヤモンド」も検索するといったものである．しかしこの処理は，検索キーが入力される検索時に行わなければならないので，検索時の処理が増えてしまい，検索結果を得るまでの時間が増えてしまうといった問題が生じる．

そこで本研究では，検索時に行う処理を減らすために検索質問として拡張される語を，あらかじめ索引語として登録する方法をとった．

以下，どのような索引語の拡張について説明する．

4.4.1 プログラム部

- **複合語の切り分け**

  ソースコードというものは，メソッド名等はそのメソッドの機能を表した名前である

ことが多い．しかし，そのメソッドの機能が一つの単語で表せないとき，二つ以上の単語を複合させて一つのメソッド名とすることも少なくない．よってこの複合語を切り分けて，本来の語だけでなく複合する前の単語をそれぞれ索引語として登録することを考える．

そこで，Java のソースコードのメソッド名等の付け方の特徴に注目すると，複合語内の単語の始まりの一文字は大文字で書き，残る後の文字は小文字で書くことが慣例となっている．この慣例を利用すれば，複合語内の単語の区切りを見つけることができる．また，名前をつける文字として利用できる “_”,“$” に注目すると，その前後も単語の区切りである．

よってこれらから，単語の区切りは以下の 3 箇所となる．

1. 文字列が，小文字から大文字になる箇所
2. 文字列が，大文字から小文字になる文字の 1 文字前の箇所
3. “_”, “$” の前後の箇所

複合語をこの区切り箇所で切り分けて，連接している単語の組合せを抽出していき，それらを新たに索引語として登録する．例として表 2 を載せる．

複合語	切り分けた単語	索引語
readInputFile	read, Input, File	read, readInput, readInputFile, Input, InputFile, File
addIDKey	add, ID, Key	add addID, addIDKey, ID, IDKey, Key
get_number	get, number	get, get_number, number

表 2: 複合語の切り分けの例

- **省略語リスト**

  前に述べたように，変数名等にはその変数の機能を表した名前をつけることが多い．しかしソースコード内には，慣習的にその機能を示した単語を省略した語を変数として使う時がある．例えば，一時的に値を保存するような変数の名前を考えるとき，その意味である「temporary（一時的な）」という単語をそのまま用いずに,「tmp」とすることの方が多い．このことから，検索キーとして「temporary」が入力された場合,「tmp」も同じ単語として認識して検索する方がよいと分かる．これらのことから索引語の拡張として，ある索引語が省略語ならば，その索引語の原形も索引語として登録するといった方法が考えられるが，効率の面を考えるとこの処理に限っては検索時に行った方がよい．よってこの省略語を原形に戻す処理のみ，検索時に行った．

省略語と原形の対応に関しては，それを載せた独自の省略語リストを作成した．このリストを参照し，入力された検索キーが省略語リストに載っている語なら，その省略語も検索するという処理を行った．この省略語リストの内容を表 3 に示す．

省略形	原形	省略形	原形	省略形	原形
arr	array	ans	answer	arg	argument
args	argument	bin	binary	buf	buffer
cat	concatenate	cls	class	char	character
cmd	command	cmp	compare	cmp	component
cnt	count	comp	compare	comp	component
col	column	config	configuration	conf	configuration
calc	calculate	chk	check	cp	copy
db	database	doc	document	dec	decrease
dev	device	dir	directory	diff	difference
eval	evaluation	etc	etcetera	err	error
exec	execute	exec	execution	esc	escape
flg	flag	func	function	hi	high
htm	html	int	integer	info	information
inv	invert	inc	increase	inc	include
inc	increment	init	initial	init	initialize
img	image	len	length	lab	laboratory
lib	library	lo	low	ls	list
ln	link	msg	message	mnt	mount
man	manual	num	number	no	number
obj	object	op	operation	pos	position
param	parameter	ps	process	qsort	quicksort
src	source	rm	remove	ref	reference
ret	return	str	string	std	standard
tmp	temporary	seq	sequence	usr	user
val	value	var	variable		

表 3: 省略語リスト

- **末尾の数字を除去**

  変数名，メソッド名等で同じ機能を持つものを２つ使いたい時に，末尾に番号を割り当てることもよくある．例えば，文字列を格納する変数を使う時，２つの変数の名前を「string1」,「string2」とする場合がある．しかしこれらの語をそのまま索引語とした場合，検索キーとして「string」と入力された時に，これら２つの変数に検索キーは一致せず，これら２つの変数が出現した部品は検索されない．よって，末尾に数字であるような語を索引語とする際に，その数字を消去した語も索引語として登録するようにした．

4.4.2 コメント部

- **複合語の切り分け**

  コメント部には，プログラムの説明を書く．よってプログラム部に現れる変数，メソッ

ド名等がコメント部にも現れる可能性が大きい．よってプログラム部で説明したような複合語がコメント部に現れる可能性も少なくないので，プログラム部で説明した複合語の切り分けをコメント部でも行った．

- **原形でない語を原形にする**

  コメント部は，説明したように自然言語で書かれている．自然言語は同じ単語でも場合によって変形して出現するものも多々ある．今回対象としている言語である英語も複数形，過去形，比較級，等非常に様々な変形が存在する．検索キーは一般的には原形で入力されるので，これらの変化している語は検索されない．そこで語形が変化している語は，原形に直した語も索引語として登録した．

  語の原形を得るための処理として，形態素解析が必要である．本研究では，この処理のために形態素解析ツール TreeTagger を使った．このツールの詳細は 4.5.1 節で述べる．

4.5 実装

4.5.1 形態素解析ツール TreeTagger

TreeTagger[19, 23] は，ドイツ Stuttgart 大学の Helmut Schmidt 博士が開発，公開している統計的手法を用いた形態素解析ツールである．実行形式で配布されており [20]，プラットフォームとして Sun/Solaris と Linux 用がある．本研究では，Linux 用 (最新バージョン 3.1) を利用した．

このツールは，品詞タグ付けの際にパラメータファイルを参照してから処理を行う．このパラメータファイルを入れ換えることによって英語以外の他言語にも適用できる．入力フォーマットに従って複数の単語を入力し実行すると，単語ごとにその品詞と原形を結果として返してくれる．品詞は Penn TreeBank Prject[31] のタグセットを用いている．

4.5.2 処理の流れ

処理の流れは以下の表 2 のようになる．

形態素解析に使用する TreeTagger は，一単語ごとに実行すると実行のオーバーヘッドが余計にかかる．よって前処理として品詞原形参照表を作成することで，１つのソースコードに対し TreeTagger の呼び出しを一回で済ませた．

以下，各 Java ソースコードに対し行う処理を説明する．

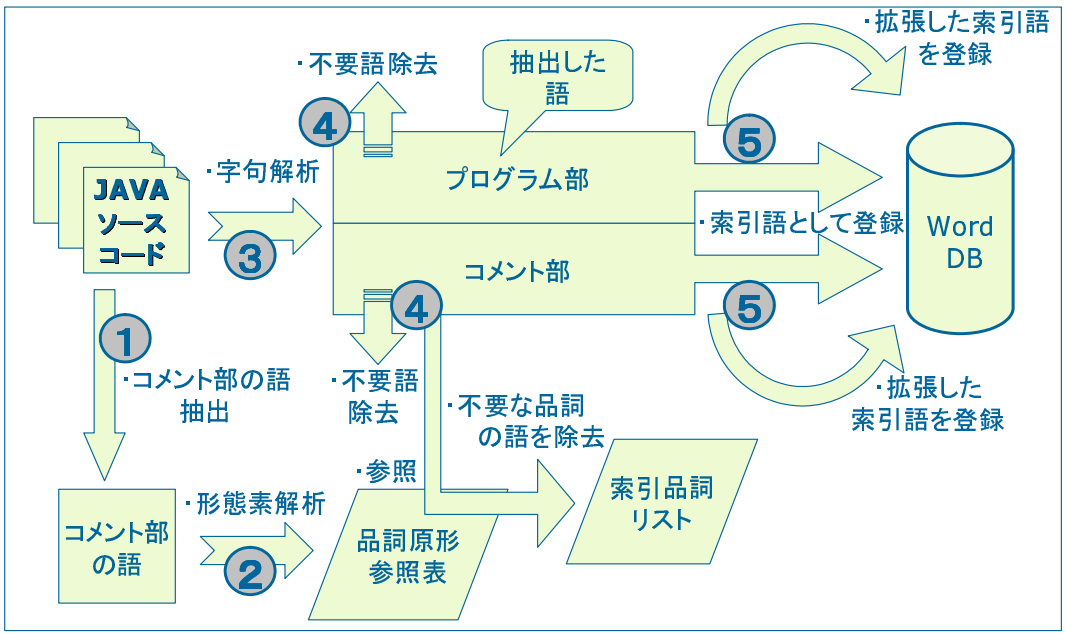

図 2: 処理の流れ

1. 前準備として，Java ソースコードからコメント部の全ての語を空白区切りで抽出する．
2. 抽出した語全てを TreeTagger に入力し，品詞と原形を求める．結果は品詞原形参照表に保存する．
3. 本処理として，字句解析を行い Java ソースコード内の全ての語を抽出する．
4. 抽出した語の中の不要語を除去する．この際コメント部の語は，品詞原形参照表を参照し品詞を確認し，その語が索引品詞リストに載っていない品詞なら不要語として除去する．
5. 残った語を索引語として登録する．拡張した索引語も登録する．

4.5.3 データベースに格納するデータ

索引に関するデータは WordDB に保存する．以下，索引情報を保存するリポジトリを説明する．

- **KeywordRepository**
  索引キーとワード ID を関連付けて保存するリポジトリ．索引語の大文字と小文字を

区別するリポジトリと区別しないリポジトリの 2 つを用意する．ワード ID とはシステム内で索引キーを識別するために使う ID である．

- **InvertedWordRepository**
  ワード ID と，その索引語が出現する部品 ID を関連付けて保存するリポジトリ．部品 ID とはシステム内で部品を識別するために使う ID である．

- **SourceRepository**
  部品ごとに，その部品に含まれる索引語のワード ID とその出現位置を保存するリポジトリ．

4.5.4 索引付けの例

実際に，提案手法によってどのような語が索引語とされるのか例を示す．サンプルの Java ソースコードを以下の図 3 に載せる．左端の < > に囲まれた番号は行番号である．

```
<1> /* Generate all possible solutions to the N queens problem.
<2>  * board[j] = i if there is a queen in row i of column j
<3>  *          = 0 otherwise
<4>  */
<5> class nQueens {
<6>    static int N = 90, a0, a1;
<7>    static char string1, string2;
<8>
<9>   static boolean safetyCheck(int row, int column, int[] board) {
<10> // Check whether it is safe to place a queen at row, column;
<11>       for (int j=1; j<column; j++) {
<12>          if (board[column-j] == row) {
<13>            return false;
<14>         }
<15>      }
<16>      return true;
<17>   }
<18> }
```

図 3: Java ソースコードの例

図 3 の Java ソースコードから提案した手法で索引語を抽出した結果を，以下の表 4 に示す．

この例から，プログラム部は “{” 等の記号，予約語，数字のみの語，１文字の語，１文字＋数字の語が全て不要語として除去されていることが分かる．コメント部は，索引品詞リ

行番号	索引語
1	Generate, possible, solutions, solution, N, queens, queen, problem
2	board, queen, row, column
3	otherwise
5	nQueens, Queens
7	string1, string, string2, string
9	safetyCheck, safety, Check, row, column, board
10	Check, safe, place, queen, row, column
11	column
12	column, board, row

表 4: 索引語の例

ストに載っていない品詞の語は全て不要語とされており，また数字や記号，１文字の語（小文字）も不要語として除去されていることが分かる．

索引語の拡張として，9 行目の複合語 “safetyCheck” は “safety” と “Check” に切り分けられ索引語とされていることが分かる．また，7 行目の “string1” に関しては，“string” も索引語となっている．また，コメント部の１行目の語 “solutions” は複数形なので，原形の “solution” も索引語として登録されていることが分かる．

5　手法の評価

5.1　評価の方針

提案した手法をまとめると以下のようになる．

- プログラム部

  手法 1. 記号を不要語として除去

  手法 2. 数字のみの語を不要語として除去

  手法 3. 予約語を不要語として除去

  手法 4. １文字の語を不要語として除去

  手法 5. １文字の語＋数字の語を不要語として除去

  手法 6. 複合語を切り分けて，切り分けた語も索引語とする

  手法 7. 検索時に省略語リストを参照して，省略語も検索

  手法 8. 語の末尾が数字である場合，数字を除去したものも索引語とする

- コメント部

  手法 9. 記号を不要語として除去

  手法 10. 数字のみの語を不要語として除去

  手法 11. １文字の語（大文字以外）を不要語として除去

  手法 12. 索引品詞リストに載っていない品詞の語は不要語として除去

  手法 13. 複合語を切り分けて，切り分けた語も索引語とする

  手法 14. 語形が変化している語は，原形も索引語とする

本節では提案手法を私が所属している研究チームで開発中の Java ソフトウェア部品検索システム *SPARS-J* に導入して適用実験を行い，提案手法の有効性を評価する．以降の節では，まず不要語が実際に検索キーとして入力されないのかを評価してから，以下の 3 点について提案手法を評価する．

1. データベース容量の変化
   - 手法 1〜手法 14 の比較
   - 索引語を拡張する手法のみと，索引語を拡張する手法＋不要語を除去する手法（全手法）の比較

2. 処理時間の変化

3. 再現率の変化

5.2 実験結果

5.2.1 不要語と検索キー

不要語が実際に検索キーとして入力されないのかを確かめるために，SPARS-J に入力された検索キー 2268 個を調べた．この検索キーは，ある８人のユーザがプログラム作成のために SPARS-J を利用した際に，入力されたものである．

その結果，入力された検索キーの中に，手法 1〜5，手法 9〜11 で除去した不要語はわずか 7 個であり，その割合は $(7/2268) \times 100 = 0.3$ % であった．それらの検索キーは，“import” と “[ ]” の２種類であった．“import” はほとんどの部品に存在していて絞り込みの役に立たない．また “[ ]” は，配列の使い方を調べるために入力された検索キーであり，SPARS-J はソフトウェア部品の再利用を目的としているシステムであり，Java の文法を調べるためのものではないので，“[ ]” は検索キーとして妥当ではない．よってこれらの検索キーが入力された場合でも，入力された語を不要語として除去しても問題ない．

また手法 12 について考えると，入力された検索キーは名詞（固有名詞含む）が 1854 個，動詞が 265 個，副詞が 2 個，形容詞が 140 個であり，索引品詞リストに載っていない語は 7 個であった．その割合は $(7/2268) \times 100 = 0.3$ % である．その 7 個の語は “out” と “Hello” の 2 種類であり，“out” は検索キーとして “system out println” と入力され，TreeTagger が前置詞と判断してしまったものであり，“Hello” は “Hello World” の “Hello” を感嘆詞と判断してしまったものである．しかしこの２つの検索キーは，明らかにプログラム部の語を目的として入力されたもの検索キーであり，コメント部にあるこれらの語は除去しても問題ない．

以上の結果と考察から，提案手法で定義した不要語を除去しても問題がない．

5.2.2 データベース容量

手法を何も行わず，字句解析した結果得られた語を全てそのまま索引語とする手法を手法 0 とする．手法 0 を基準としたとき，手法 1〜14 の索引語の総数の変化，データベースの容量の変化を以下の表 5 に示す．対象の Java ソースコードは，Netscape DevEdge[17] で公開されている Java ソースコード 261 個（部品数 281）である．

ここでいうデータベースとは，SPARS-J における Word DB のことである．

( ) 内の値は手法 0 と比較した時の値の差である．

	索引語総数 (個)		データベース容量 (Kbyte)	
手法 0	146982	(0)	8176	(0)
手法 1	79948	(-67034)	7920	(-256)
手法 2	144806	(-2176)	8032	(-144)
手法 3	134695	(-12287)	7952	(-224)
手法 4	142852	(-4130)	8048	(-128)
手法 5	146962	(-20)	8176	(0)
手法 6	188522	(41540)	12080	(3904)
手法 7	146982	(0)	8176	(0)
手法 8	149032	(2050)	8240	(64)
手法 9	138749	(-8233)	7872	(-304)
手法 10	146478	(-504)	8096	(-80)
手法 11	145368	(-1614)	8080	(-96)
手法 12	141041	(-5941)	7888	(-288)
手法 13	150844	(3862)	8672	(496)
手法 14	149468	(2486)	8320	(144)

表 5: 各手法でのデータベース容量の変化

手法 1 が最も索引語数が削減できている．ここから，Java ソースコードを占める記号の個数の割合が最も大きいと分かる．しかし，データベース容量でみるとあまり縮小できていない．これは，記号というのは語の長さが短いので，多くの個数を不要語として除去しても，容量的にあまり削減できないからだと考えられる．

データベースの容量については，他に手法 3, 9, 12 が手法 1 と同程度削減できている．

ちなみに手法 7 は検索時に省略語リストを読み込んで処理をするので，登録する索引語の数に増減はない．

次に，索引語を拡張する手法のみ行う場合と，全手法を行う場合を比較するため，５種類のデータベースを作成し，容量の変化を調べた．対象としたソースコードは，以下で公開されているものを使った．

1. DB1：Netscape DevEdge[17]（部品数 281）
2. DB2：Noel Enete[5]（部品数 511）
3. DB3：アスキーホームページ [1]（部品数 767）

4. DB4：Kossiakoff Computer Center Facility [30]（部品数 1120）

5. DB5：VB-Bookmark[9]（部品数 2844）

以下の表 6 に結果を示す.

DB1	索引語総数	手法 0 との比較	データベース容量 (KB)	手法 0 との比較
(1) 手法 0	146982		8176	
(2) 手法 6〜8, 13, 14	196998	134.0%	12752	156.0%
(3) 手法 1〜14	93875	63.9%	11344	138.7%
(3)-(2)	-103123	**-70.2**%	-1408	**-17.2**%
DB2				
(1) 手法 0	234890		9440	
(2) 手法 6〜8, 13, 14	318995	135.8%	14608	154.7%
(3) 手法 1〜14	147271	62.7%	12224	129.5%
(3)-(2)	-171724	**-73.1**%	-2384	**-25.3**%
DB3				
(1) 手法 0	571536		17664	
(2) 手法 6〜8, 13, 14	729874	127.7%	25760	145.8%
(3) 手法 1〜14	357706	62.6%	22544	127.6%
(3)-(2)	-372168	**-65.1**%	-3216	**-18.2**%
DB4				
(1) 手法 0	371595		17456	
(2) 手法 6〜8, 13, 14	479309	129.0%	25584	146.6%
(3) 手法 1〜14	234752	63.2%	21728	124.5%
(3)-(2)	-244557	**-65.8**%	-3856	**-22.1**%
DB5				
(1) 手法 0	3484982		84384	
(2) 手法 6〜8, 13, 14	4862607	139.5%	141968	168.2%
(3) 手法 1〜14	2452671	70.4%	124480	147.5%
(3)-(2)	-2409936	**-69.2**%	-17488	**-20.7**%
(3)-(2) の平均		**-68.7**%		**-20.7**%

表 6: 各データベースの容量の変化

表 6 を見ると，全てのデータベースにおいて索引語数は約 70 %，データベース容量は約 20 %減少している．このことから，不要語の除去によりデータベースの規模に関わらず容量の増加を抑えることがをできるといえる．また索引語数の面だけ見ると，提案手法を導入すると導入前より索引語数は減少している．

5.2.3 処理時間

5.2.2 節で各手法でのデータベースの容量の変化を調べたときと同じ方法で，各手法の処理時間を比較した．結果を以下の表 7 に示す．

	処理時間 (s)	
手法 0	17.909278	0
手法 1	10.78736	(-7.121918)
手法 2	17.580327	(-0.328951)
手法 3	16.886433	(-1.022845)
手法 4	18.055255	(0.145977)
手法 5	17.731305	(-0.177973)
手法 6	22.49558	(4.586302)
手法 7	18.102248	(0.19297)
手法 8	18.159239	(0.249961)
手法 9	17.69531	(-0.213968)
手法 10	18.291219	(0.381941)
手法 11	18.238228	(0.32895)
手法 12	21.855677	(3.946399)
手法 13	19.476039	(1.566761)
手法 14	22.422592	(4.513314)
手法 6〜8, 13, 14	26.284005	(8.374727)
手法 1〜14	14.911733	(-2.997545)

表 7: 各手法での処理時間の変化

手法 1〜14 の各手法を比較すると，最も処理時間が短いのは手法 1 である．理由として，手法 1 では索引語数が全手法中最も少ないため，索引語をデータベースに登録する処理が少なくて済むことが挙げられる．

逆に手法 12, 13, 14 は索引語拡張の処理をしなければならないため，処理時間が少し増えてしまっている．

また，不要語を除去せずに索引語を拡張する (手法 6〜8, 13, 14) ときは，不要語を除去してから索引語を拡張するとき (手法 1〜14) と比較して，$(14.911733(s)/26.284005(s)) \times 100 = 56.7\,\%$ と，極端に早くなっている．手法 1〜14 を適用した場合と何も行わなかった場合 (手法 0) と比べても $(14.911733(s)/17.909278(s)) \times 100 = 83.3\,\%$ と，提案した手法が処理時間

の面で効果的である．

5.2.4 再現率

完全性を評価する尺度として再現率がある．再現率は，

$$\text{再現率} = \frac{\text{検索された中の適合部品の数}}{\text{全部品の中の適合部品の数}}$$

で定義される値である．

本節では，5.2.2 節で用いた Netscape DevEdge[17] で公開されている Java ソースコード 261 個（部品数 281）に対してデータベースを作成し，５つの検索キーを想定して検索を行い，再現率を調べることにより評価を行った．

この５つの検索キーは，データベースに格納されている部品をある程度知っているユーザが，ある目的で検索システムを使ったという状況を考えて想定した．想定した検索キーは以下のものである．

	検索キー	目的としている部品
K1	set bar	バーの設定を行っている部品
K2	initialize window	ウィンドウを初期状態に戻す処理が記述されている部品
K3	font size	フォントのサイズ関係のことが記述されている部品
K4	frame update	新しいフレームにアップデートする際の処理が記述されている部品
K5	mouse event	マウスの操作で起こるイベントが記述されている部品

表 8: 想定した検索キー

また，全部品中の適合部品の数を調べなければならない．そこで，全部品中で検索キーが含まれる部品の数を grep で調べた．また，その検索キーの語形が変化して出現している部品も適合部品である．よってこのような部品の数も grep で調べた．しかし，その中で検索キーの本来の意味で使われていないものは適合部品から除外した．

再現率を調べたのは，手法 0，手法 6，手法 7，手法 8，手法 13，手法 14，5 つの手法全てを適用，の各場合についてである．以下の表 9 に各場合の検索された部品数を示す．表内の () 内の値は再現率である．これらは全て AND 検索で検索した．

	適合部品数	手法 0	手法 6	手法 7	手法 8	手法 13	手法 14	5 つの手法全て
K1	13	4 (0.31)	10 (0.77)	4 (0.31)	4 (0.31)	5 (0.38)	5 (0.38)	10 (0.77)
K2	13	3 (0.23)	3 (0.23)	7 (0.54)	3 (0.23)	3 (0.23)	3 (0.23)	13 (1.00)
K3	12	8 (0.67)	9 (0.75)	8 (0.67)	8 (0.67)	9 (0.75)	8 (0.67)	9 (0.75)
K4	9	5 (0.56)	8 (0.89)	5 (0.56)	6 (0.67)	6 (0.67)	5 (0.56)	9 (1.00)
K5	14	4 (0.29)	14 (1.00)	4 (0.29)	4 (0.29)	4 (0.29)	4 (0.29)	14 (1.00)

表 9: 再現率の変化

手法 0 と，提案手法を適用した場合を比較すると，5 種類全ての検索キーで再現率が大幅に上昇している．

また検索キーごとの比較をすると，K1，K4，K5 のように，長さが短い動詞の語が検索キーに含まれている場合，手法 6 によって効果的に再現率が上昇した．これは，複合語が主に検索洩れの原因になっており，また複合語には語の長さの短い動詞が含まれていることが多いことを示している．

しかし，語の長さが短い動詞の語といったように，複合語に含まれがちな語が検索キーに含まれていない K2，K3 のような場合，手法 6 ではあまり再現率が上昇しない．

また，K2 のように検索キーに “initialize” のような省略語の原形が含まれている場合，手法 7 によって再現率の上昇が期待できる．

また検索キー K1 と K3 については，提案手法を導入したが再現率が 1 にはならなかった．これは，K1 では複合語 “scrollbar” が切り分けられず検索キー “bar” で検索されなかったり，K3 では複合語 “resize” が検索キー “size” で検索されなかったりと，提案した複合語の切り分けのアルゴリズムでは切り分けられなかった語が検索されず，検索洩れの原因となったと考えられる．

5.3 評価のまとめ

まず 5.2.1 節で，提案手法で定義した不要語が，検索キーとして入力されないかどうかを調べた．その結果，検索キーとして入力された不要語は，検索キー 2268 個のうちの 7 個と，全体の約 0.3 %程度であった．また入力されたそれら 7 個の検索キーも，索引語として必要ないことを考察した．

次に 5.2.2 節で，不要語を除去することにより提案手法ではデータベースがどのように変化するかを，５種類のデータベースを作成し調べた．その結果，索引語の拡張のみを行うと

約 150 %データベースが増加するが，不要語を除去することにより約 130 %で済み，不要語を除去することによりデータベース容量の増加を約 20 %抑えられた．

また 5.2.3 節で，部品数 281 個のソースコードに対しデータベースを作成し，処理時間を調べた．その結果，提案手法を導入することによって登録する索引語数が減り，処理時間が約 18 秒から約 15 秒に抑えられた．

最後に，5.2.4 節で検索洩れをどれくらい減らすことができるのかを評価するために，５つの検索キーを想定し再現率を調べた．その結果，５種類全ての検索キーで再現率が上昇していた．また，５つの検索キーで比較を行い，複合語の切り分けが効果的に検索洩れを減らせることを考察した．しかし検索キーによっては，全ての適合部品を検索することはできなかった．

以上の結果から提案手法は，データベースの容量の増加を割合を減らし，かつ検索システムの検索洩れを減らすことができた．しかし，完全に検索洩れがなくなるわけではなく，提案手法に改善の余地があることが明らかになった。またその他に，処理時間の面でも効果的であった．

6　まとめと今後の課題

本研究では，Java ソフトウェア部品検索システムのための索引付け手法の提案と実装を行なった．

この手法は，既存の索引付け技術をふまえた上で，検索システムの対象が Java ソースコードであることを利用した索引付けを行なう．まず複合語の切り分け等の索引語の拡張を行ない，検索洩れを減らす．しかし索引語の拡張を行なえば索引語の数が増え，データベースの容量が増加する．そこで Java ソースコードにおける不要語を定義し，索引語から除去することによりデータベースの容量の増加を抑える．

また，提案手法を SPARS-J に適用し，データベースの容量の変化と再現率の変化を調べることにより，提案手法を評価した．まず不要語が検索キーとして入力されないこと確認した上で，不要語を除去する場合と除去しない場合を比較した．その結果，データベースの容量の増加を約 20 %抑えられることを確認した．また，ある状況を想定し検索キーを入力して，検索される部品数から再現率を計算し比較した．その結果，提案手法を導入することにより，5 つの検索キー全てにおいて再現率が上昇していることを確認した．これらの結果から，提案手法の目的が達成されていることを示した．

今後の課題として，提案手法では検索洩れを完全になくすことがきなかったことから，より検索洩れが減らせるような索引語の拡張が望まれる．他には，コメント部が日本語である場合の提案手法の導入や，他言語への提案手法の導入が挙げられる．

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 井上 克郎 教授に深く感謝致します.

本研究において, 適切な御指導および御助言を頂きました 同 楠本 真二 助教授に深く感謝致します.

本研究において, 適切な御指導および御助言を頂きました 同 松下 誠 助手に深く感謝致します.

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 独立行政法人科学技術振興機構 山本 哲男 氏に深く感謝致します.

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 横森 励士 氏に深く感謝致します.

本研究において, 御協力を頂きました Stuttgart 大学 Helmut Schmid 氏に深く感謝いたします.

最後に, その他様々な御指導, 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深く感謝いたします.

参考文献

[1] ASCII Corporation, “アスキーホームページ”, http://www.ascii.co.jp/.

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

[3] 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).

[4] C. Braun: Reuse, in John J. Marciniak, editor, Encyclopedia of Software Engineering, Vol. 2, John Wiley & Sons, pp. 1055-1069, 1994.

[5] N. Enete and D. Enete, “Noel Enete”, http://www.enete.com/.

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

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

[8] G. Hamilton: JavaBeans 1.01 Specification, Sun Microsystems (1997)

[9] ICE Engineering, Inc., “ICE Engineering”, http://www.trustice.com/.

[10] 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).

[11] S. Isoda: “Experience report on a software reuse project: Its structure, activities, and statistical results”, in Proceedings of 14th International Conference on Software Engineering (ICSE14), pp.320-326, Melbourne, Australia, 1992.

[12] 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).

[13] B. Keepence and M. Mannion: “Using patterns to model variability in product families”, IEEE Software, Vol. 16, No. 4, pp. 102-108, 1999.

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

[15] J. R. Levine, T. Mason, D. Brown 共著, 村上 列 訳: “lex & yacc”, アスキー出版局, 2001.

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

[17] Netscape, “Netscape DevEdge”, http://developer.netscape.com/.

[18] 西 秀雄: “Java ソフトウェア部品検索システム SPARS-J の構築”, 大阪大学大学院情報科学研究科修士学位論文, February (2004).

[19] H. Schmid: “Probabilistic part-of-speech tagging using decision trees”, In International Conference on New Methods in Language Processing, Manchester, UK, 1994.

[20] H. Schmid,“TreeTagger”, http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html.

[21] 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).

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

[23] 田中: “形態素解析ツール -英語と TreeTagger を中心に-”, 九州大学 情報基盤センター研究部 外国語情報メディア研究部門

[24] 徳永: “情報検索と言語処理”, 東京大学出版会, 2002.

[25] 辻野: “コンパイラ”, 昭晃堂, 2001.

[26] 梅森 文彰: “Java ソフトウェア部品検索システム SPARS-J の実験的評価”, 大阪大学大学院情報科学研究科修士学位論文, February (2004).

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

[28] 鷲崎, 深澤: “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”, オブジェクト指向 2003 シンポジウム (OO2003), (2003).

[29] 横森, 藤原, 山本, 松下, 楠本, 井上: “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), and Technical Report of SE Lab, Dept. of Computer Science, Osaka University, SEL-Nov-21-2002, Nov (2002).

[30] “Kossiakoff Computer Center Facility”, http://www.apl.jhu.edu/.

[31] "The Penn Treebank Project", http://www.cis.upenn.edu/~treebank/.